SunGuide®:

Quality Assurance Plan

SunGuideSMD-QAP-1.0.0 (Working Final)

Prepared for:

Florida Department of Transportation
Traffic Engineering and Operations Office
605 Suwannee Street, M.S. 90
Tallahassee, Florida 32399-0450
(850) 410-5600

August 19, 2010

Document Control Panel			
File Name:	SunGuideSMD-QAP-1.0.0(WorkingFinal).docx		
File Location:	SunGuide CM Repository		
	Name	**Initial**	**Date**
Created By:	Robert W. Heller	RWH	2010.07.16
Reviewed By:	Tucker Brown	TJB	2010.07.29
	Tucker Brown	TJB	2010.08.18
	Ken Irvin	KDI	2010.08.18
Modified By:	Robert W. Heller	RWH	2010.08.16
Completed By:			

Table of Contents

List of Tables

List of Figures

List of Acronyms

CMMI	Capability Maturity Model Integration
CMB	Configuration Management Board
CONOPS	Concept of Operations
DMS	Dynamic Message Sign
FAT	Factory Acceptance Testing
FDOT	Florida Department of Transportation
ITS	Intelligent Transportation Systems
ITN	Invitation to Negotiate
LOA	Letter of Authorization
PM	Project Manager
QA	Quality Assurance
QAP	Quality Assurance Plan
QAR	Quality Assurance Representative
SICP	Software Integration Case Procedure
SMD	Support Maintenance and Development
SDP	Software Development Plan
SWA	Standard Written Agreement
SwRI	Southwest Research Institute

Revision History

Revision	Date	Changes
1.0.0 (Draft)	July 29, 2010	Initial Release.
1.0.0 (Working Final)	August 19, 2010	Revised in response to FDOT comments.

1. Scope

1.1 Document Identification and Purpose

This document serves as the Quality Assurance Plan (QAP) for the Florida Department of Transportation (FDOT) SunGuide® Support, Maintenance and Development (SMD) contract. This plan addresses how Southwest Research Institute® (SwRI®) will ensure that the source code and documentation delivered are of high quality and acceptable to the Department. This QAP includes

- Description of Quality Assurance Surveillance, the purpose of the QAS, and the activities that occur during those QAS,
- Description of the purpose of internal peer reviews, how those reviews contribute to the overall quality of the delivered product, and a description of the types of peer reviews that occur within the project, the frequency of the reviews and the types of reviews selected for use are documented in the Software Development Plan (SDP),
- Testing levels and methods, including type of testing, how testing progresses from one level to the next and the contribution that testing makes to the quality of delivered products,
- Coding standards are described here and the contribution that adherence to coding standards make to the quality of delivered products.

FDOT may impose additional specific quality issues identified in a Letter of Authorization (LOA).

1.2 Project Overview

The Florida Department of Transportation (FDOT) SunGuide Support, Maintenance and Development Contract, contract number BDQ69, addresses the necessity of supporting, maintaining and performing enhancement development efforts to the SunGuide software. The SunGuide software was developed by the FDOT in a contract from October 2003 through June 2010. The SunGuide software is a set of Intelligent Transportation System (ITS) software that allows the control of roadway devices as well as information exchange across a variety of transportation agencies and is deployed throughout the state of Florida. The SunGuide software is based on ITS software available from the state of Texas; with significant customization and development of new software modules to meet the needs of the FDOT. The following figure provides a graphical view of the SunGuide software architecture:

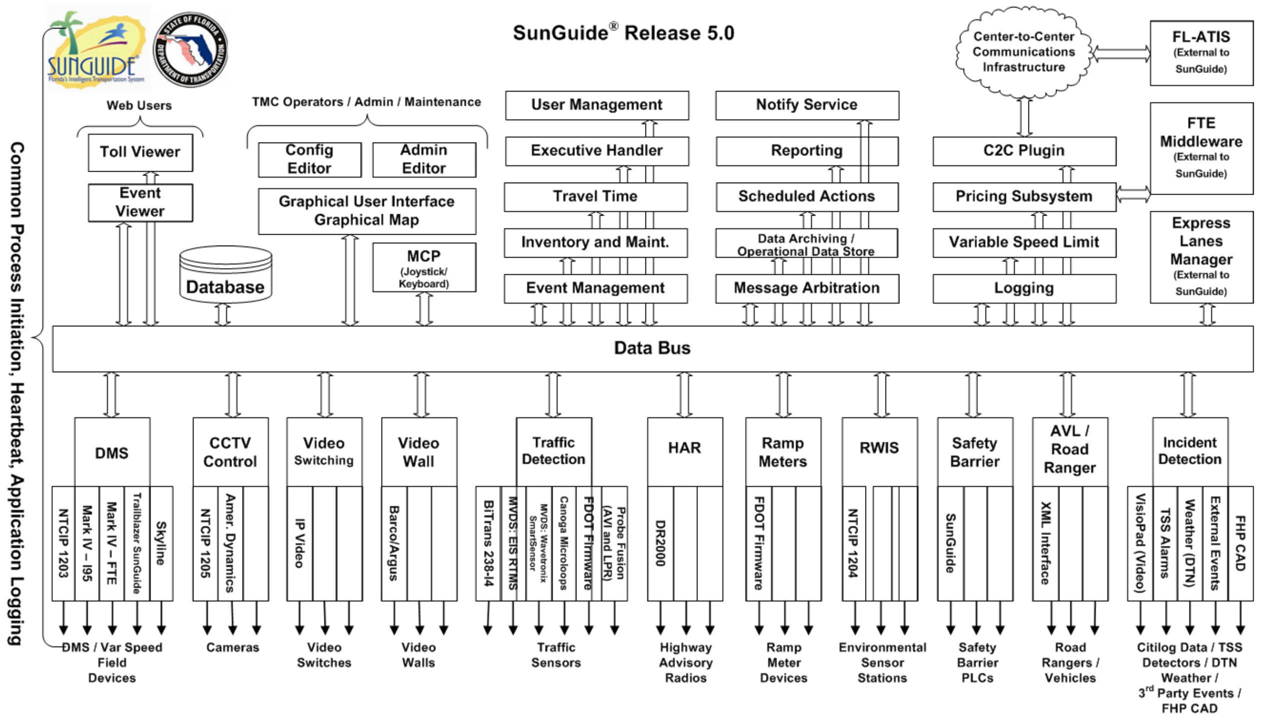

Figure 1-1 – High-Level Architectural Concept

1.3 Related Documents

Additional information regarding the SunGuide project can be found in the following documents and electronic publications:

- FDOT Scope of Services: *BDQ69, Standard Written Agreement for SunGuide Software Support, Maintenance, and Development, Exhibit A: Scope of Services*. July 1, 2010.
- Notice to Proceed: Letter to SwRI for BDQ69, July 1, 2010
- Letter of Authorization 001: Letter to SwRI for BDQ69, July 1, 2010.
- SunGuide Project website: http://sunguide.datasys.swri.edu.
- SunGuideSMD Software Development Plan, http://sunguide.datasys.swri.edu.
- SunGuideSMD Project Staffing Plan, http://sunguide.datasys.swri.edu

1.4 Contacts

The following are contact persons for the SunGuide software project:

- Elizabeth Birriel, ITS Section, Traffic Engineering and Operations Office, elizabeth.birriel@dot.state.fl.us, 850-410-5606
- Arun Krishnamurthy, FDOT SunGuide Project Manager, Arun.Krishnamurthy@dot.state.fl.us, 850-410-5615
- Khue Ngo, PBS&J Project Manager, khue.ngo@dot.state.fl.us, 850-410-5579.
- David Chang, PBS&J Project Advisor, David.Chang@dot.state.fl.us, 850-410-5622
- Robert Heller, SwRI Project Manager, rheller@swri.org, 210-522-3824
- Tucker Brown, SwRI Software Project Manager, tbrown@swri.com, 210-522-3035

2. Quality Assurance Surveillances

The SwRI QA Representative (QAR), identified in the SunGuide SMD Staffing Plan, performs surveillances of the SwRI SunGuide project. The purpose of the surveillance is to document compliance and identify non-compliance of the project with the organization process. Compliance with the organization process ensures enhanced product to the FDOT as it requires the development of this plan (QAP), which requires identification of Software Development Life Cycle, testing, coding standards, peer reviews, etc. SwRI provided a summary of the organization process to the FDOT in the SwRI response to the SunGuide SMD Invitation to Negotiate; a copy of the process is not available to the FDOT.

