The Open Group Guide

Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 1: General Guidance

Future Airborne Capability Environment

You have a choice: you can either create your own future, or you can become the victim of a future that someone else creates for you. By seizing the transformation opportunities, you are seizing the opportunity to create your own future.

Vice Admiral (ret.) Arthur K. Cebrowski

The Open Group Guide

Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 1: General Guidance

ISBN:	1-947754-57-7
Document Number:	G209 (Volume 1)

Published by The Open Group, May 2020.

Comments relating to the material contained in this document may be submitted to:

The Open Group, 800 District Avenue, Suite 150, Burlington, MA 01803, United States

or by electronic mail to:

ogface-admin@opengroup.org

Contents

1 Introduction ... 1

1.1 Background ... 1

1.2 FACE Approach ... 1

1.3 Scope of this Document ... 2

2 Portable Components Segment (PCS) ... 3

2.1 Scope ... 3

2.2 Migrating a PCS UoC from Prior Versions ... 3

2.3 Key Characteristics ... 3

2.4 Configurability ... 4

2.5 Design Alternatives ... 5

2.6 General PCS UoC Guidance ... 5

2.6.1 Using Transport Services ... 5

2.6.2 Life Cycle Management Services ... 5

2.6.3 Component State Persistence ... 5

2.6.4 Component Frameworks ... 5

2.6.5 Security Transformations ... 6

2.6.6 Injectable Interface ... 6

3 Platform-Specific Services Segment (PSSS) ... 7

3.1 Scope ... 7

3.2 Migrating a PSSS UoC from Prior Versions ... 7

3.3 General PSSS UoC Guidance ... 8

3.3.1 Using Transport Services ... 8

3.3.2 Using I/O Services ... 8

3.3.3 Life Cycle Management Services ... 8

3.3.4 Component State Persistence ... 8

3.3.5 Component Frameworks ... 8

3.3.6 Security Transformations ... 9

3.3.7 Injectable Interface ... 9

3.4 PSDS UoC Guidance ... 9

3.4.1 Key Characteristics ... 9

3.4.2 Configurability ... 10

3.4.3 Design Alternatives ... 10

3.5 PSCS UoC Guidance ... 10

3.5.1 Centralized Logging Services ... 11

3.5.2 Device Protocol Mediation ... 12

3.5.3 Streaming Media ... 13

3.5.4 Centralized Configuration Services ... 14

3.5.5 System-Level Health Monitoring ... 15

3.6 PSGS UoC Guidance ... 15

4 FACE Profile Guidance ... 16

4.1 Introduction ... 16

4.2 Strategy 16
4.3 Migration Between FACE Profiles 17

5 Graphics Services 18
5.1 Purpose 18
5.2 Introduction 18
5.2.1 Summary of Changes from Edition 2.1 18
5.2.2 How to Read this Section 19
5.2.3 Graphics Portability Considerations 19
5.2.4 Relationship to the FACE Reference Architecture 21
5.3 Graphics Services 22
5.3.1 ARINC 661 22
5.3.2 OpenGL 24
5.3.3 ARINC 739 26
5.4 Graphics Display Management Services 27
5.4.1 Introduction 27
5.4.2 Technical Description 28
5.4.3 Usage Examples 29
5.5 Graphics Requirements Decomposition 37
5.5.1 OSS Requirements for Graphics Services 37
5.5.2 PCS Requirements for Graphics Services 38
5.5.3 PSSS Requirements for Graphics Services 43

6 Safety Guidance 49
6.1 Safety Overview 49
6.2 Purpose 50
6.3 Applicability 51
6.4 Safety Assessment and Design Assurance Level 51
6.4.1 Start Assessment Very Early and Update as the Program Progresses 51
6.5 DAL and Qualification Cost are Directly Proportional 52
6.5.1 Assess Criticality During First Application, or as Part of a Notional Application 52
6.5.2 Plan for Future Use of Functionality at a Higher DAL 52
6.5.3 Safety Assessment May Vary by Context 53
6.5.4 Plan for the Most Critical Application 53
6.5.5 Safety Assessment of Configurable Software 53
6.6 Qualification Advantage of Various Software Reuse Strategies 54
6.6.1 Basic Principles of Software Qualification 54
6.6.2 Additional Qualification Considerations 55
6.7 Overview of the Principles of Portable Software 56
6.7.1 Documentation of Early Agreement 56
6.7.2 Specific Achievements for Portable Software during the First Qualification 56
6.7.3 Achievements Remaining for Re-Application of Portable Software 57
6.7.4 Qualification of Portable Software as Part of a System Qualification 57
6.7.5 Separate Documentation Requirements for Portable Software 57

6.8 Types of Reuse 57
6.8.1 Binary Reuse 57
6.8.2 Source Code Reuse 58
6.8.3 Reuse of Artifacts 58
6.9 Types of Artifacts 58
6.9.1 Requirements 58
6.9.2 Tests 59
6.9.3 Traceability 60
6.9.4 IRS/IDD/ICD 60
6.9.5 Miscellaneous Software Documentation 61
6.9.6 Software Configuration Index or Version Description Document 61
6.9.7 Software Accomplishment Summary or Qualification Summary 61
6.10 Reuse of Tests 62
6.10.1 Reuse of Manual Tests 62
6.10.2 Reuse of Automated Tests 62
6.11 Consideration of Archival of Artifacts in the FACE Repository or Reference 63
6.12 Coordination of Qualification 64
6.12.1 Early Agreement and Documentation Facilitates Efficient Qualification 64
6.13 Segregation of Software Components 64
6.13.1 Segregation for Variability 65
6.13.2 Segregation for Criticality 65
6.13.3 Spatial/Temporal Partitioning (ARINC 653) 66
6.14 Limiting Qualification Impacts Through Configurability 67
6.14.1 Safety and Configuration Files (using Deactivated Code) 67
6.14.2 Pre-Qualified Selection of Options at Run-Time (including Initialization) 68
6.14.3 Dead Code *versus* Deactivated Code 68
6.14.4 Application of Qualification Requirements to Configuration Mechanisms 69
6.15 Test Programs 69
6.15.1 Requirements-Based Testing 70
6.15.2 Low-Level Software Requirements Testing 70
6.15.3 High-Level Software Requirements Testing 70
6.15.4 System Integration Testing 70
6.15.5 Hardware Integration Testing 71
6.15.6 Configuration and Setup Testing 72
6.15.7 Data Parameter Configuration Testing 72
6.15.8 Information Interface Testing 72
6.15.9 Type Consistency Testing 72
6.15.10 Protocol Header Testing 72
6.15.11 Data Integrity Testing 72
6.15.12 Robustness Testing 72
6.16 Life Cycle Support 74
6.16.1 Distribution of Software Maintenance Releases 74
6.16.2 Collection of Problem Reports 74
6.16.3 Dissemination of Problem Reports 74

6.16.4 Software Loading Instructions 75
6.16.5 Software Compilation Instructions 75
6.16.6 Software User's Manual 75
6.17 Documentation 75
6.18 Model-Based Engineering of Software 76
6.19 Special Considerations for the Use of Run-Times and Frameworks 77
6.19.1 Language Run-Times 77
6.19.2 Frameworks 77
6.20 Conclusion 78

