System Integration Test Architecture Specification

(aka: {internal name})

Author: Vlad Rakocevic
Email: Vlad.Rakocevic@radisys.com
Owner:

002-nnnnn-0000
Revision 0.2
September 7, 2018

Approvers:

Reviewers:

Note: To verify signoff, see the on-line repository.

Name	Title/Department
Lakshmi Raman	Senior Director of Systems Engineering/Carrier Grade Platforms
Roy Kravitz	System Integration Test Manager/Carrier Grade Platforms
Lakshmi Raman	Senior Director of Systems Engineering/Carrier Grade Platforms
Robert Eng	Principal Engineer
Roy Kravitz	System Integration Test Manager/Carrier Grade Platforms
Brian Preuss	Systems Engineer/Carrier Grade Platforms
Michael Hansen	SW Engineer/ Carrier Grade Platforms
Rick Springer	HW Engineer/Carrier Grade Platforms
Jody Wagner	New Product Introduction Manager/New Product Introduction Management
Dave Lyons	SW Engineer/ Carrier Grade Platforms

Document control

This is a copy of a RadiSys controlled document. This document is valid only if it is a copy of the latest version available in the on-line repository or from the document owner. To validate the latest version of this document, refer to the on-line repository or contact the document owner.

On-line repository:
Source filename:

Agile
SITArchitecture_v1.0

Preface

> **Note**: The Preface provides instructions and information to help the reader use the document. For example, if you use special font characteristics to indicate specific things, tell the reader about it here. This section should not contain actual document subject matter.

Revision History

All revisions that start with zero (i.e., 0.5, 0.65, etc.) are preliminary drafts for internal reviews.

Table 1. Revision history

No.	Date	Author	Description
0.1	06/21/04	Vlad Rakocevic	Initial draft.
0.2	06/29/04	Vlad Rakocevic	Updates after 1st review and some additions
1.0	07/08/04	Vlad Rakocevic	Updates after 2nd review

Contents

Figures

Tables

1. Overview

This document describes the overall architecture of the System Integration Testing (SIT) process used by RadiSys Corporation for system-level verification and validation. The objective of the SIT architecture is to be generic enough to fit SIT needs for system verification of all ATCA and forthcoming Blade Server Systems. SIT process ensures that a system meets its requirements as depicted in corresponding PRD and functional specifications.

The terminology used in this document is based upon industry standards and customer expectations as much as possible.

The SIT architecture and/or terminology may be changed periodically to reflect prevalent industry standards and conventions. When such changes are effected, a new document revision will be published.

The reader is urged to review the terminology section in this document prior to reading the other sections.

1.1 Terminology

Blade — A module that plugs into a slot in a shelf. This can be for computing, switching, or user I/O processing. "Card" or "Hardware Module" is an equivalent term.

Chassis — Equivalent to "Shelf". RadiSys preferred term is "Shelf".

Chassis Management — Equivalent to "Shelf Management". RadiSys preferred term is "Shelf Management".

Compute Blade — A hardware module that plugs into a slot in a shelf and contains CPUs. Its primary purpose is processing. It may or may not also have physical I/O interfaces.

Control Plane — Functionality associated with real-time control of the user data plane, i.e., the in-band or out-of-band control processes and associated communication protocols that, in conjunction with management plane interactions, determine the structure of the user information data flow. This includes, for example, dynamic signaling and routing protocols such as ISUP or BGP, but not user-managed static routing tables.

Customer Applications — In the RadiSys ATCA modular platform framework, this is any software provided by RadiSys' customers: the system equipment manufacturers. It generally runs atop the RadiSys "middleware" provided as part of an integrated hardware/software platform, although there may be peer elements that are classified as "customer applications" simply because they are provided by the customer and not RadiSys.

E-Keying — Electronic keying. Protocol used to describe the compatibility between the Base Interface, Fabric Interface, Update Channel Interface, and Synchronization Clocks connections of Front Boards.

Element Management — Management of one or more network elements from the same vendor. Typically, an element management system runs in a network operations center and manages all devices from a particular vendor, providing standard northbound interfaces for integration with network management systems and OSSs which manage across diverse elements.

FRU — Field Replaceable Unit. An ATCA entity that can be replaced by an operator in the field, including modules, fan assemblies, etc. FRUs may or may not be hot-swappable, and are managed via the IPMI protocol. "Intelligent FRUs"

	physically include an IPM controller and directly participate in the IPMI protocol. "Managed FRUs" are either Intelligent FRUs or else are represented by an Intelligent FRU (i.e., acts as a proxy). Thus, all FRUs are Managed FRUs, but not all FRUs are Intelligent FRUs. The ATCA specification specifies most of the common FRU behavior.
I/O Blade	Input/Output Blade. A hardware module that plugs into a slot in a shelf and contains physical interfaces and associated protocol processing, possibly including an interworking function. An I/O blade's primary purpose is I/O, as opposed to a compute or switch blade that may also contain physical interfaces, but whose primary purpose is processing or switching. "I/O Blades" are generally referenced in computing platforms (e.g., blade servers) as opposed to network elements, and hence typically carry management or control plane traffic rather than user plane traffic.
Line Card	Similar to "I/O Blade", but found in network elements. A hardware module that performs user plane data path functions, including service protocol processing, interworking functions, and low level data path management functions. The physical layer interface may be collocated on the module or implemented on a mating module (e.g., in a midplane architectures)
Management Plane	Functionality associated with managing the system: the human interfaces and associated logical constructs which directly support management of faults, configuration, accounting, performance, and security.
Middleware	Software that sits above the hardware and OS layers but below "Customer Applications". Standard software provided by RadiSys in conjunction with an integrated hardware platform is always considered part of "Middleware" or lower layer software.
Module	Applies to either a module in a slot in a shelf (blade) or a mezzanine module (e.g., PMC). The more specific terms (blade, mezzanine module) are preferred for clarity.
Network Element (NE)	A single instance of a device in a network that performs an identified network function. Examples of a network element are: SGSN, Media Gateway, Router, Multi-service Switch, etc.
NE Management	Management of a single instance of a device in a network. NE management typically consists of an embedded portion, running on the device itself, such as an SNMP agent or a CLI, and an external application which communicates with the embedded software, such as an SNMP manager or GUI.
Network Management	Management of a network consisting of multiple network elements, potentially from different vendors. This is typically some sort of software system running in a NOC and communicating with element managers and/or directly with NEs.
NOC	Network Operations Center. A center that contains the management systems and/or OSSs (Operations Support Systems) used to control and monitor a service provider's network.
Node Blade	An I/O blade or compute blade (i.e., non-controller and non-switch fabric blade) in an ATCA system.
Platform	The baseline infrastructure functional entity for a shelf or system. It covers the common functions that are not specifically part of the user data plane, control plane, or management plane. This includes the common shelf hardware (e.g., fans, power supply, backplane), shelf management functions, high availability framework, etc. One view of an ATCA telecom/datacom system partitions all functionality into platform, user data plane, control plane, or management plane.