2.1 Initial Surveillance

The QAR meets with the SwRI PM (or SPM) and conducts an initial project surveillance within 30 days of contract award. The QAR utilizes the initial project surveillance checklist[1] (see Attachment A) to verify compliance with the organization process. Items of non-compliance are documented within the SwRI QA system and tracked to closure.

2.2 Periodic Surveillances

The QAR meets with the SwRI PM (or SPM) and conducts periodic project surveillances every 90 days. The QAR utilizes the periodic project surveillance checklist (see Attachment A) to verify compliance with the organization process. Items on non-compliance are documented within the SwRI QA system and tracked to closure.

[1] The checklists provided contain references to many internal SwRI processes, procedures, documents and forms. Providing copies of those is beyond the scope of this document.

3. Coding Standards

3.1 Java Coding Standard

SwRI will use the Sun Java coding standard that may be found at the following URL:

http://java.sun.com/docs/codeconv/CodeConventions.pdf

Historically, SwRI maintained its own Java coding standard, it was based on the Sun standard and eventually morphed into that standard. The Sun standard is considered the "industry standard" today.

3.2 C# Coding Standard

The SwRI coding standard in use during the development of the SunGuide system is in Attachment B of this document.

In addition to the coding standards, SwRI uses ReSharper plug-in for Visual Studio. ReSharper assists development of C# projects with continuous code analysis for errors and suggestions for code optimizations.

- ReSharper also allows for customizations to the suggestions to include parts of the coding standard. For instance,
- ReSharper can look at method names to ensure they use camel case (or another method consistent with the coding standard) and flag the method if the method name does not meet the standard.
- ReSharper also checks comments of methods to make sure they exist and also spell check all comments. Suggestions for changes are made known through a vertical bar next to the scroll bar that contains tick marks where suggestions are made in the file. These marks are color coded so that a user can open a file identify critical comments (e.g. compilation errors) from suggestions (e.g. comment had spelling mistake).
- ReSharper also helps clean up code by flagging unused methods and segments of code that are unreachable or unnecessary (i.e. a function has an embedded return statement so the rest of the code will never be executed).

3.3 Other

Other programming languages are used within the SunGuide system.

- In addition to using C# in the coding of the Operator Map, SwRI also makes use of "Java Script." Though there are minor differences between Java and Java Script, SwRI staff follow the Java coding standard when writing Java Script.
- PL/SQL is used extensively in the database in support of storing of events. The coding of PL/SQL follows the example coding published by Oracle. SwRI programmers make modifications to the existing PL/SQL in a manner consistent with the existing code.

4. Peer Reviews

The objective of a review is to involve peers, management, and customers to examine a portion of or an entire baseline item to discover defects. The results of the review are accumulated, consolidated and recorded. At the conclusion of the review, the results are provided to the author(s) and the author is responsible for resolving the defects. It is apparent from this process that the goal is to produce a product with improved quality and reduced defect count.

4.1 Review Types

The following paragraphs describe the different types of reviews that may occur in the review of work products developed for the FDOT.

*4.1.1 **Management Review***

SwRI utilizes Management Reviews for documents developed for the FDOT. In a management review, a non-author member of the SwRI management team (usually the Section Manager responsible for the project) reviews the document, capturing comments within the document being reviewed. The author reviews the comments and corrects any defects. If there are noted defects or comments with which the author does not agree, the author shall discuss those with the reviewer and reach agreement for resolution.

*4.1.2 **Buddy Check Peer Reviews***

The goals of a Buddy Check are to identify defects and issues in a work product, to point out needed improvements and to increase uniformity among work products (e.g. was the appropriate coding standard followed) within a project and the Organization. Once a work product is ready to undergo a buddy check, perform the following steps. The following shall be performed during the review.

- The author identifies a “buddy” to review the work product. The term “buddy” is not often defined other than it typically meets the following criteria: the buddy is familiar with the domain, the buddy is a SwRI employee approved to work on the project, the buddy has similar project responsibilities, there is not “reporting” relationship between the buddy and the author (neither reports to the other in the SwRI management ladder).
- The “buddy” shall review the work product.
- The “buddy” shall capture comments in the work product.
- The author shall review the comments created by the “buddy”.
- The author shall address the comments and recommendations. The author shall correct those items that the author agrees are defects if they will not impact project cost or schedule. The author shall work with the “buddy” to address items that the author believes are not actually defects and do not require a correction.

SwRI conducts Buddy Check Peer Reviews in a non-attribution manner as much as possible. In keeping with this policy, the reviews are closed to project management, organization management and the FDOT. Furthermore, the results of these reviews are not shared with the customer.

4.1.3 Walk-Through Peer Reviews

SwRI conducts formal walkthroughs in a lecture style format to reviewers who have not previously seen the work product, i.e. the author leads (walks) the reviewers through the work product during the review. SwRI conducts Walk-Throughs with four major roles identified:

- Work Product Author: This is the primary author of the work product or a staff member identified to represent a product which may be the result of a team development effort;
- Peer Review Leader: A SwRI staff member trained in leading peer reviews (there is formal training), familiar with the domain and designated by the PM or SPM to lead this review,
- Recorder or Scribe: Is a member of the review team chosen to record the results of the review session. The recorder may be the author, but that is typically not done as it limits the ability of the author to listen.
- Peer Reviewers: Staff familiar with the domain of the work product and not the PM or members of SwRI management.

The following shall be performed during the review.

- The author of the work product being reviewed shall present the work product in a methodical order
- The author shall describe the work product in a lecture-type presentation.
- Reviewers shall take notes as the author presents the work product.
- As defects/issues are identified (e.g. adherence to coding standards), they shall be discussed, classified by severity, and logged by the recorder.
- The number of defects/issues shall be summarized and the review leader and/or the reviewers determine if a re-review is necessary.
- At the conclusion of the review, the author shall collect the notes from the team members and the defect/issue log from the recorder.

Following the review, the defect log is provided to the author, the author addresses the defects, the review leader ensures that the logged defects are addressed; the results are stored according in the project repository.

SwRI conducts Walk-Through Peer Reviews in a non-attribution manner as much as possible. In keeping with this policy, the reviews are closed to project management, organization management and the FDOT. Furthermore, the results of these reviews are not shared with the customer.

4.1.4 Customer Reviews

At times required by an LOA, SwRI will participate in Customer Reviews. The customer reviews that may be required by the SunGuide SMD contract include

- CONOPS Reviews often by the SunGuide Configuration Management Board (CMB),
- Requirements Reviews,
- Preliminary Design Reviews,
- Detail Design Reviews,
- Physical Configuration Audits.

These are described in more detail in the contract BDQ69 Attachment A “Scope of Services.”

4.2 Selection of Review Types

Selecting the appropriate review method requires selecting a criterion such as priority, complexity or risk. For example, a portion of design is considered complex because it has many external interfaces. Therefore, that portion of the design will undergo a formal technical review versus a walkthrough or buddy check.

4.2.1 Support

SwRI will perform Buddy Checks on all code modifications that result of support calls or footprints issues (Defects, aka Bugs).

4.2.2 Enhancement

Selecting the appropriate review method depends on a number of subjective criteria.

- Complexity: a portion of design is considered complex because it has many external interfaces, or the interfaces are not well defined or are in development concurrently with the modification to SunGuide.
- Size: Small, medium and large are terms that refer to numbers of labor hours, but also are mitigated by the size of the development team, SunGuide experience of the developers, and if the task is repetitive in nature. The terms are left undefined and as noted above are subjective in nature.
- The terms complexity and size refer to the actual modification being considered, not the overall size of a Release. That is a release may be Major or Moderate but consist of many parts that are small in size (Footprints Enhancement Requests that are independent of each other). It is the size of this Work Product that is to be considered.
- SwRI conducts more Buddy Checks and fewer Walk-Throughs depending on the experience level of the developer.
- SwRI has not used Formal Technical Reviews in the SunGuide project historically and does not anticipate using them in the current contract.