7 Security Guidance 80
7.1 Introduction 80
7.1.1 Basic Security 80
7.1.2 Security Principles 81
7.2 System Security Architecture 87
7.3 Process 87
7.3.1 Risk Management 87
7.3.2 Software Assurance 88
7.3.3 Security Controls 88
7.4 Security Design Considerations 89
7.4.1 Challenges of Reusing UoCs 89
7.4.2 Verification of System Configuration 89
7.4.3 Auditing 90
7.4.4 Privileged Processes 90
7.4.5 System State Management 91
7.5 Security Transformations 92
7.5.1 Introduction 92
7.5.2 Data Modeling Requirements for Security Transformation 92
7.5.3 Security Transformation Considerations 93
7.6 Example Scenarios – Team 94
7.6.1 Scenario B 94
7.7 Security Standard References 95
7.7.1 Software Engineering Institute Secure Coding Standard 96
7.7.2 Common Weakness Enumeration 96
7.7.3 Open Web Application Security Project 96
7.7.4 ISO/IEC 27034 96
7.7.5 Risk Management Framework for DoD IT 96
7.7.6 CMU/SEI-2009-SR-001 97
7.7.7 Program Protection Plan 97
7.7.8 Systems Security Engineering 97

8 Implementation Scenario Examples 98
8.1 Communications Manager Scenario 98
8.1.1 Introduction 98
8.1.2 Scope 98
8.1.3 Assumptions 101
8.1.4 Computing Environment 101

8.2 Digital Map Manager Scenario ... 107
8.2.1 Introduction ... 107
8.2.2 Scope ... 107
8.2.3 Assumptions ... 110
8.2.4 Computing Environment ... 111
8.3 Required Navigation Performance Manager Scenario ... 119
8.3.1 Introduction ... 119
8.3.2 Scope ... 119
8.3.3 Assumptions ... 122
8.3.4 Computing Environment ... 123
8.4 Secure Reporting Status Scenario ... 131
8.4.1 Introduction ... 131
8.4.2 Scope ... 133
8.4.3 Assumptions ... 135
8.4.4 Computing Environment ... 135

A Acronyms ... 141

List of Figures

Figure 1: Portable Components.......... 4
Figure 2: FACE Profile Diagram.......... 16
Figure 3: ARINC 661 MVC Overview.......... 22
Figure 4: PCS Graphics Services Example.......... 25
Figure 5: PSSS Graphics Services Example.......... 25
Figure 6: ARINC 739 Server Distributed Graphics Communication.......... 27
Figure 7: Compositing Architecture.......... 29
Figure 8: Multiple Graphical UoCs Screen Spaces.......... 34
Figure 9: Security Transformation Modeling Examples.......... 93
Figure 10: Security Transformation Example.......... 94
Figure 11: Scenario B Component Robustness Levels.......... 95
Figure 12: Communication Manager Operational View.......... 98
Figure 13: Communication Manager Functional View.......... 99
Figure 14: Communication Manager Physical View.......... 100
Figure 15: Communications Manager Partitioned View.......... 101
Figure 16: Digital Map Operational View.......... 107
Figure 17: Digital Map Functional View.......... 108
Figure 18: Digital Map Physical View.......... 109
Figure 19: Digital Map Partition View.......... 110
Figure 20: Required Navigation Performance Operational View.......... 119
Figure 21: Required Navigational Performance Functional View.......... 120
Figure 22: Required Navigational Performance Physical View.......... 121
Figure 23: Required Navigational Performance Partition View.......... 122
Figure 24: NSA SSE-100-1 Scenario B Reporting Status.......... 132
Figure 25: Secure Reporting Status Operational View.......... 133
Figure 26: Secure Reporting Status Functional View.......... 133
Figure 27: Secure Reporting Status Physical View.......... 134
Figure 28: Secure Reporting Status Partition View.......... 135
Figure 29: Secure Reporting Status Data Flow View.......... 140

List of Tables

Table 1: OSS Requirements for Graphics Services 37
Table 2: PCS User Application Requirements for Graphics Services 38
Table 3: PCS Cockpit Display System Requirements for Graphics Services 39
Table 4: PCS ARINC 739A Requirements for Graphics Services 41
Table 5: PCS OpenGL Requirements for Graphics Services 42
Table 6: PSGS Sub-Section Requirements for Graphics Services 43
Table 7: PSSS Cockpit Display System Requirements for Graphics Services 44
Table 8: PSSS User Application Requirements for Graphics Services 46
Table 9: PSSS ARINC 739A Requirements for Graphics Services 46
Table 10: PSSS OpenGL Requirements for Graphics Services 47
Table 11: DO-178 Design Assurance Levels 70
Table 12: Security Principles Inherent to the FACE Reference Architecture 81
Table 13: Implement System Interfaces with Least Privilege 85
Table 14: Security Transformation Segment Considerations 93

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 700 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/library.

This Document

This document is a Reference Implementation Guide to be used in conjunction with the FACE™ (Future Airborne Capability Environment) Technical Standard, Edition 3.0.

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this Guide:

Principal Authors

- Kirk Avery, Lockheed Martin
- Stephanie Burns, Collins Aerospace
- H. Glenn Carter, U.S. Army CCDC Aviation & Missile Center
- Robert Daniels, NAVAIR
- James “Bubba” Davis, U.S. Army CCDC Aviation & Missile Center
- Chris Edwards, U.S. Army CCDC Aviation & Missile Center
- Stephen Fulmer, U.S. Army CCDC Aviation & Missile Center
- Patrick Huyck, Green Hills Software
- Syltinsy P. Jenkins, NAVAIR
- William Kinahan, Skayl LLC
- Marc Moody, Boeing
- Joseph Neal, L3Harris Corporation
- Stephen Price, U.S. Army CCDC Aviation & Missile Center
- Josh Shapiro, Northrop Grumman
- Joel Sherrill, U.S. Army CCDC Aviation & Missile Center
- Scott Wigginton, U.S. Army CCDC Aviation & Missile Center
- Joe Wlad, Verocel
- Levi VanOort, Collins Aerospace

Additional Contributors

- Judy Cerenzia, The Open Group
- Christopher Crook, U.S. Army PEO Aviation
- Daniel Herring, Core AVI
- Robert Jefferies, NAVAIR

- Paul Jennings, Presagis
- Titus Marcel, Textron Systems
- Matthew Mueller, NAVAIR
- Marcus Quettan, NAVAIR
- Dudrey Smith, AdaCore
- Jeffrey VanDorp, General Electric (GE) Aviation
- Michael Vertuno, Northrop Grumman
- Mark W. Vanfleet, National Security Agency (NSA)

Referenced Documents

The following referenced documents are included for the application of this Guide. For dated references, only the edition cited applies.