Shelf	A single enclosure containing a backplane or midplane and multiple slots supporting various hardware modules. Typically rack mountable. Also typically contains a fan assembly and possibly power transforming circuitry.
Shelf Control	Generally equivalent to "Shelf Management". RadiSys preferred term is "Shelf Management"
Shelf Management	Low-level Management and control of the common elements of a shelf, including slot management (Module ID, hot-insertion, etc.), health monitoring, diagnostics, fan management, LED control, timing distribution, etc. Essentially covers "hardware management" functions.
Switch Blade	A hardware module that plugs into a slot in a shelf containing one or more switch fabrics and whose primary purpose is switching of user, control, and/or management traffic. It may or may not also have physical I/O interfaces.
System	A somewhat general term with several meanings: 1) a single shelf with integrated HW and SW components; 2) relating to the lower level ("baseline") functionality of a shelf with integrated HW and SW – i.e., "system software"; 3) an aggregation of one or more shelves, each with integrated HW and SW components – this may be more clearly described as a "multi-shelf system".
System Control	Generally equivalent to "System Management". RadiSys preferred term is "System Management".
System Management	Management and control functionality beyond the scope of shelf management. This term implies a broader scope than shelf management in two dimensions: broader functionality and control/management of single or multi-shelf systems. The broader functional scope typically includes all platform functionality as well as the infrastructure for network element management.
System Software	Software responsible for system and shelf management.
User Data Plane	Functionality associated with real-time user data. That is, functionality required to transport user data from an ingress interface to an egress interface, including, for example, protocol termination, encryption, data classification, metering, interworking, forwarding/switching, and traffic management.

1.2 Acronyms/Definitions

ATCA	Advanced Telecommunications Computing Architecture
CDM	Chassis Data Module. Shelf FRU module for Chugach platform.
CPM	Compute Module
CMM	Chassis Management Module (HW blade for shelf management).
Chugach	Intel's platform for ATCA: Dutch Harbor Chassis + CMM + Kennicott.
DCPEM	DC Power Entry Module
Dutch Harbor	Intel's 14U chassis for ATCA. Officially MPCHC0001.
DSM	Disk Storage Module
IDM	IP Distributor Module
IPMB	Intelligent Platform Management Bus

IMPB-0	Intelligent Platform Management Bus Channel 0 as defined in the IPMI specification. This is the logical aggregation of IPMB-A and IPMB-B. The use of IPMB Channels 1 through 7 are not defined in this specification.
IPMB-A	Intelligent Platform Management Buses A and B, respectively. Refers to the two redundant IPMBs that aggregate into IPMB-0.
IPMB-B	See IPMB-A.
IPMI	Intelligent Platform Management Interface
IPMC	Intelligent Platform Management Controller
ISM	Intershelf Switch Module
Kennicott	Intel's dual Xeon CPU blade for ATCA. Officially MPCBL0001.
NAS	Network Attached Storage
PEM	Power Entry Module
QSS1	QuickSilver Shelf #1, based on Rital/Kaparel design.
QuickSilver	RadiSys ATCA program in general, specifically the non-pilot portion developed by RadiSys.
RTM	Rear Transition Module
SCM	Switching and Control Module
SDP	Shelf Display Panel
SDR	Sensor Data Record
SEL	System Event Log
ShMC	Shelf Management Controller
SIT	System Integration Testing
SPM	Shelf Peripheral Manager

1.3 Purpose of the document

The purpose of this document is:

1 To define the scope and objectives of the System Integration Testing

2 To describe the architecture and components of the SIT

3 To describe script modules to ensure re-usability and faster script development

4 To provide guidelines how to define and write reusable System Test Procedures

5 To recommend the format for recording and presenting the test results

6 To describe the framework, organization and use of the SIT scripts in test automation

7 To recommend a coding standard

1.4 Document's Goals

There are several goals that this document wants to achieve:

1 To provide a uniform and common base for design, development and writing of SIT test procedures

2 To provide the architecture of the automated scripts and scripting modules used during SIT

3 To make future SIT script development more efficient by satisfying goals outlined in points 1 and 2

2. Scope and Objectives of the System Integration Testing

System Integration Testing is a validation process of a system under test. SIT is ensuring that a system meets requirements depicted in the PRD and System Functional specifications.

The main objectives of the System Integration Testing are:

1 Functional verification of integrated system components and their interaction

2 Functional verification of integrated system components under operational stress and fault conditions

3 Performance verification of a system under test and its components

4 Robustness verification of a system under test and its components

5 High Availability verification of a system under test

6 Regulatory compliance verification (e.g., NEBS, EMC compliance, etc.) of a system under test

7 Mechanical verification of a system under stress conditions

8 Functional and thermal verification under environmental stress conditions of a system under test

> **Note**: Some of the SIT objectives may not be applicable to some systems under test due to lack of systems functionalities.

Each of these areas is targeted at the system level, that is, testing of interactions between the system components. System level testing may address and use some tests done at the component level, but with different test setup.

The SIT does not include:

1 Functional verification tests performed on a feature-by-feature basis, and verification plans for each system component detail the testing at that level. System Integration Testing focuses on ensuring that the features work together in a system.

2 Conformance and compliance testing performed on a component basis. For example, ATCA protocol Conformance Verification of an ATCA blade is rather performed on the blade verification level than on the system level. However, the ATCA System Power Negotiation feature would be tested during the SIT phase, focusing primarily on power stress and fault conditions in the ATCA chassis (for example, the system under test would include fully populated ATCA chassis with ShMC. The blades in the chassis would request different power and they will be periodically shut-down and rebooted, hence stressing the Power-Negotiation functionality between blades and ShMC).