4.2.2.1 Small Enhancement

Typically, the SunGuide SMD Project Team conducts “Buddy Checks” for small enhancements. In this situation, a minor enhancement is one that is small in labor hours, or well understood. One that is “well understood” is one that may require a moderate amount of labor hours but repetitive in nature (making the same modification in multiple places throughout the system).

4.2.2.2 Moderate / Large Enhancement

Typically, the SunGuide SMD Project Team conducts Walk-Throughs for larger enhancements.

4.3 Review Plan

Incorporated in each SDP LOA update will be a table similar to that shown in Table 4.1. For each work product, the table identifies Peer review method, Rationale, and method of reporting and tracking the results of the reviews.

Table 4.1 - Work Product Review Plans

Baseline Item or Portion of Baseline Item	Review Type				Rationale	Reporting and Tracking Method for Review Results
	Buddy Check	Walk-through	Management Review	Customer Review		
Work Product Name	☐	☐	☐	☐		

5. Testing

It is generally acknowledged throughout the software development industry that money spent is software testing more than pays for itself in the lifetime of a software product (this is going to be said without reference). SwRI performs software testing to expose and correct as many defects in the software to be delivered as possible. Elimination of the defects by definition provides a better product to the FDOT.

5.1 Unit Testing

Once the code is progressed to a stable state and developers believe the code is ready for unit testing, automated tests are written to test classes, methods, algorithms, and other sections of code that should be tested on a small scale. Tests are written in test cases that will traverse all code paths within the segment, including error cases. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct and behave in the manner in which they were intended. Unit testing creates a bottom-up testing style approach and makes integration testing much easier. These tests add additional cost but usually end up saving money in the end by catching fundamental problems much earlier in the development process.

5.2 Integration Testing

Integration Testing combines the smaller modules validated in the Unit Testing procedure and tests them as a group. Integration Testing is performed in a Top-Down Testing approach where the top integrated module (i.e. module in the group upon whose execution is dependent on other modules within the group) is tested forcing the branches of the sub-level modules to be tested step by step until reaching the end of the top-level module. Between Integration Testing and Unit Testing, all parts of the modules should receive sufficient testing.

5.2.1 New Tests

With new developments and enhancements to existing code, new functionality emerges and needs new test cases to ensure functionality. As part of this process, new requirements are tested in an automated test, if possible, and a manual test. Automated tests are written so that they can be run multiple times with minimal effort. Automated tests also allow these test cases to be incorporated with automated regression tests for future releases. Manual tests are then performed on all requirements to ensure they meet the exact wording of the requirement. As part of manual testing, the Software Integration Case Procedures (SICP) document is written. The manual tests steps will also be performed at Factory Acceptance Testing (FAT) but need to be well exercised in advance of FAT.

5.2.2 Regression Testing

After the completion of Integration Testing, it is extremely important to determine if the "untouched" parts of an enhancement continue to perform as expected.

- Both automated and manual testing is required in order to do regression testing.
- Automated tests are generally made up of the Unit and Integration testing written during the development of that feature.
- Tests are re-run to ensure the functionality behaves as expected.
- Tests become obsolete due to enhancements to the system. In these cases, the tests are removed or updated to reflect the current expected behavior.

SunGuide has an extensive user interface so regression testing is performed on device interfaces (device control user interfaces) and event management. These tests are performed by developers performing the common operator tasks. Deviations from expected behavior are addressed as they are found during testing.
The Admin Editor is also tested before every release. A tester will manually perform every task available in the Admin Editor to ensure the functionality was not broken by an enhancement. The Admin Editor exercises the SunGuide subsystems database access (a change to a Dynamic Message Sign (DMS) in the Admin Editor relies on the DMS subsystem to change the entry in the database).

5.3 Factory Acceptance Testing (FAT)

The intention of FAT is to demonstrate to the FDOT that the software meets the FDOT approved requirements. As noted earlier, as part of FAT, an SICP is written in order to manually test the requirements of the system in FDOT’s presence. The SICP contains the software requirements, test steps, and traceability of the software requirements to system requirements. In preparation for FAT, SwRI executes multiple dry runs of the SICP on a test system similar to an operator workstation from a Traffic Management Center. Each dry run of the SICP is documented; i.e. which tests passed and which failed. Failed tests are addressed by developers and a new dry run is executed against the SICP. This procedure continues until all tests are run successfully. Additionally, SwRI will perform additional testing against the SICP for particular sections if we feel the testing may not have covered all test conditions (e.g., threading issues, device load). Once the dry runs are completed the testing is run with FDOT present and each test in the SICP must be marked by an FDOT representative as passed or failed. At the end of FAT, this test record is kept by SwRI and digital copies are made available to FDOT and posted to the SunGuide web site.

Attachment A
Quality Assurance Surveillance Checklists

INSTITUTE QUALITY SYSTEMS
INITIAL SURVEILLANCE CHECKLIST

Project Number:	Project Manager:	
Project Start Date:	Project End Date:	Date(s) Conducted:
General Description of New Project:		

Project Information from the Proposal

01	The general project information (above) is correct? [modify if otherwise]	☐ S	☐ P	☐ U	☐ N/A
02	A PDP View was selected for the project? [enter the view selected _______]	☐ S	☐ P	☐ U	☐ N/A
03	A Project Profile Sheet (PPS) was completed and filed? *(if a large project)*	☐ S	☐ P	☐ U	☐ N/A
04	A QA Bid was submitted for the project? [number of QA hours ______]	☐ S	☐ P	☐ U	☐ N/A
05	Candidate risks were identified in the proposal?	☐ S	☐ P	☐ U	☐ N/A
06	Engineering tasks were provided for in the proposal?	☐ S	☐ P	☐ U	☐ N/A
07	Requirements Management tasks were provided for?	☐ S	☐ P	☐ U	☐ N/A
08	Work product reviews (peer reviews) were provided for?	☐ S	☐ P	☐ U	☐ N/A
09	Configuration Management tasks were provided for?	☐ S	☐ P	☐ U	☐ N/A
10	Candidate DAR (heavy) tasks, if any, were provided for?	☐ S	☐ P	☐ U	☐ N/A
11	Was a waiver for process submitted for this project? [enter the waiver # ____]	☐ S	☐ P	☐ U	☐ N/A

OBJECTIVE EVIDENCE ASSESSED / OBSERVATIONS / COMMENTS / N/A EXPLANATION (continue on back, if necessary)

Project Management Related

12	The Project Startup Checklist from the PAL was used?	☐ S	☐ P	☐ U	☐ N/A

13	The checklist is complete or nearing completion on all tasks?	☐ S	☐ P	☐ U	☐ N/A
14	Has a project folder been created on P: with QA having 'read-only' access?	☐ S	☐ P	☐ U	☐ N/A
15	A SwRI Project Record Sheet created and provided to QA?	☐ S	☐ P	☐ U	☐ N/A
16	The staff or personnel for the project team are ready? [# of personnel _____]	☐ S	☐ P	☐ U	☐ N/A
17	For any 'borrowed' people, are formal commitments in place?	☐ S	☐ P	☐ U	☐ N/A
18	For any special training required, have arrangements been made?	☐ S	☐ P	☐ U	☐ N/A
19	Has there been a formal kick-off meeting for the project? [when ___________]	☐ S	☐ P	☐ U	☐ N/A
20	Have there been any other meetings, such as a Team or Management?	☐ S	☐ P	☐ U	☐ N/A
21	Has the default plan been reviewed and a tracking tool usage begun? *(for a medium in size project, with the approved plan due within 30 days)*	☐ S	☐ P	☐ U	☐ N/A
22	Has work started on developing a unique custom plan for the project? *(for a large size project, with the approved plan due within 60 days)*	☐ S	☐ P	☐ U	☐ N/A
23	Has the Metrics folder with selected Metrics been established on the 'P' drive?	☐ S	☐ P	☐ U	☐ N/A
24	Are all of the required metrics to be utilized?	☐ S	☐ P	☐ U	☐ N/A
25	Is the project prepared with adequate resources, etc. to proceed on schedule?	☐ S	☐ P	☐ U	☐ N/A
QA Related					
26	A QA charge code for the project has been established? [specify _____________]	☐ S	☐ P	☐ U	☐ N/A
27	What is the approximate time for the next QA surveillance? [specify ___________]	☐ S	☐ P	☐ U	☐ N/A