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

- AC 20-148: Reusable Software Components, Federal Aviation Authority, 2004
- Aerospace Recommended Practice (ARP) 4754: Guidelines for Development of Civil Aircraft and Systems (Revision A), SAE International, 2010
- Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, SAE International, 1996
- ARINC Characteristic 739-1: Multi-Purpose Control and Display Unit (MCDU), June 1990
- ARINC Characteristic 739A-1: Multi-Purpose Control and Display Unit (MCDU), December 1998
- ARINC Report 661-5: Cockpit Display System Interfaces to User System, July 2013
- ARINC Specification 429: Mark 33 Digital Information Transfer System (DITS), May 2004
- ARINC Specification 653P1-3: Avionics Application Software Standard Interface, Part 1: Required Services, November 2010
- ARINC Specification 653P1-4: Avionics Application Software Standard Interface, Part 1: Required Services, August 2015
- ARINC Specification 653P2-3: Avionics Application Software Standard Interface, Part 2: Extended Services, August 2015
- Committee on National Security Systems (CNSS) 4009: Information Assurance (IA) Glossary, April 2015
- Common Criteria (CC) for Information Technology Security Evaluation, Version 3.1, Revision 5, April 2017 (this is equivalent to ISO/IEC 15408:2009)
- Common Object Request Broker Architecture (CORBA®) Specification, Version 3.3, published by the Object Management Group (OMG®), November 2012
- Common Weakness Enumeration, A Community-Developed List of Software Weakness Types, June 2019, retrieved from https://cwe.mitre.org/
- Data Distribution Service for Real-time Systems Specification, Version 1.2, published by the Object Management Group (OMG®), July 2001

- Design Assurance Guidance for Airborne Electronic Hardware, DO-254, published by Radio Technical Commission for Aeronautics (RTCA), April 2000
- Electronic Industries Alliance (EIA) J-STD-016-1995: IEEE Standard for Information Technology – Software Lifecycle Processes – Software Development – Acquirer-Supplier Agreement (withdrawn)
- Enterprise Integration Patterns, 2012; refer to http://www.eaipatterns.com/
- FACE™ (Future Airborne Capability Environment): Conformance Policy, Version 2.0 (X1608), September 2016, published by The Open Group; refer to: www.opengroup.org/library/x1608
- FACE™ (Future Airborne Capability Environment) Contract Guide: Guidance in Writing Solicitations and Proposals with FACE Requirements, Version 2.0 (G18D), September 2018, published by The Open Group; refer to: www.opengroup.org/library/g18d
- FACE™ (Future Airborne Capability Environment) Shared Data Model Governance Plan (X1817), June 2018, published by The Open Group; refer to: www.opengroup.org/face
- FACE™ (Future Airborne Capability Environment) Technical Standard, Edition 3.0 (C17C), November 2017, published by The Open Group; refer to: www.opengroup.org/library/c17c
- Federal Information Security Management Act (FISMA), U.S. Federal Law, 2002
- IEEE Std 1003.1-2008: IEEE Standard for Information Technology – Portable Operating System Interface (POSIX®) – Base Specifications, Issue 7, December 1, 2008
- IETF RFC 5424: The Syslog Protocol, March 2009; refer to: http://tools.ietf.org/html/rfc5424
- IETF RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog, March 2009; refer to: http://tools.ietf.org/html/rfc5425
- ISO/IEC 15408:2009: Information Technology – Security Techniques – Evaluation Criteria for IT Security; refer to: https://www.iso.org/standard/50341.html
- ISO/IEC 27034:2011: Information Technology – Security Techniques – Application Security; refer to: https://www.iso.org/standard/44378.html
- ISO/IEC/IEEE 12207:2017: Systems and Software Engineering – Software Life Cycle Processes; refer to: https://www.iso.org/standard/63712.html
- ISO/IEC/IEEE 24765:2017: Systems and Software Engineering Vocabulary; refer to: https://www.iso.org/standard/71952.html
- Leveson, Nancy G., Safeware: System Safety and Computers, Addison Wesley Publishing Company, 1995
- Long, Mohindra, Seacord, Sutherland & Svoboda, The CERT Oracle Secure Coding Standard for Java®, Addison-Wesley Professional, 2011
- National Security Telecommunications and Information Systems Security Policy (NSTISSP) No. 11: National Policy Governing the Acquisition of Information Assurance

(IA) and IA-Enabled Information Technology (IT) Products, January 2000 (revised June 2003) – NSTISSC is now known as the Committee on National Security Systems (CNSS)

- NIST Special Publication 800-37: Risk Management Framework for Information Systems and Organizations, December 2018
- NIST Special Publication 800-53: Security and Privacy Controls for Federal Information Systems and Organizations, April 2013
- NIST Special Publication 800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, November 2016
- NSA SSE-100-1: National Security Agency Information Assurance Guidance for Systems Based on a Security Real-Time Operating System (RTOS), 2009
- OSGi Service Platform, Release 7, OSGi Alliance, 2018; refer to: www.osgi.org/Release4
- OWASP™ Foundation, The Free and Open Software Security Community, August 2019, retrieved from: https://www.owasp.org/index.php/Main_Page
- Software Assurance Countermeasures in Program Protection Planning, Deputy Assistant Secretary of Defense for Systems Engineering (DASD (SE)) and Department of Defense Chief Information Officer (DoD CIO), Washington, DC, 2014
- Software Considerations in Airborne Systems and Equipment Certification, DO-178B, published by Radio Technical Commission for Aeronautics (RTCA) Inc., December 1992
- Software Considerations in Airborne Systems and Equipment Certification, DO-178C, published by Radio Technical Commission for Aeronautics (RTCA) Inc., January 2012
- Software Engineering Institute CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems, 2016
- Software Engineering Institute CERT C++ Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems, 2016
- Software Engineering Institute Special Report CMU/SEI-2009-SR-001, Making the Business Case for Software Assurance, April 2009.
- U.S. Department of Defense Directive DoDI 8500.1, Information Assurance (IA), 2002
- U.S. Department of Defense Instruction DoDI 8500.2: Information Assurance (IA) Implementation, 2003
- U.S. Department of Defense Military Standards: MIL-STD-498 (cancelled 1998) and MIL-STD-1553
- XML Metadata Interchange (XMI®) Specification, Version 2.4.2, published by the Object Management Group (OMG®), April 2014

1 Introduction

1.1 Background

Today's military aviation community airborne systems are typically developed for a unique set of requirements by a single vendor. This form of development has served the military aviation community well; however, this stovepipe development process has had some undesired side-effects including long lead times, cumbersome improvement processes, lack of hardware and software reuse between various aircraft platforms, which result in a platform-unique design.

The advent of significantly complex mission equipment and electronics systems has caused an increase in the cost and schedule to integrate new hardware and software into aircraft systems. This – combined with the extensive testing and airworthy qualification requirements – has begun to affect the ability of the military aviation community to deploy these new capabilities across the military aviation fleet.

The current military aviation community procurement system does not promote the process of hardware and software reuse across different programs. In addition, the current aviation development community has not created sufficient standards to facilitate the reuse of software components across the military aviation fleet. Part of the reason for this is the small military aviation market. Another part is the difficulty in developing qualified software for aviation. An additional problem is the inability to use current commercial software Common Operating Environment (COE) standards because they do not adhere to the stringent safety requirements developed to reduce risk and likelihood of loss of aircraft, reduced mission capability, and ultimately loss of life.

1.2 FACE Approach

The Future Airborne Capability Environment (FACE) was designed to address the affordability initiatives of today's military aviation community. The approach used by The Open Group FACE Consortium is to develop a Technical Standard for a software COE designed to promote portability and create software product lines across the military aviation community.

Several components comprise the FACE approach to software portability and reuse:

- Business processes to adjust procurement and incentivize industry
- Promote development of reusable software components
- A software COE standard to promote the development of portable components between differing aviation architectures

The FACE approach allows software-based "capabilities" to be developed as components that are exposed to other software components through defined interfaces. It also provides for the reuse of software across different hardware computing environments. The FACE Technical Standard does not guarantee compliance with any safety certification standards.

Ultimately, the FACE key objectives are to reduce development, integration costs, and time-to-field avionics capabilities.