3. Architecture and components of the SIT

Figure System Integration Testing Architecture depicts the overall SIT architecture and its components.

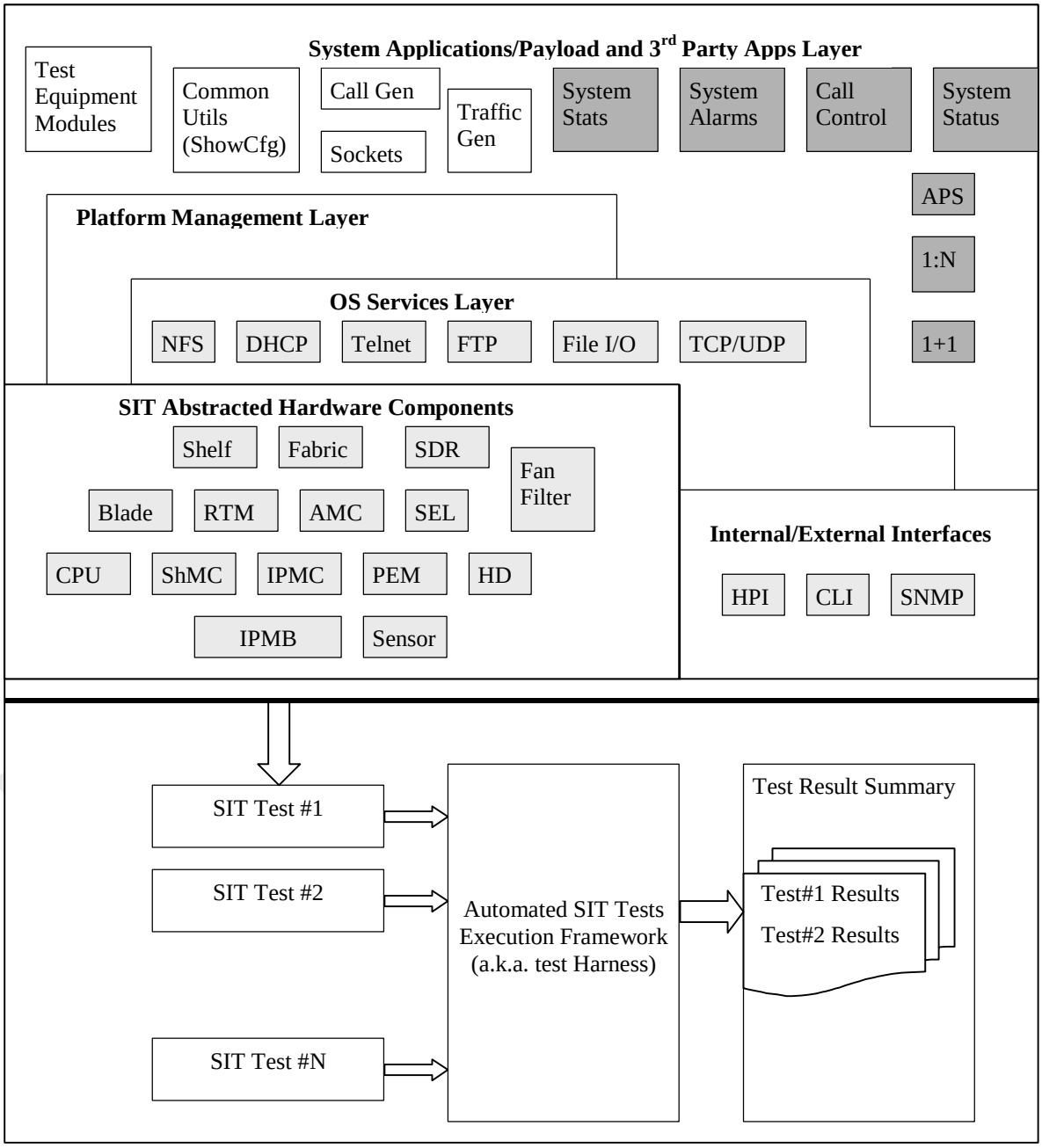

Figure 1 System Integration Testing Architecture

Note: Shaded blocks represent abstracted system or product components.

The SIT architecture consists of the following entities:

1 SIT Hardware Components abstracted in the software modules such as: Shelf, Fabric, Blade, CPU, Hard Drive, IPMC, Shelf Manager, SDR, SEL, FRU, etc... These are the elements of the system under test. The

test cases are going to perform some action on them, e.g. mount hard drive, read, write, send command, reset, clear, etc... These commands are captured as subroutines/methods within the modules. Depending on implementation, the software module may store some permanent or temporary data in their local variables/data storage or data may be returned to the caller of the method, or data may be stored in a dedicated file. If an asynchronous interaction is required between the main test case modules and the SIT Hardware Components, then some appropriated mechanism has to be in place to handle it (e.g. polling for results, or event generation, or call back routines). The software abstraction of System Hardware Components shall be implemented with inheritance in mind as much as the language used for implementation may allowed.

Note: Please, refer to corresponding SIT Modules design document for detail description of the modules. These documents are used as a guideline for the modules implementation.

2 Internal and External System Interfaces – these are software modules that provide an API to the test cases in order to use system interfaces such as: SNMP, RMCP, HPI, CLI... The idea is that a test case shall use the common API and specify the interface and command that it's going to exercise. The underlying System Interface SW modules shall take care of formatting the requests in corresponding protocol and send them to the system under test

3 OS Services Layer– captures broad services, supported by the OS, required by the test cases to connect and interact with the system under test. The typical services abstracted in this category are: Telnet, Tftp, NFS, File system handling, Web services. Platform Management and System Applications layers use the OS services. In some cases the Platform Management and System Applications may directly utilize the SIT hardware components.

4 Platform Management Layer – it captures common routines used to verify at the system level the underlying platform and hardware.