OBJECTIVE EVIDENCE ASSESSED / OBSERVATIONS / COMMENTS / N/A EXPLANATION (continue on back, if necessary)

INSTITUTE QUALITY SYSTEMS **PERIODIC SURVEILLANCE CHECKLIST**	
Project Manager:	
Project End Date:	Date(s) Conducted:
General Description of New Project:	
Brief Description of Current Project Activities	

	Project Monitoring And Control				
1	What is the date of the most recently conducted Management Review Meeting ?	☐ S	☐ P	☐ U	☐ N/A
2	Was a copy of the MRM (or spreadsheet) provided to QA (or access provided) ?	☐ S	☐ P	☐ U	☐ N/A
3	Was there a recent Team Review (or status) Meeting with minutes taken ?	☐ S	☐ P	☐ U	☐ N/A
4	If there were changes to (Baseline items, Baseline documents, cost variance for WBS tasks,the project schedule, contract deliverables, organization structure, staffing, training, stakeholder involvement, Project Plan revision considerations), was the MRM updated ?	☐ S	☐ P	☐ U	☐ N/A
5	Were the projects selected metrics recently updated and analyzed ?	☐ S	☐ P	☐ U	☐ N/A
6	Based on the analysis, were any resulting actions noted in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
7	Has the project tracked (team/mgmt meetings, QA surveillances, customer reviews, Lessons Learned meetings, Process Effectiveness Questionnaires entries) in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
8	Has the project tracked (‘What’s New’, Risks, DAR activity, Engineering activities, Work Product or Peer Reviews, Configuration Management activities) in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
9	Has the project reviewed the project’s risk and updated the Risk Mgmt Plan as needed ?	☐ S	☐ P	☐ U	☐ N/A
10	Has the project tracked engineering items (resources, interfaces, requirements and the bidirectional traceability of the requirements ?	☐ S	☐ P	☐ U	☐ N/A
11	Have there been any changes to the Project Management Reviews that have required changes to the Project Plan, and those changes were noted in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
12	Have there been any changes to the Configuration Management in the Project Plan and those changes were noted in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
13	Have there been any changes to the DAR activities as noted in the Project Plan and those changes were noted in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
14	Has the project tracked cost/schedule/resources for the project and made a Corrective Action, with a Corrective Action Plan documented in the MRM ?	☐ S	☐ P	☐ U	☐ N/A

15	Have there been any changes in the resource Commitments documented in the Project Plan, that were then noted in the MRM ?	☐ S	☐ P	☐ U	☐ N/A
16	Have there been any changes (requirements, cost, schedule, commitments) needed for the Project Plan, that were documented in the MRM or tracking spreadsheet ?	☐ S	☐ P	☐ U	☐ N/A
17	If the project is doing EVM, have the Active Phases been opened, relevant data collected, EVM parameters calculated, and the EVM reports generated ?	☐ S	☐ P	☐ U	☐ N/A
	Project Closeout				
18	If the project is complete and ready for closeout, has a Project Closeout Checklist been Initiated , and the archiving of project records and files begun ?	☐ S	☐ P	☐ U	☐ N/A
	Configuration Management				
19	Did the project establish a Project Repository and a PR/CR system as needed ? *(specify if a CM tool is to be used: __________________)*	☐ S	☐ P	☐ U	☐ N/A
20	Did the project create a Baseline (configuration items tagged with a Version Number, etc.) ?	☐ S	☐ P	☐ U	☐ N/A
21	Were new Configuration Items added and tagged as needed (if applicable) ?	☐ S	☐ P	☐ U	☐ N/A
22	Were Problem Reports created, reviewed, worked, and managed (as needed) ?	☐ S	☐ P	☐ U	☐ N/A
23	Were Change Requests created, reviewed, worked, and managed (as needed) ?	☐ S	☐ P	☐ U	☐ N/A
24	Were the PR/CRs addressed, implemented, and tracked to closure (as needed) ?	☐ S	☐ P	☐ U	☐ N/A
25	Did the project completed a periodic Configuration Status Accounting Report ? *(specify the date of the recent CSA report: ________________)*	☐ S	☐ P	☐ U	☐ N/A
26	Did the project do a physical configuration audit and create a VDD (if applicable) ?	☐ S	☐ P	☐ U	☐ N/A
27	Did the project do a functional configuration audit, by completing an ATP exercise with the associated Audit (or ATP) Report (if applicab le) ?	☐ S	☐ P	☐ U	☐ N/A
	Measurement And Analysis, Use Of Metrics				
28	Has the project selected the appropriate metrics to be used based on the PDP View, and created the Metrics Collection records in Organizational Metrics on the P: Drive ?	☐ S	☐ P	☐ U	☐ N/A
29	Has the project entered the metrics data as required at least quarterly ?	☐ S	☐ P	☐ U	☐ N/A
30	Has the project used the selected metrics to quantifiably manage the project, and take action when undesirable trends are identified in the metrics ?	☐ S	☐ P	☐ U	☐ N/A
	Root Cause And Decision Analysis				

31	Has the project watched for DAR candidates and evaluated whether appropriate DAR action needs to be taken ?	☐ S	☐ P	☐ U	☐ N/A
32	If a DAR Heavy candidate was determined, was a DAR Worksheet with detailed DAR planning specifics entered ?	☐ S	☐ P	☐ U	☐ N/A
33	If a DAR Heavy candidate, was a DAR Heavy Worksheet updated with evaluation criteria ?	☐ S	☐ P	☐ U	☐ N/A
34	If a DAR Heavy candidate, was the worksheet updated with Evaluation Methods and alternative solutions, culminating in a selected solution ?	☐ S	☐ P	☐ U	☐ N/A
35	If a DAR Medium or Light, was the appropriate documentation completed ?	☐ S	☐ P	☐ U	☐ N/A
36	If there was the opportunity to perform Root Cause Analysis, was a Root Cause Analysis Report created, along with Project-spe cific Corrective Action ?	☐ S	☐ P	☐ U	☐ N/A
	Quality Assurance				
37	Does the Project Manager prepare for and participate in the quarterly QA Surveillance ?	☐ S	☐ P	☐ U	☐ N/A
38	If the project has an Open NCR or CAR, has the project responded appropriately ?	☐ S	☐ P	☐ U	☐ N/A
38	Has the project established the date/time for the next QA Surveillance ? *(enter the approximate timeframe for the next surveillance: __________________.)*	☐ S	☐ P	☐ U	☐ N/A
	Subcontract Management				
	If the project does <u>not</u> have a subcontract requiring Subcontract Management, skip down to the engineering tasks and Question number 46.	☐ S	☐ P	☐ U	☐ N/A
40	Has the project established criteria for selecting a subcontractor ?	☐ S	☐ P	☐ U	☐ N/A
41	Has the project developed the requirements and criteria for subcontractor products, and thus a subcontractor Statement of Work (SOW) ?	☐ S	☐ P	☐ U	☐ N/A
42	Has the project created a baselined Subcontractor Selection Plan, and started the evaluation process for selecting a subcontractor ?	☐ S	☐ P	☐ U	☐ N/A
43	Depending on the processes to be used (ours/theirs), has an appropriate Subcontractor Project Plan been developed and approved ?	☐ S	☐ P	☐ U	☐ N/A
44	Has the project performed the planned Subcontractor reviews (documented, etc.) ?	☐ S	☐ P	☐ U	☐ N/A
45	Have the required QA Surveillances of the Subcontractor, and Audit Reports, been done ?	☐ S	☐ P	☐ U	☐ N/A
46	Has the ATP been done and the Subcontractor products reviewed and accepted ?	☐ S	☐ P	☐ U	☐ N/A