1.3 Scope of this Document

The Reference Implementation Guide for FACE Technical Standard, Edition 3.0 is provided as three volumes:

- Volume 1: General Guidance
- Volume 2: Computing Environment
- Volume 3: Data Architecture

This volume has been designed to:

- Support the FACE Technical Standard, Edition 3.0

  Note: In the event of conflict between the FACE Technical Standard and the FACE Reference Implementation Guide, the FACE Technical Standard takes precedence.

- Present best practices for designing and implementing Units of Conformance (UoCs) in the Portable Components Segment (PCS) and Platform-Specific Services Segment (PSSS) of the FACE Reference Architecture
- Guide the developer through example implementation scenarios where FACE UoCs are bundled together to provide typical airborne capabilities
- Provide further information on the use of Graphics Services within PCS and PSSS UoCs
- Provide guidance on addressing Safety and Security concerns during the planning and design phases of UoC development

Reference Implementation Guide for FACE Technical Standard, Edition 3.0, Volume 2: Computing Environment provides guidance and assistance for the following items:

- Developing UoCs in the Transport Services Segment (TSS), Input/Output Services Segment (IOSS), and Operating Services Segment (OSS) of the FACE Reference Architecture
- FACE OSS Profiles and Programming Language Run-times
- Injectable Interface and Life Cycle Management Services
- Programming Language Mapping Rules

Reference Implementation Guide for FACE Technical Standard, Edition 3.0, Volume 3: Data Architecture provides content to aid in understanding the overall approach the FACE Consortium has taken with respect to data management within the FACE Technical Standard. Central to the development and integration of FACE UoCs is the representation of data, according to the FACE Data Architecture.

2 Portable Components Segment (PCS)

2.1 Scope

This chapter provides guidance to Software Suppliers for implementing capabilities allocated to a PCS UoC as described in the FACE Technical Standard §3.10. The guidance includes suggested implementation approaches, design alternative considerations, and rationale for various aspects of the requirements of the FACE Technical Standard.

Section 2.2 addresses “backward compatibility” – changes in the FACE Technical Standard that would impact a PCS UoC that is conformant to a prior version in becoming conformant to the latest version. Section 2.3 describes key characteristics of a PCS UoC. Configurability for a PCS UoC is discussed in Section 2.4. Section 2.5 presents design alternatives. And finally Section 2.6 provides guidance for PCS UoCs in general.

2.2 Migrating a PCS UoC from Prior Versions

This section highlights changes in the FACE Technical Standard, Edition 3.0 that may be of interest to a Software Supplier of PCS UoCs aligned or conformant to prior versions of the FACE Technical Standard. The intent is to help the Software Supplier understand what would impact migrating the PCS UoC to conform to the FACE Technical Standard, Edition 3.0.

PCS UoC requirements were added related to new features:

- Life Cycle Management Services
- Component State Persistence
- Component Frameworks
- Security Transformations
- Injectable Interface

Readers are encouraged to learn more about these features in the FACE Technical Standard and corresponding sections of Volume 1 or Volume 2 of this document.

2.3 Key Characteristics

In the FACE Technical Standard, a key quality attribute is reusability through portability as the mechanism. The PCS exists in the FACE Reference Architecture to emphasize that portability focus. The interfaces and requirements allocated to UoCs in the other segments promote a separation of concerns that gives Software Suppliers and System Integrators more options to realize capabilities portably. A capability that is platform-independent, mission-oriented, or common services is a good candidate to be realized as a PCS UoC.

A PCS UoC is constrained by requirements in the FACE Technical Standard to encourage portability. A PCS UoC contains platform-independent business logic separated from platform implementations by interfaces in the FACE Technical Standard, as shown in Figure 1. The TS Interface is used for all communication to other UoCs. Any data sent over the TS Interface is represented in the PCS UoC UoP Supplied Model (USM).

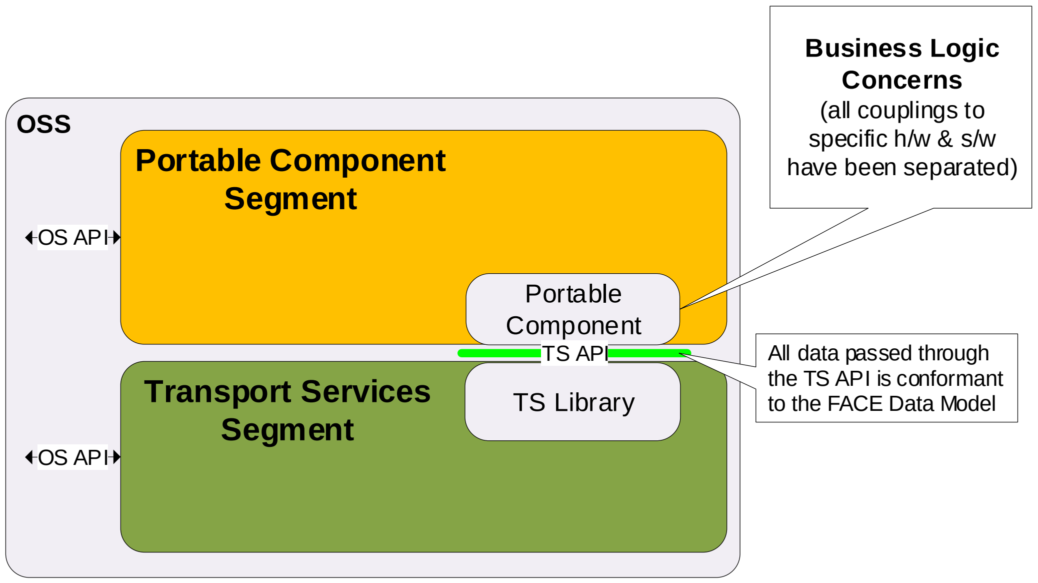

OSS
Portable Component Segment
OS API
Portable Component
TS API
Transport Services Segment
TS Library
OS API
Business Logic Concerns (all couplings to specific h/w & s/w have been separated)
All data passed through the TS API is conformant to the FACE Data Model

Figure 1: Portable Components

A PCS UoC is abstracted from peripheral I/O through the TS Interface. A PSSS UoC implements the appropriate peripheral I/O.

A PCS UoC uses Graphics Services to render displays. Graphics Services are accessed via the TS Interface. Refer to Chapter 5 for guidance on Graphics Services.

2.4 Configurability

It is recommended that a PCS UoC be configurable via Configuration Services. The relevant PCS UoC requirement is in the FACE Technical Standard §3.10.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when the PCS UoC can be configured in any of the following ways:

- Scalability and performance tuning of algorithms
- Initializing parameters
- Enabling/disabling behavior during program initialization

2.5 Design Alternatives

It is recommended that capabilities be implemented as a collection of loosely-coupled, tightly-cohesive PCS UoCs that perform collaboratively in order to encourage composability as a mechanism that supports both extensibility and maintainability quality attributes.

A PCS UoC design must address the direction of interface messages. TS Interface communications are bidirectional when the PSDS UoC is both a data producer and a data consumer.

A PCS UoC design may need to consider transactional message exchanges in a command/response communication pattern, meaning its logic may send a command message via the TS Interface and logically wait for a response before continued processing. Transactional messages frequently support timeout semantics that must also be addressed.

2.6 General PCS UoC Guidance

This section provides guidance that is applicable to all PCS UoCs.

2.6.1 Using Transport Services

A PCS UoC may be designed to use Transport Services via the TS Interface. The relevant PCS UoC requirements are in the FACE Technical Standard §3.10.1. Volume 2 of this document provides guidance for Transport Services, including its design goals and examples for using the TS Interface that are applicable to a PCS UoC.