5 System Applications/Payload and 3rd Party Applications Layer – at this layer we abstract all the applications that may run on top of the OS and use the OS services. The following are major subgroups of these layer:

- System Applications - System Stats, System Alarms, Call Control, System Interfaces Status collection, different type of redundancy and protections.
- Third party applications or in-house developed or enhanced applications that use OS services (e.g. sockets, traffic generators, call generators…) to produce some well known stress or security scenario (e.g. SYM flooding, …).
- Common utilities used in SIT for discovering the configuration of the system under test and to perform common operations on the system under test (e.g. sw/fw update) are abstracted at this layer. They utilize OS services as well.
- Test Equipment Modules – modules used for communication with external test equipment. They usually run on top of the OS.

6 SIT Test Cases – they are coded SIT test procedures as depicted in SIT Test Case Template. They consist of the steps outlined in procedures, and they utilize other software modules to execute the instructions. They are enlisted in the test harness framework and are executed automatically. SIT tests are organized in corresponding SIT Test Case folders that depict the scope of the particular SIT testing. There are the following folders:

FunctionalVerification

-Integration

-Stress

-LimitTests - CfgSize

Performance

- Response

- Throughput

- Latency

Conformance

High Availability

Regulatory

Mechanical

Thermal

Power

7 Automated SIT Test Execution Framework – it is a tool that automatically executes enlisted test cases. It is described in detail in corresponding Automated SIT Test Execution Framework requirements document. Here, we briefly describe its functionality within overall SIT architecture.

3.1 Automated SIT Tests Execution Framework (a.k.a. Test Harness)

The Test Harness is a separate SIT product that is used to provide automatic execution of multiple SIT test cases over user specified time or number of iterations. The Test Harness executes all tests enlisted in the test order file for the specified number of iterations.

The Test Harness program is started from the command line. A configuration file is passed as an argument. It specifies the SIT test configuration that is used during the run (e.g. number of chassis, number of SCMs, CPUs, ...), directory path to store the summary results of the run, the path to the test run environmental variables, etc.... Each test case also specifies its own requirements for the number of chassis, blades, HDs, CPUs, etc..., and if that number is greater than the number specified in the Test Harness configuration file, particular test case will not be executed even though it is scheduled for the run. If the number is less than the number specified in the configuration file, then the test is executed against the test case hardware requirements.

The Harness program also scans for the test order file. That file contains the list of the SIT test cases that will be executed during the run.

Before any test execution, the Harness Test checks the version of the software and firmware (read from the configuration file) for all the blades, CPUs, IPMCs, ShMCs, etc..., in the system, downloads a new image if necessary, and performs cold power-up of the system under test.

The Test Harness program assumes that the user lists the tests in the right order, to accommodate dependencies between the test cases (e.g. post-test conditions of one test is pre-test condition of another test).

The summary of the all test results are posted on the RadiSys Corporation Intranet and has the following format:

Table 2. Overall Test Results

Test Status	Count
Successful Tests	
Conditionally Successful Tests	
Failed Tests	
Skipped Tests	
Tests skipped cause lack of HW resources	
Total	

Clicking on one of the test categories brings you to Test Harness Current Results for each executed Test Case. Failures are also logged into ClearQuest.

Table 3. Current Test results

Date	Name	Output	Error Log	Source

Note: As a result of the test scripts execution the following problems/issues are reported in the problem tracking system: 1. System under test software, firmware or hardware functional errors 2. Test Script implementation errors 3. Test Harness implementation errors

4. SIT Test Configurations

The SIT test plan document shall describe all test cases and the configuration that are used for System Integration Testing. The test configurations descriptions are placed at the beginning of the test plan document and referred by the SIT Test Cases.