	Following are the Areas for 'Development' Projects : Engineering Preparation [Including 'Software Life Cycles']				
47	Has the project needed to update the Project Plan to tailor the selected Software Life Cycle, Analysis and Design Methods, or Coding Standards ?	☐ S	☐ P	☐ U	☐ N/A
48	Has the project been able to clearly apply the selected SDL and implement it via the Project Plan and the Work Breakdown Structure (WBS) ?	☐ S	☐ P	☐ U	☐ N/A
	Requirements Development And Management				
49	Has the project successfully elicited the clients requirements and produced documented Stakeholder Needs ? (or did the client provide the requirements ?)	☐ S	☐ P	☐ U	☐ N/A
50	Was a Requirements Specification Document (RSD) produced ?	☐ S	☐ P	☐ U	☐ N/A
51	Were the requirements analyzed and validated, with defects and issues identified ?	☐ S	☐ P	☐ U	☐ N/A
52	Were the requirements baselined such that changes could be controlled ?	☐ S	☐ P	☐ U	☐ N/A
53	Was a Requirements Traceability Matrix (RTM) created and then maintained ?	☐ S	☐ P	☐ U	☐ N/A
	Analysis And Design				
54	Has the project performed analysis and design and developed a high-level Software Design Document (SDD), a high level Database Design Document (DBDD), and a high-level Interface Control Document (ICD) ?	☐ S	☐ P	☐ U	☐ N/A
55	Has the project done more analysis and design and developed a detailed SDD, a detailed DBDD, and a detailed ICD ?	☐ S	☐ P	☐ U	☐ N/A
	Implementation And Unit Test				
56	Has the project performed coding or product construction and produced coded software that compiles 'clean' without significant errors ?	☐ S	☐ P	☐ U	☐ N/A
57	Has the project conducted Unit Testing and documented the tests and the results	☐ S	☐ P	☐ U	☐ N/A
	Integration and Integration Test				
58	Has the project conducted Integration Testing and documented the tests and the results ?	☐ S	☐ P	☐ U	☐ N/A
59	Has the Integration Testing been controlled and Problem Reports/Change Requests (PR/CRs) documented as a result of the testing ?	☐ S	☐ P	☐ U	☐ N/A
	Acceptance Test,Deployment, and Field Support				

60	Has the project performed Acceptance Testing activities and documented the tests and theresults (with a customer/client signature if the last ATP run) ?	☐ S	☐ P	☐ U	☐ N/A
61	If appropriate, has the project performed Alpha or Beta Testing activities and documented the tests and the results (with a customer/client signature if the last run) ?	☐ S	☐ P	☐ U	☐ N/A
62	If appropriate, has the project performed Commissioning (or Burn-In) Testing activities and documented the tests and the results (with a customer/client signature if appropriate) ?	☐ S	☐ P	☐ U	☐ N/A
	Reviews (Of Work Products) –				
63	If the project planned for a Formal Technical Review, have preparations been made and is the formal review scheduled and adequately prepared for ?	☐ S	☐ P	☐ U	☐ N/A
64	If the project conducted a Formal Technical Review, did it document the defects and Issues, produce a Peer Review Report, and update a Defect/Issues repository ?	☐ S	☐ P	☐ U	☐ N/A
65	If the project planned for a Walkthrough Peer Review, have all preparations been made and is the walkthrough review scheduled and adequately prepared for ?	☐ S	☐ P	☐ U	☐ N/A
66	If the project conducted a Walkthrough Review, did it document the defects and issues, produce a Peer Review Report, and update a Defect/Issues repository ?	☐ S	☐ P	☐ U	☐ N/A
67	If the project planned for Buddy Check reviews, did it produce adequate documentation (participants, material reviewed, date, findings, etc.) ?	☐ S	☐ P	☐ U	☐ N/A
68	If the project planned for Management Reviews, did it result in an approved work product or a ‘red-lined’ product to be further worked and refined ?	☐ S	☐ P	☐ U	☐ N/A
69	If the project planned for Customer Reviews, did it produce records of the review, possibly with an updated PR/CR tracking repository ?	☐ S	☐ P	☐ U	☐ N/A
70	If the project planned for or happened to conduct a ‘heading check’, did it produce minutes of the meeting, topics covered, and significant decisions made ?	☐ S	☐ P	☐ U	☐ N/A
	End User Documentation				
71	If the project prepared End User Documentation, did it produce a Software Users Manual (SUM), and a Version Description Document (VDD), and/or Training Materials ?	☐ S	☐ P	☐ U	☐ N/A
72	Were On-Line Help Files prepared as a part of the End User Documentation ?	☐ S	☐ P	☐ U	☐ N/A

	Delivery (to the Customer)				
73	If the project has made a delivery of a required work product, was the PAL Document Distribution Cycle followed so that the delivered product was then archived ?	☐ S	☐ P	☐ U	☐ N/A
74	If the project has made a delivery of a non-document work product (i.e., software), was a formal product (from ATP testing) delivered or installed, and then archived ?	☐ S	☐ P	☐ U	☐ N/A

Attachment B
Coding Standard for C Sharp (C#)

1. Introduction

1.1. Purpose

This document details the standards and practices for the development of software in the C Sharp (C#) language. The standards are designed to accomplish several goals:

- Provide a uniform look and feel to all software developed by a group.
- Make it easier for developers to review each other's code.
- Prevent the introduction of defects through the use of best practice structural and syntactic conventions.
- Make it easier to find and fix defects.
- Make it more efficient for developers to switch between projects when resource reallocations are required.
- Improve software portability and reusability.
- Improve long term maintenance.

The majority of the time and money spent on production software is spent on maintenance. Since maintenance is a future activity, the software written today is a means of communicating ideas and information to some future programmer, perhaps even the original author. Therefore, the primary goal when writing software must be to produce code that is ***correct, easy to read*** and ***easy to maintain***.

1.2. Scope

This standard is applicable to all software developed in C#, regardless of whether the result is for a Windows desktop application, Web Service, Web site, etc.

1.3. Variances

For a specific project, there may be special reasons that require deviation from these standards. In such cases, it is the project manager's responsibility to define the deviations and to inform all project team members. This must be documented in a memo or other written format that can become part of the project records.

2. Naming Conventions

The naming scheme is one of the most influential aids to understanding the logical flow of an application. A name should tell "what" rather than "how." By avoiding names that expose the underlying implementation, which can change, a layer of abstraction that simplifies the complexity is preserved. For example, use "getNextStudent()" instead of "getNextArrayElement()". A tenet of naming is that difficulty in selecting a proper name may indicate a need to further analyze or define the purpose of an item. Make names long enough to be meaningful, but short enough to avoid verbosity. Programmatically, a unique name serves only to differentiate one item from another. Expressive names function as an aid to a human reader; therefore, it makes sense to provide a name that a human reader can comprehend. However, be certain that the chosen names are in compliance with the C# language's rules and standards.

2.1 Method Names

The common code based used by projects utilizes camel casing to allow a programmer to easily distinguish between code from Microsoft libraries and internal code. Individual processes may either use camel casing ("getNextStudent") or the Microsoft standard of capitalizing each letter of words ("GetNextStudent"). Naming must be consistent within a process. When maintaining code, continue the naming style already in use.

Do:

- Use the verb-noun method for naming routines that perform some operation on a given object, such as "calculateInvoiceTotal()".
- Ensure method overloads perform similar operations.
- When naming methods, include a description of the value being returned, such as "getCurrentWindowName()".
- When naming methods whose operation includes specific units of measure, include the type of units, such as "computeDistanceInMeters()".
- Make names descriptive without excessive length.

Don't:

- Use elusive names that are open to subjective interpretation, such as "AnalyzeThis()". Such names contribute to ambiguity more than abstraction.
- Include class names in the name of class properties, such as "Book.BookTitle". Instead, use "Book.Title".