2.6.2 Life Cycle Management Services

A PCS UoC may be designed to provide one or more of the Life Cycle Management (LCM) Services. The relevant PCS UoC requirements are in the FACE Technical Standard §3.10.1.2. Volume 2 of this document provides guidance for LCM Services, including its design goals and examples for providing LCM Services that are applicable to a PCS UoC.

2.6.3 Component State Persistence

A PCS UoC may be designed to use Component State Persistence (CSP) via the CSP Interface. The relevant PCS UoC requirement is in the FACE Technical Standard §3.10.1. Volume 2 of this document provides guidance for the CSP Interface, including its design goals and examples for providing the CSP Interface that are applicable to a PCS UoC.

2.6.4 Component Frameworks

There are three design alternatives for a PCS UoC to component frameworks. The relevant PCS UoC requirement is in the FACE Technical Standard §3.10.1.3.

A PCS UoC may be designed to leverage OSGi as a component framework via the OS Interface. Alternatively, it may be designed to leverage a component framework that is packaged within its UoC boundary. There is currently no guidance for these alternatives.

The third design alternative is to leverage the TSS Component Framework Capability via the TS Interface. Volume 2 of this document provides guidance for using the TSS Component Framework Capability.

2.6.5 Security Transformations

A PCS UoC may be designed to support Security Transformations. The relevant PCS UoC requirements are in the FACE Technical Standard §3.10.1.4. Chapter 6 provides guidance for Security Transformations.

2.6.6 Injectable Interface

A PCS UoC must provide the Injectable Interface for most FACE interfaces it uses:

- Each message exchanged over the TS Interface
- Configuration Services Interface
- Component State Persistence Interface

The relevant PCS UoC requirement is in the FACE Technical Standard §3.10.1. Volume 2 of this document provides guidance for the Injectable Interface, including its design goals and examples for providing the Injectable Interface that are applicable to a PCS UoC.

A PCS UoC does not provide the Injectable Interface for OS Interface operations defined by POSIX® or ARINC 653, or for the FACE HMFM Interface.

3 Platform-Specific Services Segment (PSSS)

3.1 Scope

This chapter provides guidance to Software Suppliers for implementing capabilities allocated to a PSSS UoC as described in the FACE Technical Standard §3.6. The guidance includes suggested implementation approaches, design alternative considerations, and rationale for various aspects of the requirements of the FACE Technical Standard.

Section 3.2 addresses “backward compatibility” – changes in the FACE Technical Standard that would impact a PSSS UoC that is conformant to a prior version in becoming conformant to the latest version. Section 3.3 provides guidance for PSSS UoCs in general, and subsequent sections provide guidance for a UoC aligned to one of the three PSSS sub-segments:

- Platform-Specific Device Services (PSDS UoC)
- Platform-Specific Common Services (PSCS UoC)
- Platform-Specific Graphics Services (PSGS UoC)

3.2 Migrating a PSSS UoC from Prior Versions

This section highlights changes in FACE Technical Standard, Edition 3.0 that may be of interest to a Software Supplier of PSSS UoCs aligned or conformant to prior versions of the FACE Technical Standard. The intent is to help the Software Supplier understand what would impact migrating the PSSS UoC to conform to the FACE Technical Standard, Edition 3.0.

PSSS UoC requirements were added related to new features:

- Life Cycle Management Services
- Component State Persistence
- Component Frameworks
- Security Transformations
- Injectable Interface

Readers are encouraged to learn more about these features in the FACE Technical Standard and corresponding sections of Volume 1 or Volume 2 of this document.

The I/O Services Interface is redesigned to eliminate the I/O Message Model (IOMM). The IOMM was originally conceived to help abstract PSSS UoC I/O processing from the I/O bus architecture and protocol. Based on feedback from Software Suppliers and System Integrators, the FACE Technical Standard now recognizes the significant inherent coupling between I/O processing logic and characteristics of the I/O bus architecture. FACE Technical Standard, Edition 3.0 replaces the IOMM solution with I/O Service declarations for each supported I/O bus architecture. All I/O bus architecture supported by the IOMM solution in prior versions of the FACE Technical Standard remains supported with the new I/O Service declarations. The new declarations also support more flexible configuration of I/O parameters at run-time and payload types tailored to the I/O bus architecture.

3.3 General PSSS UoC Guidance

This section provides guidance that is applicable to one or more of the PSSS sub-segments.

3.3.1 Using Transport Services

A PSSS UoC may be designed to use Transport Services via the TS Interface. The relevant PSSS UoC requirements are in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for Transport Services, including its design goals and examples for using the TS Interface that are applicable to a PSSS UoC.

3.3.2 Using I/O Services

A PSSS UoC may be designed to use I/O Services via the IOS Interface. The relevant PSSS UoC requirements are in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for I/O Services, including its design goals and examples for using the IOS Interface that are applicable to a PSSS UoC.

3.3.3 Life Cycle Management Services

A PSSS UoC may be designed to provide one or more of the Life Cycle Management (LCM) Services. The relevant PSSS UoC requirements are in the FACE Technical Standard §3.6.1.1. Volume 2 of this document provides guidance for LCM Services, including its design goals and examples for providing LCM Services that are applicable to a PSSS UoC.

3.3.4 Component State Persistence

A PSSS UoC may be designed to use Component State Persistence (CSP) via the CSP Interface. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for the CSP Interface, including its design goals and examples for providing the CSP Interface that are applicable to a PSSS UoC.

3.3.5 Component Frameworks

There are three design alternatives for a PSSS UoC to component frameworks. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1.

A PSSS UoC may be designed to leverage OSGi as a component framework via the OS Interface. Alternatively, it may be designed to leverage a component framework that is packaged within its UoC boundary. There is currently no guidance for these alternatives.

The third design alternative is to leverage the TSS Component Framework Capability via the TS Interface. Volume 2 of this document provides guidance for using the TSS Component Framework Capability.

3.3.6 Security Transformations

A PSSS UoC may be designed to support Security Transformations. The relevant PSSS UoC requirements are in the FACE Technical Standard §3.6.1.3. Chapter 6 of this document provides guidance for Security Transformations.

3.3.7 Injectable Interface

A PSSS UoC must provide the Injectable Interface for most FACE interfaces it uses:

- Each message exchanged over the TS Interface
- Each I/O bus architecture supported over the IOS Interface
- Configuration Services Interface
- Component State Persistence Interface

The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for the Injectable Interface, including its design goals and examples for providing the Injectable Interface that are applicable to a PSSS UoC.

A PSSS UoC does not provide the Injectable Interface for OS Interface operations defined by the POSIX standard or ARINC 653, or for the FACE HMFM Interface.

3.4 PSDS UoC Guidance

This section provides guidance that is applicable to a UoC in the PSDS sub-segment.

3.4.1 Key Characteristics

A PSDS UoC normalizes the interface between one or more peripherals (e.g., EGIs, GPS receivers, radar altimeters) and PCS UoCs or other PSSS UoCs that need to interact with those peripherals. The function of a PSDS UoC for a given peripheral is to insulate other UoCs from the effects of changes to the specific peripheral model used. For example, a system might originally be designed to incorporate Vendor A's EGI, but later it might prove advantageous to replace it with Vendor B's EGI. In this scenario, Vendor A's PSDS UoC would be replaced with a corresponding PSDS UoC from Vendor B, while other UoCs are isolated from the replacement.