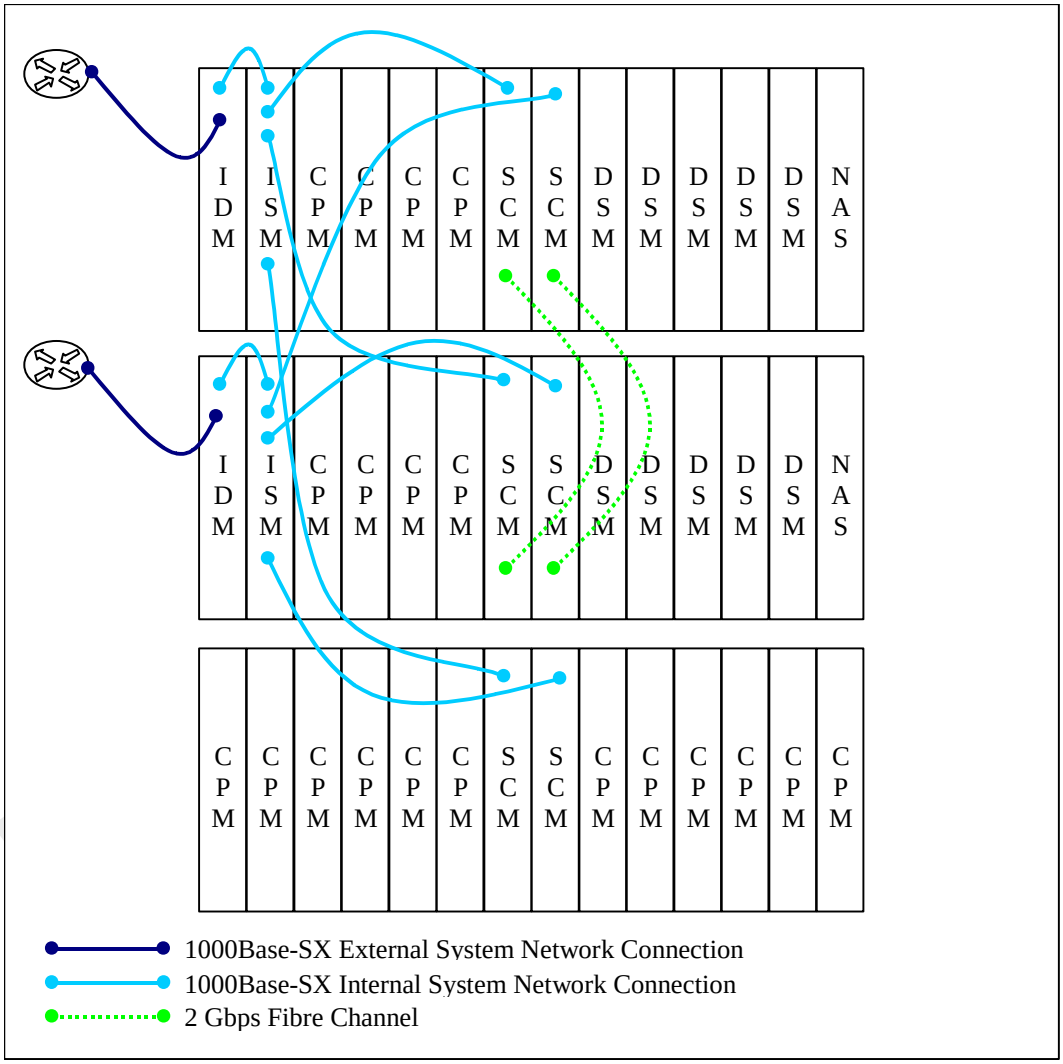

Figure 2 An illustration of a SIT Test Case Configuration

5. SIT Test Case Template

Table 4. SIT Test Case Template

Test ID			
Reference Doc:	Reference Section	Priority	Configuration
Test Purpose			
External SW/Test Equipment required			
Test Methodology			
Pre Test Conditions			
Test Procedure			
Step#	Action	Expected Behavior	Pass/Fail Criteria
Post-Test Condition			
Test Variations			

Table 13 describes a template used for writing System Integration Test Cases.

1 Test ID is a unique number that identifies a particular test. It encapsulates an absolute requirement number defined in a corresponding functional specification, e.g. Test ID: 124. The Test ID has to be unique during the lifetime of a product. It shall not be renumbered across multiple releases.

2 The Reference Document describes the document that defines a particular requirement, which in most cases will be PRD or a functional specification, e.g. Reference Doc: AdvancedTCA BladeServer Product Requirements Revision 0.86.

3 Reference Section points to the section number and its title, within the reference document, where the requirement is described.

4 Priority defines the test plan priority status. It assumes the following values: High, Moderate, Low, or Deprecated.

5 Configuration field points to one of many system test configurations that are defined at the beginning of test plan documents.

6 Test Purpose describes what is the purpose of the test, e.g. what type of system functionality the test is trying to verify.

7 Pre-Test Conditions describes the starting conditions of the system before the execution of the current test. This field is beneficial for the tests that run in a sequence and do not start from the “powered-up” system. This may reduces the time required for running multiple automated tests.

8 Test Methodology describes at a high level the approach (answers the question “How”) we are taking to verify the functionality outlined in the test plan.

9 Test Procedure field describes detailed instructions grouped into steps that are carried out during the test execution. Each executed step would result in correct or incorrect behavior that shall be observed and

recorded. The Pass/Fail criterion defines the result or behavior that defines the step as a pass or failure. Sometimes, even though a particular step does not result in correct behavior, the whole test may pass.

10 Test Variations specify different set of parameters or hardware that will be used for subsequent runs.

6. SIT Test Case Review Procedure

The purpose of the review process of the SIT test cases is to identify problems with the test procedures before they get coded and implemented. The focus of the review process is:

1 To validate the approach/methodology used to verify particular requirements or functionalities

2 To identify the effectiveness of the proposed approach (e.g. is there a more efficient way to verify the same functionality?)

3 To verify that proposed test configuration is the most appropriate with respect to the testing goals, development effort, equipment availability, equipment cost, etc…

4 To validate that proposed tests Pas/Fail criteria is correct

5 To verify that the procedures steps are detailed enough and correct to proceed with implementation

6.1 The Review Team

The SIT review team consists of at least three (4 preferably) members who would assume the following five formal roles: Moderator, Reader, Recorder, Inspectors and the Author.

The Moderator, who must be very competent technically, leads the inspection process. He or she paces the review meeting, and follows up on rework (if any).

The Reader takes the team through the test cases by paraphrasing its operation. Author shall never take this role, since he may read what he meant instead of what he defined.

A Recorder notes each problem on a standard form. This frees the other team members to focus on thinking deeply about the test cases.

The Author deals with scheduling a meeting place and disseminating materials before the meeting. During the review, the Author's role is to understand the problems found and to illuminate unclear areas.

Inspectors points to the problems as the Reader covers the sections of the test cases. Each member of the review team is at the same time an inspector, including the Author.

6.2 The Review Process

The test cases review is a process consisting of several steps; all are required for optimal results. The steps are:

Planning - When all (or at least significant number) of the test cases are completed the Author identifies the moderator (usually more senior member in the SIT team) for the review process. The Author and the Moderator forms a review team. The Author distributes the test case document to each team member, as well as other related documents such as system requirements and documentation. The Author schedules the meeting(s) time, room(s), and other resources required for the review.

Overview - This optional step is a meeting for cases where the inspection team members are not familiar with the project. The Author provides enough background to team members to facilitate their understanding of the SIT testing.

Preparation - Inspectors individually examine the test cases and related materials. They use a checklist to ensure they check all potential problem areas. Each inspector marks up his or her copy of the test cases with suspected problem areas.

Inspection Meeting - The entire team meets to review the test cases. The Moderator runs the meeting tightly. The only subject for discussion is the tests under review; any other subject is simply not appropriate and not allowed.

The person designated as Reader reads through the test cases and calls for the comments.

The Recorder summarizes the comments, discussion and resolution for each encountered problem in the test case.

The inspectors use a checklist to be sure they are looking at all important items.

All errors are recorded and classified as a Major or Minor.

Two forms get filled out. The “Test Cases Review Check List” is a summary of the errors of each of the test case that’s reviewed. The “Review Error List” is the details of each error requiring rework.

Rework - The Author makes all suggested corrections, gets a new version of the document, and request a new review meeting if that is what was agreed upon. He may also request electronic review.

Follow-up - The Moderator checks the rework. If the Moderator is satisfied the inspection is formally completed.