2.2 Variable Names

Do:

- Append computation qualifiers (Avg, Sum, Min, Max, Index) to the end of a variable name where appropriate.
- Coding Standard for C# 3
- Use complimentary pairs in variable names, such as min/max, begin/end, and open/close.
- Use camel casing ("documentFormatType") where the first letter of each word except the first is capitalized.

- Use boolean variable names containing Is which implies Yes/No or True/False values, such as “fileIsFound”.
- Use a meaningful name even for short-lived variables.
- Include units of measure for variables with a specific unit of measure, such as “distanceInMeters”.

Don’t:

- Use mysterious names that are open to subjective interpretation, such as “xxK8.”
- Use terms such as Flag when naming status variables, which differ from Boolean variables in that they may have more than two possible values. Instead of “documentFlag”, use a more descriptive name such as “documentFormatType”.
- Use single-letter variables names, other than for short (less than three lines) loop indexing.
- Use literal numbers or literal strings, such as for (i = 1; i < 7; i++), where 7 means nothing to someone reading the code. Instead, use named constants, such as for (i = 1; i <
- NUM_DAYS_IN_WEEK; i++) for ease of maintenance and understanding.
  - There are exceptions, the values 0, 1 and null can nearly always be used safely.
  - Often 2 and -1 can also be used.
  - Strings intended for logging or tracing are exempt from this rule.

2.3 Graphical User Interface (GUI) Widgets

GUI controls should be named to reflect their type. This makes the code easier to understand and promotes future maintainability. The following standard is recommended for naming controls. Controls not expressly cited in this table should be named in a similar fashion.

Control Prefix	**Example**
TextBox txt:	txtName, txtPassword
Label lbl:	lblErrorMsg
ListBox list:	listPrescriptions
ComboBox combo:	comboNames
ListView lvw:	lvwRxData
TreeView tree:	treePatients
Button btn:	btnPrint

Controls that are static (i.e., never referenced in the code), such as a GroupBox label that never changes, need not follow this convention.

2.4 Class Names

Class names should be Pascal-cased and should be descriptive of the class’ intended purpose.

- CacheManager
- XmlSerializer

When using Visual Studio, always change the default name assigned by the Integrated Development Environment (IDE). For example, change ***Form1*** to ***ConfigurationForm*** and ***Service1*** to ***SvcEmr***.

In many cases, the .NET development environment allows the declaration of instances of remote classes in the same manner as a local instance of a “normal” class. This is a powerful feature;

however, it should be made apparent that the instance is "not normal." Additionally, an "interface" is a special type of object. Use the following prefixes for special types of classes:

"I" for interfaces **IAsyncResult**
"Svc" for web service classes **SvcEmr, SvcPharmacy**
"RemObj" for remotable objects **RemObjWaveManager**

Miscellaneous class usage information:

- Base classes are a useful way to group objects that share a common set of functionality. Base classes can provide a default set of functionality, while allowing customization through extension.
- Minimize the use of abbreviations, but use those that are created consistently. An abbreviation should have only one meaning. For example, if "min" is used to abbreviate minimum, do so everywhere and do not use "min" to also abbreviate minute.
- File and folder names, like procedure names, should accurately describe their purpose.
- Avoid reusing names for different elements, such as a routine called "ProcessSales()" and a variable called "iProcessSales".
- Avoid homonyms, such as write and right, when naming elements to prevent confusion during code reviews.
- When naming elements, avoid commonly misspelled words. Also, be aware of differences that exist between regional spellings, such as color/colour and check/cheque.
- Avoid typographical marks to identify data types, such as "$" for strings or "%" for integers.

3. Code Format

3.1 Namespaces

Namespaces should be in lower case and according to the following pattern:

gov.its.<process>.<folder>.<subfolder>

For example, for LCS, the base classes are contained in the “gov.its.lcs” namespace. Folders can be within that namespace such as “gov.its.lcs.handlers”. For classes in these types of namespaces, the project should contain corresponding folders.

3.2 Methods

Since code tends to be viewed one screen at a time, a single method should fit on one screen, if possible. Very rarely should a method size exceed two printed pages (including comments). Every C# method shall have a standard method header. This leverages the eXtensible Mark-up Language (XML) Documentation feature in Visual Studio for creating code comment reports as in this example:

```
/// <summary>
/// Clean up any resources being used. Stops and releases timer resources
/// </summary>
/// <param name=”disposing”></param>
/// <returns>True if no error occured or False if an error occured.</returns>
protected override bool Dispose( bool disposing )
```

When calling a method, do not put each of the arguments on a separate line. Where possible, the procedure call should occupy a single line. For procedure calls with a lot of arguments, it is acceptable to split the line.

3.3 Class Format

- Classes must have class documentation and a copyright header at the top of the file.
- Sample copyright headers should be provided by the project manager.
- All fields for a class should be at the top of the class.
- Use regions to encapsulate variables, constructors and methods.
- Typical use of regions would be:

```
public class MyClass
{
#region private member variables
#endregion private member variables
#region protected member variables
#endregion
#region constructors
public MyClass()
Coding Standard for C# 6
{
}
#endregion constructors
#region public methods
#endregion
#region protected methods
#endregion
#region public methods
#endregion
}
```

- For methods, regions may be used differently in order to group by functionality. For instance:

```
#region travel time calculations
#endregion travel time calculations
#region aggregators of data
#endregion aggregators of data
#region configuration of links
#endregion configuration of links
```

- Adding the region name to the "#endregion" is useful when scanning through a code file.

3.4 Indentation and Braces

Indentation, spacing, and other formatting rules are enforced using **StyleCop** (http://code.msdn.microsoft.com/sourceanalysis). Settings can vary on a per project basis. The project should provide a sample *Settings.SourceAnalysis* file containing the appropriate settings which can be ignored. The settings file only provides the exceptions to rules set in **StyleCop**.

- The standard indent level is four spaces. Make sure editors insert ***spaces*** when indenting, not tab characters.
- For switch statements, the CASE entries shall be at the same level as the switch.
- As a general rule, methods should contain no more than four levels of indentation.
- Line up braces on the left. Always use braces, even for single lines of code following if, else, for, or while constructs. This saves confusion and mistakes, and it does not cause the compiler to generate any extra code.

  WRONG:
  ```
  while( ... ) {
  ...
  }
  ```
  RIGHT:
  ```
  while( ... )
  {
  ...
  }
  ```
- Even though C# allows a looping or conditional construct within a single body line to forego the brackets {}, brackets should always be used. This reduces the potential for errors if another line of code is added later and makes the code more readable.

  WRONG:
  ```
  while ( true )
  DoWork();
  ```
  RIGHT:
  ```
  while ( true )
  {
  DoWork()
  }
  ```

3.5 Line Length

Comment and code lines should not extend too long. Previously, this was 80 characters, but with C# more verbose and screens having higher resolutions, this can vary. Now, this is more subjective.

3.6 Comments

- Use double slashes for comments. This precludes having to use an ending *\.

```
// This is a comment block
```

- Use // TODO to mark places where code still needs to be implemented, tested or modified.
- Indent comments to the same level as the code to which it applies.
- Under most conditions, do NOT put comments at the end of a code line. The comment should immediately precede the lines to which it applies.
- Do not use stars at end of comments (box format).
- Use XML tags for documenting types and members. Example:

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

The following Commenting Checklist was taken from *Code Complete: A Practical Handbook of Software Construction, 1st Edition* by Steven C. McConnell (1993).

- Does the source listing contain most of the information about the program?
- Can someone pick up the code and immediately begin to understand it?
- Do the comments explain the codes intent or summarize what the code does, rather than just repeating the code?
- Has tricky code been re-written rather than commented?
- Are the comments up to date?
- Are comments clear and correct?
- Does the commenting style allow comments to be easily modified?
- Do the comments focus on *why* rather than *how*?
- Do the comments prepare the reader for the code to follow?
- Does every comment count? Have redundant, extraneous, and self-indulgent comments been removed or improved?
- Are surprises documented?
- Have abbreviations been avoided?
- Is the distinction between major and minor comments clear?
- Is code that works around an error or undocumented feature commented?
- Are units on data declarations commented?
- Are the ranges of values on numeric data commented?
- Are coded data meanings commented?
- Are limitations on input data commented?
- Are flags documented to the bit level?