A PSDS UoC relies on both the TS Interface and the IOS Interface to support this normalization. As an example, an EGI Manager PSDS UoC may communicate with a specific EGI device via the IOS Interface for a MIL-STD-1553 I/O bus architecture. Using the TS Interface, this EGI Manager could produce both EGI navigation data and EGI peripheral status while also consuming EGI peripheral commands. Messages exchanged via the TS Interface are represented in a FACE USM and form the basis of the system's contract for EGI interactions among UoCs. The EGI Manager may be upgraded to support a new version of the EGI peripheral or may be replaced to communicate with a new EGI peripheral that uses a different I/O bus architecture. The key quality attribute is system maintainability, specifically via replaceability.

A PSDS UoC may also support peripheral-specific functions such as initialization, configuration and status reporting, Built-In Test (BIT) management, state management, communications protocols, or message filtering. These functions are frequently highly coupled to the I/O bus architecture. For example, a GPS peripheral accessed via a serial interface could have a significantly different protocol compared to a GPS peripheral accessed via MIL-STD-1553. In this case, two GPS PSDS UoCs could implement the same TS Interface messages and encapsulate the logic differences between I/O bus architectures.

A PSDS UoC design must address the direction of interface messages. Typically, IOS Interface communications is bidirectional to support read/write operations, issue peripheral commands, and report peripheral status. TS Interface communications are bidirectional when the PSDS UoC is both a data producer and a data consumer.

A PSDS UoC design may need to consider transactional message exchanges. Using a GPS peripheral as an example, at the TS Interface the function to initiate BIT may be transactional, meaning receiving a message to initiate BIT is expected to be followed by a corresponding message reporting BIT status. That one transactional exchange at the TS Interface may involve many message exchanges over the IOS Interface. Transactional messages frequently support timeout semantics that must also be addressed.

3.4.2 Configurability

It is recommended that a PSDS UoC be configurable via Configuration Services. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when the PSDS UoC can be configured in any of the following ways:

- Supporting different models or versions of the peripheral type
- Supporting peripheral initialization parameters
- Supporting multiple instances of the same peripheral type

3.4.3 Design Alternatives

It is recommended that a PSDS UoC service a specific peripheral type (e.g., EGI or GPS).

A PSDS UoC design will be significantly influenced by the decisions supporting its requirements and the key characteristics described in Section 3.4.1. The design may consider using multiple threads to achieve performance requirements for potentially simultaneous communications at both the TS Interface and IOS Interface. Both the TS Interface and IOS Interface support synchronous communication using polling, with timeout support, and asynchronous communication using callbacks. Refer also to the general guidance for a PSSS UoC in Section 3.3.

3.5 PSCS UoC Guidance

This section provides guidance that is applicable to a UoC in the PSCS sub-segment. There are five capabilities that are described by the FACE Technical Standard as common services: Centralized Logging Services, Device Protocol Mediation, Streaming Media, Centralized Configuration Services, and System-level Health Monitoring. It is recommended that a Software Supplier focus a PSCS UoC on providing one of these capabilities to decouple unrelated capabilities and promote composable alternatives for the System Integrator.

3.5.1 Centralized Logging Services

This section documents guidance for a PSCS UoC that provides the Centralized Logging Service capability, receiving log messages from other UoCs and storing them in a consistent manner. No guidance is given for implementing localized logging within a UoC.

3.5.1.1 Key Characteristics

In the terminology of IETF RFC 5424: the Syslog Protocol, a Centralized Logging Service PSCS UoC could function as a relay or a collector. Other UoCs in the system function in the originator role.

A Centralized Logging Service PSCS UoC receives log messages via the TS Interface. It is recommended a domain-specific data model represent a log message in accordance with IETF RFC 5424 so multiple UoCs in the same system use the same message model.

The model of a log message should focus on the semantically distinct elements of a Syslog message, as described in IETF RFC 5424 §6. For example, modeling separate fields for Facility and Severity is more meaningful than encoding both values into a PRIVAL string representation. It is recommended that the model of a log message address the following semantically distinct elements: Facility, Severity, Timestamp, Host Name, Application Name, Process ID, Message ID, and Message. It is recommended that a variant log message model include Structured Data.

3.5.1.2 Configurability

It is recommended that a Centralized Logging Service PSCS UoC be configurable via Configuration Services. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when it can be configured in any of the following ways:

- Supporting both collector and relay roles as described in IETF RFC 5424
- Supporting both file system and peripheral-based storage in the collector role
- Supporting log message filtering
- Supporting multiple storage repositories in the collector role based on log message properties

3.5.1.3 Design Alternatives

A Centralized Logging Service PSCS UoC design will be significantly influenced by the decisions supporting its requirements and the key characteristics described in Section 3.5.1.1. The design may consider using multiple threads to achieve performance requirements for potentially simultaneous communications at both the TS Interface and IOS Interface. Both the TS Interface and IOS Interface support synchronous communication using polling, with timeout support, and asynchronous communication using callbacks. Refer also to the general guidance for a PSSS UoC in Section 3.3.

In the collector role described in IETF RFC 5424, a Centralized Logging Service PSCS UoC writes log messages to storage using either the OSS Interface to write to a file system or the IOS Interface to write to a peripheral. It is recommended that the log message be stored as a string according to IETF RFC 5424 to facilitate potential automated processing of Syslog messages.

In the relay role, it forwards received log messages to a collector using the IOS Interface for inter-partition communication. The FACE Technical Standard requires that a log message sent for relay via the IOS Interface be encoded as a string according to IETF RFC 5424. This allows the System Integrator to select a Syslog collector implementation independent of the relay implementation.

The decision to support the POSIX standard or ARINC 653 OS interfaces is independent of the key characteristics of a Centralized Logging Service PSCS UoC. The TS Interface allows it to receive log messages from other UoCs regardless of their operating system support. Both operating systems provide file system support for the collector role. A distributed dispatch IO/Service deployment can be used by the System Integrator when a Centralized Logging Service PSCS UoC performs the relay role to a collector in another partition.

A Centralized Logging Service PSCS UoC may be used to implement a system's security log functions. In this case, the UoC may be required to adhere to the Security OSS Profile. Otherwise, the selection of OSS Profile should be determined by the System Integrator based on system deployment criteria or by the Software Supplier based on portability and reuse considerations.

3.5.2 Device Protocol Mediation

This section documents guidance for a PSCS UoC that provides the Device Protocol Mediation (DPM) capability, translating I/O messages between an IOSS UoC and a PSDS UoC.

3.5.2.1 Key Characteristics

The DPM capability is described in the FACE Technical Standard to address the potential need to translate I/O communication between a PSDS UoC and an IOSS UoC deployed in the system by the System Integrator. It provides the opportunity to reuse a PSDS UoC in systems with alternative I/O peripherals. Protocol mediation is a common use case for the capability.

A DPM PSCS UoC is noteworthy because it provides the IOS Interface for communication with a PSDS UoC. It is the only case in the FACE Technical Standard where an inter-segment interface is provided specifically for intra-segment communication. For the PSDS UoC to be reused without modification, the DPM PSCS UoC must implement the behaviors associated with providing an I/O Service.

A DPM PSCS UoC is intended to bridge a portable, reusable PSDS UoC for deployment in varied system configurations. That intent may be realized by the System Integrator as a point solution without a portability goal. A Software Supplier may find it advantageous to implement the UoC to provide protocol mediation as a reusable widget.