6.3 Review Check List

Table 5. Test Case Review Check List

Test ID	**<ID number>**
	Approached Method is correct
	Approached method is effective
	Test configuration is adequate
	Pas/Fail criteria is clear and correct
	Steps are detailed and correct
Test ID	**<ID number>**
	Approached Method is correct
	Approached method is effective
	Test configuration is adequate
	Pas/Fail criteria is clear and correct
	Steps are detailed and correct

6.4 Review Error List

The Review Error List shall be appended at the end of the test case document. The following is the proposed format of the Review Error List:

Table 6. Review Error List

Project: <>	**Release: <>**
Review Date	

Table 6. Review Error List

Project: <>	**Release: <>**			
Role	Name			
Author				
Moderator				
Reader				
Recorder				
Inspector				
Action Request	Description	Who	Due	Status
Test ID / Comment/Error #	Description		Major	Minor

7. SIT Test Case Results

The results of SIT Tests shall be stored in a log file. It is responsibility of each test case to record the results of its run in a file. If the test is ran in debugging mode, each executed step (as described in SIT Test Case Template) shall output the Pass/Fail criteria in corresponding log file. That will help in debugging and understanding of the test runs. At the end of the test execution (regarding of the execution mode), the following output shall be recorded in the log file:

<Status of the run>: Test Name

#####<number of> max counted value#####

#####<number of> min counted value#####

#####<number of> average counted value#####

<number of> successful

<number of> conditionally successful

<number of> failed

<number of> skipped

8. SIT Test Script Documentation

The test plan documentation shall be included immediately after the test case file header. It shall contain:

1. Test ID
2. Test Name
3. Reference Document
4. Reference Document Section
5. Status
6. Purpose
7. Test Configuration
8. Test Methodology
9. Author
10. Version
11. Pass/Fail Criteria
12. Notes (optional)

Here is an example of the test plan documentation:

=header2 perldoc

Test ID: 274

Test Name: Automation script for Shelf Mangement Power Negotiation Test 2.3.1

Reference Document: ATCA BladeServer System Functional Requirements

Reference Document Section: 2.3.1

Status: Active

Purpose: To verify Shelf Management Power Negotiation Functionality with the full blown chassis

Test Configuration:

Two chassis

Chassis 1:

4 APC's should be installed in slots 1,6,9, and 14

8 DSMs shall be installed in slots 2,3,4,5,10,11,12,13

SCM can be installed in slots 7 or 8

Chassis 2:

4 APC's should be installed in slots 1,6,9, and 14

8 DSMs shall be installed in slots 2,3,4,5,10,11,12,13

SCM can be installed in either slot 7 or 8

Additional SW and/or equipment required: SmartBits

Test Methodology:

Reset several time all the ATCA blades in the chassis. Change the power requirements for several blades so that we exceed the chassis power budget.

Make sure that those blades do not come up and that ShMC are operational, by

looping Ethernet traffic through powered-up blades and between 2 systems.

Log the powering sequence and status.

Log the traffic stats.

Pass Criteria:

Author: CGA

Ver..: 1.0

Date: 02/08/03

NOTE: MUST BE ROOT TO FLOOD PING

=

9. SIT Test Scripts Review Procedure

The purpose of the review process of the SIT test scripts is to identify implementation issues with the test procedures before they get tested on the system. The focus of the review process is:

1 To validate that the test case steps and instructions were properly coded

2 To validate completeness of the test scripts

3 To identify the effectiveness and modularity of the test scripts

4 To identify the performance deficiency of the test scripts

5 To verify that the system communication interfaces are properly initialized and established

6 To verify that the script modules interaction is complete and that proper mechanism for data sharing exists

7 To verify the code is done according to coding standard

8 To verify that Pass/fail logic is properly coded

9 To identify any bad programming practice

9.1 The Review Team

The scripts review team consists of at least three (4 preferably) members who would assume the following five formal roles: Moderator, Reader, Recorder, Inspectors and the Author.

The Moderator, who must be very competent technically, leads the inspection process. He or she paces the review meeting, and follows up on rework (if any).

The Reader takes the team through the script by paraphrasing its operation. Author shall never take this role, since he may read what he meant instead of what he implemented.

A Recorder notes each problem on a standard form. This frees the other team members to focus on thinking deeply about the code.

The Author deals with scheduling a meeting place and disseminating materials before the meeting. During the review, the Author's role is to understand the problems found and to illuminate unclear areas.

Inspectors points to the problems as the Reader covers the sections of the code. Each member of the review team is at the same time an inspector, including the Author.

9.2 The Review Process

The SIT script review is a process consisting of several steps; all are required for optimal results. The steps are:

Planning - When an SIT module(s) or test scripts are code complete (before any testing on the target) the Author identifies the moderator (usually more senior member in the SIT team) for the review process. The Author and the Moderator forms a review team. The Author distributes the code to each team member, as well as other related documents such as test case documentation. The Author schedules the meeting(s) time, room(s), and other resources required for the review.

Overview - This optional step is a meeting for cases where the inspection team members are not familiar with the SIT modules or scripts. The Author provides enough background to team members to facilitate their understanding of the SIT testing.

Preparation - Inspectors individually examine the code and related materials. They use a checklist to ensure they check all potential problem areas. Each inspector marks up his or her copy of the test scripts with suspected problem areas.

Inspection Meeting - The entire team meets to review the code. The Moderator runs the meeting tightly. The only subject for discussion is the code under review; any other subject is simply not appropriate and not allowed.

The person designated as Reader reads through the code and calls for the comments. The Reader continuously decides how many lines of code to paraphrase, picking a number that allows reasonable extraction of meaning. Typically he's paraphrasing 2-3 lines at a time. He paraphrases every decision point, every branch, case, etc.

The Moderator must keep the meeting fast-paced and efficient. A reasonable review rate is between 150 and 200 non-blank lines per hour.

Note that comment lines require as much review as code lines. Misspellings, poor grammar, and poor communication of ideas are as deadly in comments as outright bugs in code.

The Recorder summarizes the comments, discussion and resolution for each encountered problem in the test case.

The inspectors use a checklist to be sure they are looking at all important items.

All errors are recorded and classified as a Major or Minor.

Two forms get filled out. The "Test Cases Inspection Checklist" is a summary of the number of errors of each type that's found. It's used for understanding how effective the inspection process is. The "Inspection Error List" is the details of each error requiring rework.

Rework - The Author makes all suggested corrections, implements the changes, and request a new review meeting if that is what was agreed upon. He may also request an electronic review.

Follow-up - The Moderator checks the rework. If the Moderator is satisfied the inspection is formally completed.