3.7 Spacing

Use spaces for clarity. If necessary, sacrifice space for readability. Again, these settings can be set per project using **StyleCop**.

OK:

```
for( i = 1; i < 5; i++ );
```

NOT:

```
for(i=1;i<5;i++);
```

3.8 Miscellaneous

- Only use “this.” to prevent name clashing (if the method parameter matches a class field).
- Avoid multiple or conditional return statements.
- If implementing one of the Object methods (e.g., “Equals”, “GetHashCode”), it is required that both must be implemented. Also override when implementing the “IComparable” interface.
- Avoid the use of exceptions as flow control. Exceptions should be thrown in exceptional circumstances.
- Always log that an exception is thrown.
- Code reviews are completed using the checklist in Appendix C. When developing code, the checklist should be gone over to ensure compliance.

4. **Coding** Practices

4.1 Clarity

Write code for clarity and understanding. Leave the optimization for the compiler. Do not try to guess where performance “bottlenecks” will occur. If it is later decided that the code runs too slowly, performance tools can help identify the true “bottlenecks.” Use parenthesis to make code clearer (but do not get carried away). If the order of operation is not intuitively obvious, use parenthesis.

4.2 Object Scope

In a class definition, all member variables shall be private or protected. Access to member variables by anyone other than a derived class shall be through either member methods or the get set accessors.

```
public UddiKeyCollection Keys
{
get
{
return this.keys;
}
set
{
this.keys = value;
}
}
```

4.3 Constants

A constant is an expression that can be fully evaluated at compile time. Constants should be all upper case with underbars:

```
public const int MAX_QUEUE_ITEMS = 100;
```

The same holds true for constant enumerations.

4.4 Flags

Flags are only to be used to mark events or options. Not counting the flag initialization, there should only be one place the flag is set, and one place the flag is cleared. It can be checked any number of places. If multiple flags are required to control the logical flow of a process, and these flags are set or cleared in more than one place, then what is really present is state processing. Define a state variable, constants for the various states, procedures for entering (and/or exiting if necessary) the defined states, and the program logic for each event in each state.

4.5 Data Types

The system namespace is the root namespace for fundamental types in the .NET Framework. This namespace includes classes that represent the base data types used by all applications: “Object” (the root of the inheritance hierarchy), “Byte”, “Char”, “Array”, “Int32”, “String”, and so on. Many of these types correspond to the primitive data types that C# uses. When writing code using .NET Framework types, use the C# corresponding keyword when a .NET Framework base data type is expected. The following table lists some of the value types the .NET Framework supplies, briefly describes each type, and indicates the corresponding type in C#. The table also includes entries for the Object and String classes, for which C# has corresponding keywords.

Category	**Class Name**	**Description**	**C# Data Type**
Integer	Byte	An 8-bit unsigned integer.	byte
	SByte	An 8-bit signed integer. Not CLS compliant.	sbyte
	Int16	A 16-bit signed integer.	short
	Int32	A 32-bit signed integer.	int
	Int64	A 64-bit signed integer.	long
	UInt16	A 16-bit unsigned integer. Not CLS compliant.	Ushort
	UInt32	A 32-bit unsigned integer. Not CLS compliant.	Uint
	UInt64	A 64-bit unsigned integer. Not CLS compliant.	ulong
Floating point	Single	A single-precision (32-bit) floatingpoint number.	float
	Double	A double-precision (64-bit) floating-point number.	double
Logical	Boolean	A Boolean value (true or false).	bool
Other	Char	A Unicode (16-bit) character.	char
	Decimal	A 96-bit decimal value.	decimal
	IntPtr	A signed integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform).	IntPtr No built-in type.
	UIntPtr	An unsigned integer whose size depends on the underlying platform (a 32- bit value on a 32-bit platform and a 64-bit value on a 64-bit platform). Not CLS compliant.	UIntPtr No built-in type.
Class objects	Object	The root of the object hierarchy.	object
	String	An immutable, fixed-length string of Unicode characters.	string

4.6 GUIs

All C# user interfaces will comply with standards specified in the book *Microsoft Windows User Experience (Microsoft Professional Edition), 1st Edition* by Microsoft Corporation (1999) in the absence of another standard. When other GUI standards are being applied (i.e., “VA” GUI standard), the user interface must still comply with the published Microsoft standards. Use the checklists provided in Appendices A and B for designing and implementing Windows Forms based GUIs.

4.7 Generated Code

Code that is automatically generated from the Visual Studio development environment, or other tools such as the XML Schema Definition Language (XSD) utility, does not have to comply with this coding standard. Do not attempt to change generated code, in most cases modifying the code break it. StyleCop can be made to ignore these files by adding a tag at the top of the file.

4.8 XML

Since XML is tightly integrated into C# and the .NET framework, it is briefly addressed here.

- When designing XML schema, XSD should be used. Document Type Definitions (DTD) will not be used.
- XML tags should be all lower case where possible. This convention does not apply when generating schema using automated tools. For instance, generating a DataSet schema from an existing database table.
- XML documents should be indented for readability. Where possible, build a hierarchical structure of data types by including multiple schema of simpler types.
- When creating XML for transmission, create in an appropriately indented format. This will allow the text to be human readable without needing to be reformatted.

Appendix A
Windows Form Design Checklist

Organization

Placement of controls provide an ordering that:

- ☐ Puts most important and frequently used controls placed in the upper left?
- ☐ Is organized to flow in a top-to-bottom, left-to-right sequence? Similar to reading a page in a book.
- ☐ Is logical and sequential?
- ☐ Minimizes pointer and eye movement distances?
- ☐ Minimizes the number of times a person’s hand has to travel between the keyboard and mouse.

Visual Clutter

- ☐ Maintains low screen density levels. Controls and labels do not occupy greater than 30 to 40 percent of the form.

Aesthetics

Provide a visually pleasing composition through:

- ☐ Adequate use of white space.
- ☐ Balance.
- ☐ Groupings.
- ☐ Alignment of elements.

Appendix B
Windows Form Implementation Checklist

Control Alignment

- ☐ Controls and their labels evenly spaced?
- ☐ Control labels centered vertically on the associated control? Use the Layout-Align-Vert. Center menu pick.
- ☐ Controls and buttons should not touch each other or the window border.
- ☐ Leave at least two blank spaces between the left and right borders and the widest element within the form.

Tab Order

- ☐ Tab order follows a logical sequence?
- ☐ Static labels immediately precede their associated control in the tab order?

Groups

Groups are normally associated with Radio Buttons and Check Boxes.

- ☐ Choice descriptions left aligned?
- ☐ Selection indicators (buttons/checkboxes) left aligned?
- ☐ Captions inscribed in borders left aligned?
- ☐ Tab order and Group style properly assigned?
- ☐ The tab order is important in group assignments. The “Group” style should be set on the first item of a group and the first item immediately following a group in the tab order.

Buttons

- ☐ Command buttons that affect the entire window displayed horizontally and centered at the window bottom? Use the Layout-Arrange Buttons menu pick.
- ☐ Buttons that raise another dialog box should have a descriptive name followed by an ellipsis (…) to indicate that further interaction is required.

Keyboard Accelerators (Hot Keys)

Windows Forms items may be selected by pressing the Alt key and the underlined letter of a control’s label. These are known as accelerators. They are assigned to the label of a control in the Caption property by placing an ampersand (&) symbol before the letter to be used as an accelerator.

- ☐ Are all accelerators unique on the dialog box?
- ☐ The default key (normally the OK button) should not have an accelerator key assigned.
- ☐ Each accelerator key tested?

APPENDIX C
Code Review Checklist

General Code Smoke Test

- ☐ **Does the code build correctly?**
  No errors should occur when building the source code. No warnings should be introduced by changes made to the code.
- ☐ **Does the code execute as expected?**
  When executed, the code does what it is supposed to.
- ☐ **Do you understand the code you are reviewing?**
  As a reviewer, you should understand the code. If you don't, the review may not be complete, or the code may not be well commented.
- ☐ **Has the developer tested the code?**
  Insure the developer has unit tested the code before sending it for review. All the limit cases should have been tested.