3.5.2.2 Configurability

It is recommended that a DPM PSCS UoC be configurable via Configuration Services. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when it can be configured in any of the following ways:

- Supporting multiple protocol mediation options
- Supporting appropriate parameters for each protocol mediation option

3.5.2.3 *Design Alternatives*

A DPM PSCS UoC design will be significantly influenced by the decisions supporting its requirements and the key characteristics described in Section 3.5.2.1. The design may consider using multiple threads to achieve performance requirements for potentially simultaneous communications at the IOS Interface to both the PSDS UoC and the IOSS UoC. The IOS Interface supports synchronous communication using polling, with timeout support, and asynchronous communication using callbacks. Refer also to the general guidance for a PSSS UoC in Section 3.3.

The decision to support the POSIX standard or ARINC 653 OS interfaces is dependent upon the OSS Profile of the PSDS UoC and IOSS UoC pair it mediates. The IOS Interface is not an inter-partition interface, meaning the three UoCs will be deployed to the same OS partition.

The selection of OSS Profile should be determined by the System Integrator based on system deployment criteria or by the Software Supplier based on portability and reuse considerations.

3.5.3 Streaming Media

This section documents guidance for a PSCS UoC that provides the Streaming Media capability, providing access to streaming media formats via the TS Interface.

3.5.3.1 *Key Characteristics*

A Streaming Media PSCS UoC sends streaming media payloads via the TS Interface. It is recommended a domain-specific data model represent a streaming media payload message so multiple UoCs in the same system use the same message model. The model of a streaming media payload message is not required to conform to a streaming media standard ICD.

A Streaming Media PSCS UoC is noteworthy because it can directly access media drivers in order to achieve potential low latency, high bandwidth requirements. The FACE Reference Architecture recognizes that this performance-sensitive capability merits direct access rather than using the IOS Interface to reach the media drivers via an I/O Service.

3.5.3.2 *Configurability*

It is recommended that a Streaming Media PSCS UoC be configurable via Configuration Services. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when it can be configured in any of the following ways:

- Supporting multiple streaming media options
- Supporting appropriate parameters for each streaming media option
- Supporting multiple stream sources for each media streaming option

3.5.3.3 *Design Alternatives*

A Streaming Media PSCS UoC design will be significantly influenced by the decisions supporting its requirements and the key characteristics described in Section 3.5.3.1. The design may consider using multiple threads to achieve performance requirements for potentially simultaneous communications at both the TS Interface and media driver interface. The TS Interface supports synchronous communication using polling, with timeout support, and asynchronous communication using callbacks. Refer also to the general guidance for a PSSS UoC in Section 3.3.

A Streaming Media PSCS UoC may be designed to support media device status and control or media stream control via the TS Interface. The corresponding messages would be added to the UoC's USM.

Both the decision to support the POSIX standard or ARINC 653 OS interfaces and the selection of OSS Profile should be determined by the System Integrator based on system deployment criteria or by the Software Supplier based on portability and reuse considerations.

3.5.4 Centralized Configuration Services

It is recommended that Centralized Configuration Services not be used in a system design. The feature of dependency injection via the Injectable Interface, new in FACE Technical Standard, Edition 3.0, allows the System Integrator to assign a realization of Configuration Services to a UoC that transparently implements the functions for centralized management. Volume 2 of this document provides guidance for providing Configuration Services.

The remainder of this section documents guidance for a PSCS UoC that provides the Centralized Configuration Services capability, providing configuration data from a repository via the TS Interface. The concept of localized configuration is addressed in the FACE Technical Standard by Configuration Services.

3.5.4.1 Key Characteristics

The Centralized Configuration Services capability is described in prior versions of the FACE Technical Standard to provide a System Integrator an alternative design to centrally manage configuration for multiple UoCs in the system, as opposed to maintaining localized configuration for each UoC.

A Centralized Configuration Services PSCS UoC publishes configuration data to each supported UoC via the TS Interface. It must therefore be implemented with knowledge of the expected configuration messages for each UoC, as well as any potential time-dependent ordering. A UoC using Configuration Services may internally query its configuration data in order.

When a UoC expects to receive its configuration data from a Centralized Configuration Services PSCS UoC, it must define its configuration data as messages in its USM. This may be perceived to make the configuration data more publicly visible. A UoC using Configuration Services documents its configuration data for the System Integrator, which may be perceived to be less publicly visible than a USM.

3.5.4.2 Configurability

It is recommended that a Centralized Configuration Services PSCS UoC itself be configurable via Configuration Services. The relevant PSSS UoC requirement is in the FACE Technical Standard §3.6.1. Volume 2 of this document provides guidance for using Configuration Services. Portability and reusability are enhanced when it can be configured in any of the following ways:

- Supporting multiple repository backend storage mechanisms
- Supporting multiple TS Interface connections to publish configuration data

- Supporting multiple configuration data message types associated to each TS Interface connection

3.5.4.3 *Design Alternatives*

A Centralized Configuration Services PSCS UoC design will be significantly influenced by the decisions supporting its requirements and the key characteristics described in Section 3.5.4.1. The design may consider using multiple threads to achieve performance requirements for potentially simultaneous communications at the TS Interface. The TS Interface supports synchronous communication using polling, with timeout support, and asynchronous communication using callbacks. Refer also to the general guidance for a PSSS UoC in Section 3.3.

Both the decision to support the POSIX standard or ARINC 653 OS interfaces and the selection of OSS Profile should be determined by the System Integrator based on system deployment criteria or by the Software Supplier based on portability and reuse considerations.

3.5.5 System-Level Health Monitoring

No guidance is provided for a PSCS UoC that provides the System-level Health Monitoring capability. Refer to Volume 2 of this document for a detailed description of health monitoring and fault management from an OS perspective. The concept of a Computing Platform Health Manager is conceptually analogous to a System-level Health Monitoring PSCS UoC.

3.6 PSGS UoC Guidance

A PSGS UoC provides graphics services to PCS UoCs and other PSSS UoCs. Graphics Services are described extensively in Chapter 5. Refer also to the general guidance for a PSSS UoC in Section 3.3.

4 FACE Profile Guidance

4.1 Introduction

The FACE Reference Architecture supports any combination of general-purpose, safety-critical, or secure avionics capabilities through the use of profiles. Profiles affect the interfaces and functionalities available within an implementation of the FACE Reference Architecture. This chapter guides you through the strategies of profile selection for a given development or migration. This chapter will provide insight for:

- Migration of a FACE UoC designed for one profile to another profile
- Migration of a legacy component to conform to a FACE Profile
- Conversion from non-timed-based to timed-based partitioning

4.2 Strategy

It is recommended that a profile selection strategy align with the profile needs of a given product's short and long-term roadmaps. Component developers should select their profile based on an understanding of the value of portability between profiles; not doing so may cause difficulty in migrating between profiles in the future. Using APIs associated with more restrictive FACE Profiles facilitates ease of migrating to less restrictive FACE Profiles.