9.3 Review Check List

Table 7. Test Script Check List

Test ID		**<ID number or Module Name>**
Number of Errors		Error Type
Major	**Minor**	
		Code does not meet coding standard
		Method/Function size and complexity unreasonable
		Unclear expressions of ideas in the code
		Poor encapsulation
		Poor logic
		Poor commenting
		Error condition not caught
		Incorrect Syntax
		Performance issues
		Other

9.4 Inspection Report

The Inspection Reports shall be submitted to SIT Lead responsible for particular product and release. The SIT lead shall submit the inspection report to corresponding product/release Lotus Notes **teamroom**. The following is the proposed format of the Inspection Report:

Table 8. Inspection Report

Project: <>	**Release: <>**			
Script file name	<Name>			
Inspection Date	<Date>			
Role	Name			
Author				
Moderator				
Reader				
Recorder				
Inspector(s)				
Action Request	Description	Who	Due	Status
Line number	Error Type / Description		Major	Minor

10. Coding Standard

This chapter recommends some coding conventions for development of the scripting code and shall be followed as much as possible. Following the coding standard provides uniform code and reduces the maintenance and code inspection effort. However, at times there may be need to be inconsistency and to break the rule. In such occasions the most important thing is to know when and why we are inconsistent and to feed back these exceptions to the coding standard.

Two good reasons to break a particular rule:

1 When applying the rule would make the code less readable, even for someone who is used to reading code that follows the rules.
2 To be consistent with surrounding code that also breaks the rules (maybe for historic reasons), though this may be an opportunity to clean up pre-dated implementation.

10.1 Code lay-out

10.1.1 Indentation/Column Alignment

The indentation of code provides quick, visual feedback regarding the scope and nesting level of each section of code. Column alignment of code, as well as comments, allows the eye to follow down a fixed line as code is read, rather than having to jump around to find important code items. Further, subordinate code, as well as the end of a particular statement, should be readily identifiable from opening brace to closing brace (see below).

The number of spaces to indent each subsequent level of coding should be consistent.

The choice of four (4) spaces indentation is recommended for indentation level. This choice provides:

1 Better visual feedback for code subordination
2 Better alignment with if statements
3 Better control of run-away nesting of code

If the inherited or surrounded code uses tabs instead of spaces, or mixes tabs and spaces, the code should be converted to using spaces exclusively.

10.1.2 Maximum Line Length

All lines should be ended within 80 columns, whenever possible, to ease the readability of code. This allows all text to be visible on the screen without having to scroll horizontally. In addition, try to leave the 80th column blank for the cursor to reside to prevent horizontal scrolling.

It has been pointed out that current screen and editing technology allows use of wider text and 80 columns can sometimes seem confining. While this may be true, the following factors suggest benefits to staying within 80 columns:

- Printed files are more easily handled in 8.5 x 11 portrait mode.
- Wider code windows take up more screen real estate. Code windows are often viewed side by side.
- Some editors (i.e. Slick Edit) demonstrate unexpected horizontal text orientation when switching between

files that are wider than the code window.

- Intended code review methodology will be facilitated by the 80 characters limit.

The preferred way of wrapping long lines is by using language specific implied line continuation inside parentheses, brackets and braces. If necessary, you can add an extra pair of parentheses around an expression, but sometimes using a backslash (for Python and C) looks better. Make sure to indent the continued line appropriately.

10.1.3 Imports

Imported files or modules should always be put at the top of the file, just after any module comments and docstrings, and before module globals and constants. Imports should be grouped, with the order being

1 standard library imports

2 related major package imports

3 application specific imports

A blank line should be put between each group of imports.

Relative imports for intra-package imports are highly discouraged. Always use the absolute package path for all imports.

10.1.4 White Space in Expressions and Statements

The appropriate use of white space is extremely important in the readability of programs. White space is simply blank space in the code. It can be blank lines or blank space mixed into a line. It servers to break code into pieces the eye can readily identify.

All too often, the code runs on and on, with no breaks. Consider how you would feel reading this document if there were no headings and the sentences were strung together as one big paragraph. It would be pretty boring and very difficult.

Do not use white spaces:

1 immediately inside parentheses, brackets or braces,

2 immediately before a comma, semicolon, or colon,

3 immediately before the open parenthesis that starts the argument,

4 immediately before the open parenthesis that starts an indexing,

Always surround these binary operators with a single space on either side: assignment (=), comparisons (==, <, >, !=, <>, <=, >=), Booleans (and, or, not).

10.1.5 Comments

General Rules

Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes.

Comments shall be complete sentences.

If a comment is a phrase or sentence, its first word should be capitalized, unless it is an identifier that begins with a lower case letter.

If a comment is short, the period at the end is best omitted.

Block comments generally consist of one or more paragraphs built out of complete sentences, and each sentence should end in a period.

Block Comments

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (unless it is indented text inside the comment).

Paragraphs inside a block comment are separated by a line containing a single "#".

Block comments are best surrounded by a blank line above and below them (or two lines above and a single line below for a block comment at the start of a new section of function definitions).

Inline Comments

In-line comments are those comments contained on the same line as instructions. They are brief, by nature, due to limited space. They must be worded in a very concise manner. These comments greatly increase program readability because they tell the reader what the instruction does without him/her having to decipher the code. Even when the code meaning may seem obvious, an in-line comment helps the reader scan the code faster. Even when you (the author of the code) think the code is easy to understand, an in-line comment helps those not familiar with the program.

Inline comments should be separated by at least two spaces from the statement.

They should start with a # and a single space.

Inline comments are unnecessary and in fact distracting if they state the obvious.

File Headers

All SIT source files must contain a File Header, as shown below. It serves multiple purposes. It provides copyright information for the file, it identifies that the correct file has been opened, it gives the casual reader an idea of the purpose of the file, and identifies major changes made to the file after completion of the program.

```
################################################################################
#
# RadiSys Corporation
# <Place a product name here>
# (C) Copyright RadiSyS Corporation 2004 All Rights Reserved.
#
# The source code for this program is not published or otherwise divested of its trade secrets,
# irrespective of what has been deposited with the U.S. copyright office.
#
# %Z% %I%
#
# File Name:   <filename and type>
#
# Description: <description goes here>
#
# Change Activity:
# xxx mm/dd/yy <place a brief description of the change here>
#
################################################################################
```

Function Headers

Every function/subroutine must begin with a Function Header, as shown below. The use of such a header provides the following important benefits:

1 ·Provides a strong eye catcher for scanning code.