Comments and Coding Conventions

- ☐ **Does the code respect the project coding conventions?**
  Check that the coding conventions have been followed. Variable naming, indentation, and bracket style should be used.
- ☐ **Does the source file start with an appropriate header and copyright information?**
  Each source file should start with an appropriate header and copyright information. All source files should have a comment block describing the functionality provided by the file.
- ☐ **Are variable declarations properly commented?**
  Comments are required for aspects of variables that the name doesn't describe. Each global variable should indicate its purpose and why it needs to be global.
- ☐ **Are units of numeric data clearly stated?**
  Comment the units of numeric data. For example, if a number represents length, indicate if it is in feet or meters.
- ☐ **Are all functions, methods and classes documented?**
  Describe each routine, method, and class in one or two sentences at the top of its definition. If you can't describe it in a short sentence or two, you may need to reassess its purpose. It might be a sign that the design needs to be improved.
- ☐ **Are function parameters used for input or output clearly identified as such?**
  Make it clear which parameters are used for input and output.
- ☐ **Are complex algorithms and code optimizations adequately commented?**
  Complex areas, algorithms, and code optimizations should be sufficiently commented, so other developers can understand the code and walk through it.
- ☐ **Does code that has been commented out have an explanation?**
  There should be an explanation for any code that is commented out. "Dead Code" should be removed. If it is a temporary hack, it should be identified as such.
- ☐ **Are comments used to identify missing functionality or unresolved issues in the code?**

A comment is required for all code not completely implemented. The comment should describe what's left to do or is missing. You should also use a distinctive marker that you can search for later (For example: "TODO:francis").

Error Handling

- ☐ **Are assertions used everywhere data is expected to have a valid value or range?**
  Assertions make it easier to identify potential problems. For example, test if pointers or references are valid. NOTE: This is not typical for our code. What are the pros/cons of assertion usage for range checking?
- ☐ **Are errors properly handled each time a function returns?**
  An error should be detected and handled if it affects the execution of the rest of a routine. For example, if a resource allocation fails, this affects the rest of the routine if it uses that resource. This should be detected and proper action taken. In some cases, the "proper action" may simply be to log the error.
- ☐ **Are resources and memory released in all error paths?**
  Make sure all resources and memory allocated are released in the error paths.
- ☐ **Are all thrown exceptions handled properly?**
  If the source code uses a routine that throws an exception, there should be a function in the call stack that catches it and handles it properly.
- ☐ **Is the function caller notified when an error is detected?**
  Consider notifying your caller when an error is detected. If the error might affect your caller, the caller should be notified. For example, the "Open" methods of a file class should return error conditions. Even if the class stays in a valid state and other calls to the class will be handled properly, the caller might be interested in doing some error handling of its own.
- ☐ **Has error handling code been tested?**
  Don't forget that error handling code that can be defective. It is important to write test cases that exercise it.

Resource Leaks

- ☐ **Is allocated memory (non-garbage collected) freed?**
  All allocated memory needs to be freed when no longer needed. Make sure memory is released in all code paths, especially in error code paths.
- ☐ **Are all objects (Database connections, Sockets, Files, etc.) freed even when an error occurs?**
  File, Sockets, Database connections, etc. (basically all objects where a creation and a deletion method exist) should be freed even when an error occurs. For example, whenever you use "new" in C++, there should be a delete somewhere that disposes of the object. Resources that are opened must be closed. For example, when opening a file in most development environments, you need to call a method to close the file when you're done.
- ☐ **Is the same object released more than once?**
  Make sure there's no code path where the same object is released more than once. Check error code paths.
- ☐ **Does the code accurately keep track of reference counting?**

Frequently a reference counter is used to keep the reference count on objects (For example, COM objects). The object uses the reference counter to determine when to destroy itself. In most cases, the developer uses methods to increment or decrement the reference count. Make sure the reference count reflects the number of times an object is referred.

Thread Safeness

- ☐ **Are all global variables thread-safe?**
  If global variables can be accessed by more than one thread, code altering the global variable should be enclosed using a synchronization mechanism such as a mutex. Code accessing the variable should be enclosed with the same mechanism.
- ☐ **Are objects accessed by multiple threads thread-safe?**
  If some objects can be accessed by more than one thread, make sure member variables are protected by synchronization mechanisms.
- ☐ **Are locks released in the same order they are obtained?**
  It is important to release the locks in the same order they were acquired to avoid deadlock situations. Check error code paths.
- ☐ **Is there any possible deadlock or lock contention?**
  Make sure there's no possibility for acquiring a set of locks (mutex, semaphores, etc.) in different orders. For example, if Thread A acquires Lock #1 and then Lock #2, then Thread B shouldn't acquire Lock #2 and then Lock #1.

Control Structures

- ☐ **Are loop ending conditions accurate?**
  Check all loops to make sure they iterate the right number of times. Check the condition that ends the loop; insure it will end out doing the expected number of iterations.
- ☐ **Is the code free of unintended infinite loops?**
  Check for code paths that can cause infinite loops. Make sure end loop conditions will be met unless otherwise documented.

Performance

- ☐ **Do recursive functions run within a reasonable amount of stack space?**
  Recursive functions should run with a reasonable amount of stack space. Generally, it is better to code iterative functions.
- ☐ **Are whole objects duplicated when only references are needed?**
  This happens when objects are passed by value when only references are required. This also applies to algorithms that copy a lot of memory. Consider using an algorithm that minimizes the number of object duplications, reducing the data that needs to be transferred in memory.
- ☐ **Does the code have an impact on size, speed, or memory use?**
  Can it be optimized? For instance, if you use data structures with a large number of occurrences, you might want to reduce the size of the structure.
- ☐ **Are you using blocking system calls when performance is involved?**
  Consider using a different thread for code making a function call that blocks.

- ☐ **Is the code doing busy waits instead of using synchronization mechanisms or timer events?**
  Doing busy waits takes up CPU time. It is a better practice to use synchronization mechanisms.
- ☐ **Was this optimization really needed?**
  Optimizations often make code harder to read and more likely to contain bugs. Such optimizations should be avoided unless a need has been identified. Has the code been profiled?

Functions

- ☐ **Are function parameters explicitly verified in the code?**
  This check is encouraged for functions where you don't control the whole range of values that are sent to the function. This isn't the case for helper functions, for instance. Each function should check its parameter for minimum and maximum possible values. Each pointer or reference should be checked to see if it is null. An error or an exception should occur if a parameter is invalid.
- ☐ **Are arrays explicitly checked for out-of-bound indexes?**
  Make sure an error message is displayed if an index is out-of-bound.
- ☐ **Are functions returning references to objects declared on the stack?**
  Don't return references to objects declared on the stack, return references to objects created on the heap.
- ☐ **Are variables initialized before they are used?**
  Make sure there are no code paths where variables are used prior to being initialized. If an object is used by more than one thread, make sure the object is not in use by another thread when you destroy it. If an object is created by doing a function call, make sure the object was created before using it.
- ☐ **Does the code re-write functionality that could be achieved by using an existing API?**
  Don't reinvent the wheel. New code should use existing functionality as much as possible. Don't rewrite source code that already exists in the project. Code that is replicated in more than one function should be put in a helper function for easier maintenance.

Bug Fixes

- ☐ **Does a fix made to a function change the behavior of caller functions?**
  Sometimes code expects a function to behave incorrectly. Fixing the function can, in some cases, break the caller. If this happens, either fix the code that depends on the function, or add a comment explaining why the code can't be changed.
- ☐ **Does the bug fix correct all the occurrences of the bug?**
  If the code you're reviewing is fixing a bug, make sure it fixes all the occurrences of the bug.

Math

- ☐ **Is the code doing signed/unsigned conversions?**
  Check all signed to unsigned conversions: Can sign completion cause problems? Check all unsigned to signed conversions: Can overflow occur? Test with Minimum and Maximum possible values.