2 ·Provides a description of why the function is called and what it does.

3 ·Clearly identifies input parameters and return values.

The Parameter section should provide the details of the individual parameters. If the descriptions of the parameters can be given on the same lines as the parameters themselves, then the Parameter section can read: “See list below”.

The Function Header below shows the format of the header.

```
################################################################################
#
# Function:    <function name>
#
# Description: <place a description of the function here>
#
# Parameters:  <describe parameters or refer to comments on parm lines>
#
#
# Returns:     <describe the value(s) returned>
#
################################################################################
```

10.1.6 Documentation Strings

Write documentation strings for all public modules, subroutines/functions, modules/classes, and methods. The documentation shall be included above the modules, subroutines/functions, modules/classes, and methods implementation, but below their headers.

10.1.7 Naming Convention

Appropriately naming program elements (functions, variables, modules, classses, etc.) can contribute greatly toward readability and maintainability of programs. The benefits of a naming convention are fairly well agreed upon. However, the implementation aspects are subject of considerable debate, academic and otherwise.

Descriptive: Naming Styles

There are a lot of different naming styles. It helps to be able to recognize what naming style is being used, independently from what they are used for. The following naming styles are commonly distinguished:

1 b (single lowercase letter)

2 B (single uppercase letter)

3 lowercase

4 lower_case_with_underscores

5 UPPERCASE

6 UPPER_CASE_WITH_UNDERSCORES

7 CapitalizedWords (or CapWords, or CamelCase -- so named because of the bumpy look of its letters). This is also sometimes known as StudlyCaps.

8 mixedCase (differs from CapitalizedWords by initial lowercase character)

9 Capitalized_Words_With_Underscores

There's also the style of using a short unique prefix to group related names together. For example, the X11 library uses a leading X for all its public functions. In addition, the following special forms using leading or trailing underscores are recognized (these can generally be combined with any case convention):

- _single_leading_underscore: weak "internal use" indicator.

- single_trailing_underscore_.

- __double_leading_underscore: class-private names.

- __double_leading_and_trailing_underscore__: "magic" objects or attributes that live in user-controlled namespaces, e.g. __init__, __import__ or __file__. Sometimes these are

defined by the user to trigger certain magic behavior (e.g. operator overloading); sometimes these are inserted by the infrastructure for its own use or for debugging purposes. Since the infrastructure may decide to grow its list of magic attributes in future versions, user code should generally refrain from using this convention for its own use. User code that aspires to become part of the infrastructure could combine this with a short prefix inside the underscores, e.g. __bobo_magic_attr__.

Prescriptive: Naming Conventions

Names to avoid

Never use the characters `l' (lowercase letter el), `O' (uppercase letter oh), or `I' (uppercase letter eye) as single character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use `l' use `L' instead.

Function/Subroutine Names

Function names should be mixedCase. The name shall describe performed operation within a function. Each new word in a function name shall start with the capital letter, e.g. matchFileSubset.

Module Names

Modules should have short, lowercase names, without underscores. Since module names are mapped to file names, and some file systems are case insensitive and truncate long names, it is important that module names be chosen to be fairly short -- this won't be problem on Unix, but it may be a problem when the code is transported to Mac or Windows.

Class Names

Almost without exception, class names use the CapWords convention. Classes for internal use have a leading underscore in addition.

Local Variable Names

Local names are defined using mixedCase convention.

Global Variable Names

Global names are defined using mixedCase convention, prefixed with lowercase letter “g”.

Method Names and Instance Variables

The story is largely the same as with functions: in general, use mixedCase.

Use one leading underscore only for internal methods and instance variables which are not intended to be part of the class's public interface.

Designing for inheritance

Always decide whether a class's methods and instance variables should be public or non-public. In general, never make data variables public unless you're implementing essentially a record. It's almost always preferable to give a functional interface to your class instead.

Also decide whether your attributes should be private or not. The difference between private and non-public is that the former will never be useful for a derived class, while the latter might be. Yes, you should design your classes with inheritance in mind.

Private attributes should have two leading underscores, no trailing underscores.

Non-public attributes should have a single leading underscore, no trailing underscores.

Public attributes should have no leading or trailing underscores, unless they conflict with reserved words, in which case, a single trailing underscore is preferable to a leading one, or a corrupted spelling, e.g. class_ rather than klass.

A References

A.1 Related Documents

A.1.1 RadiSys Documents

DocName. Publisher, Month Day, Year.

A.1.2 Other Documents

DocName. Publisher, Month Day, Year.

A.1.3 Industry-Standard References

Table 9. Industry-Standard References

Specification	Description	Location
ACPI	Advanced Configuration and Power Interface specification	www.acpi.info
APM	Advanced Power Management specification	www.microsoft.com/hwdev/archive/BUSBIOS/amp_12.asp
Intel 815(E) Chipset	Intel 815(E) chipset datasheet	http://developer.intel.com/design/chipsets/embedded/273428.htm
Intel C-ICH	Intel Communications I/O Hub datasheet	http://developer.intel.com/design/chipsets/embedded/273573.htm
Intel Celeron processor	Intel Celeron processor datasheet	http://developer.intel.com/design/celeron
Intel Pentium III processor	Intel Pentium III processor datasheet	http://developer.intel.com/design/pentiumiii
ATX, microATX	Form factor specifications	www.formfactors.org
PCI	PCI local bus specification	www.pcisig.com
SDRAM DIMMs	PC SDRAM module specification	http://developer.intel.com/technology/memory/
SMBus	System management bus	www.smbus.org
USB	Universal Serial Bus specification	www.usb.org/developers
VESA	Video Electronics Standards Association	www.vesa.org

A.2 Terms

Term Description (use “PM1 Parameter” paragraph style)

Acronym (Unabridged Acronym) Description (use “PM1 Parameter” paragraph style)

B Requirements Matrix

This section lists applicable product requirements and identifies where this specification addresses each requirement.

Table 10. Requirements Matrix

Requirement	Section Addressed	Comments