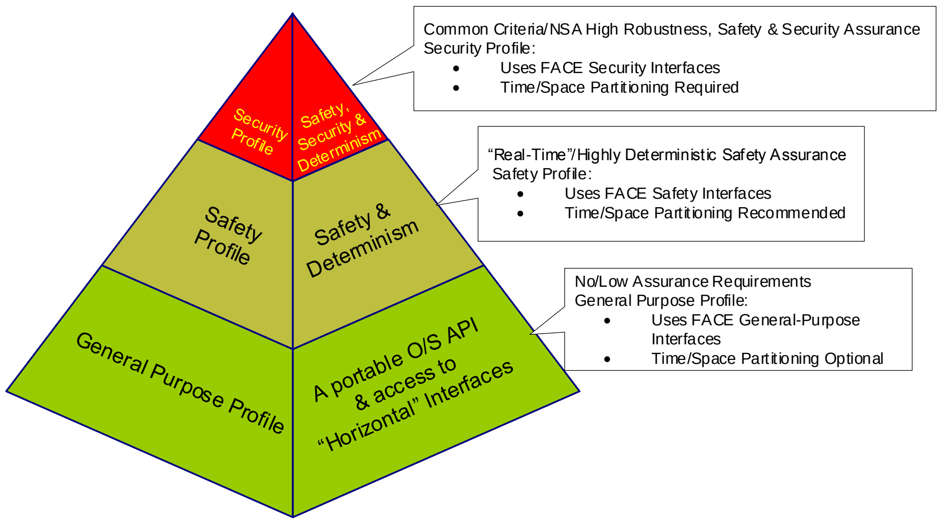

Security Profile
Safety, Security & Determinism
Safety Profile
Safety & Determinism
General Purpose Profile
A portable O/S API & access to "Horizontal" Interfaces
Common Criteria/NSA High Robustness, Safety & Security Assurance
Security Profile:
• Uses FACE Security Interfaces
• Time/Space Partitioning Required
"Real-Time"/Highly Deterministic Safety Assurance
Safety Profile:
• Uses FACE Safety Interfaces
• Time/Space Partitioning Recommended
No/Low Assurance Requirements
General Purpose Profile:
• Uses FACE General-Purpose Interfaces
• Time/Space Partitioning Optional

Figure 2: FACE Profile Diagram

An example would be using the Safety Profile APIs to meet requirements of a General Purpose Profile component to gain value in portability for a product planned to be used in the Safety Profile for the future based on its product roadmap.

4.3 Migration Between FACE Profiles

Component developers should clearly indicate to which FACE Profile(s) the component is designed to apply, as well as key characteristics. At a minimum, this should include safety-related information relative to the development assurance-level categories contained in safety-related documents such as Radio Technical Commission for Aeronautics (RTCA) DO-178B/C, RTCA DO-254, and ARP 4754A, as well as security-related information relative to security assurance levels contained in documents such as ISO/IEC 15408:2009.

FACE Profiles are defined as a set of permissible APIs. More restrictive FACE Profiles are subsets of less restrictive FACE Profiles. ARINC 653 is optional in the General Purpose Profile. ARINC 653 and the POSIX standard are required in the Safety and Security Profiles. A component is portable from a more restrictive to a less restrictive profile, with the exception of moving from ARINC 653 to a POSIX standard-only general-purpose OS, because support for ARINC 653 APIs is optional in the General Purpose Profile.

Other key characteristics are helpful to include for purposes of migrating between FACE Profiles. For software, this may include things such as program memory needed, dynamic memory used, execution time on a given platform, etc. These characteristics may be further broken down to include various use cases.

5 Graphics Services

5.1 Purpose

The purpose of this chapter is to provide additional implementation guidance for Graphics Services providers.

5.2 Introduction

Graphics Services are a unique concept when considered in the context of the FACE Technical Standard. Graphics Services, in and of themselves, are not defined as a specific segment within the FACE Technical Standard, but as a contributor to multiple segments within the FACE Reference Architecture. Graphical software components contribute to many segments within an implementation of the FACE Reference Architecture. This chapter describes, in detail, how Graphics Services contribute to the overall FACE Reference Architecture.

FACE Graphics Services are based on the Model View Controller (MVC) architecture design pattern, dividing the "business" logic (the Model) away from the Graphics Services (the View) that are responsible for rendering the image. The Controller manages data input, state modifications, and commands changes to Model and View components.

Graphics Services draw heavily on existing commercial and avionics standards to provide a fully featured graphical suite. The graphical suite of software components is intended to cover the capabilities that can be used to create Human Machine Interfaces (HMI) including: 2D Rendering, 3D Rendering, Text Rendering, Context Creation, Windowing Capabilities, and Distributed Graphics Rendering. To enable the graphical capabilities within the FACE Reference Architecture, the following standards have been selected:

- OpenGL/EGL
- ARINC 739
- ARINC 661

This document assumes the reader has a general understanding of graphics applications and rendering techniques. It is written to provide guidance on how to use the selected standards within the FACE Reference Architecture. Certain topics have been added to provide clarifications for existing system compatibility to an implementation of the FACE Reference Architecture.

5.2.1 Summary of Changes from Edition 2.1

OpenGL/EGL:

- Added new OpenGL SC 2.0
- Updated EGL to 1.4, and added the EGL_EXT_compositor extension

- Restructured OpenGL/EGL versions applicable to OS profiles

ARINC 661:

- Added requirements for User Application (UA) portability
- Added requirements for display management and windowing control
- Added minimum required widget profile
- Updated to ARINC 661-5

ARINC 739:

- No significant changes, recommend for move to I/O Segment

5.2.2 How to Read this Section

The information contained in this portion of the Reference Implementation Guide should be used in developing PCS and PSSS UoCs, and by integrators when developing the integration configuration of FACE UoCs.

5.2.3 Graphics Portability Considerations

Unlike many other applications, FACE UoCs that produce graphics execute under restrictions that force architectural decisions:

- There are physical restrictions on the display area available (number, size, resolution, and aspect ratio of the physical display surface)
- There are performance restrictions on the applications (many graphical applications are computationally intensive, require hardware co-processor support to achieve adequate performance, and, in certain circumstances, can place a user in danger if adequate performance is not achieved)

The FACE Technical Standard's software portability objectives suggest that new UoCs can be combined together without change, or that new UoCs can be added to an existing system without significant integration effort. However, when the UoCs produce graphics, they have to share the physical display real estate that is available. In almost all circumstances, when combining graphical applications some level of integration effort should be expected. The choices made in the creation of the FACE Technical Standard attempt to minimize this integration effort.

5.2.3.1 ARINC 661 Considerations

The main decision supporting the portability of graphical applications was that to adopt ARINC 661 as a mandatory standard when more than one graphical application is to be used in a system. ARINC 661 was identified as the only commonly available graphical standard that isolates (to an extent) the layout of graphics created by one UoC from that created by another, while also being able to be used in DO-178 certifiable systems. While the General Purpose Profile is far more permissive, restrictions mandated in the Safety and Security Profiles are intended to support and promote portability for UoCs between these systems.

ARINC 661 divides an avionics system into two main components: the Cockpit Display System (CDS), responsible for display of graphical information on-screen, and one or more UAs that implement the logic of the avionics system. Communication between these components is standardized using the ARINC 661 protocol. The CDS normally has no "intelligence" about the avionics system that it is implementing, and must be driven by a protocol stream to render graphics on the screen; the UA likewise has no means of displaying graphics other than sending protocol to a "listening" CDS, and receiving protocol representing crew interaction.

Another aspect of portability of ARINC 661 components results from the lack of consistent styling enforced by the standard. ARINC 661 clearly allows such styling as "yellow, wide, dashed non-flashing line", but unfortunately the style settings are implemented in different ways by different CDS vendors. This capability enables a widget set to be created which can be used in implementations deployed to multiple customers or aircraft without the need to modify the internals of the widget. However, a UA attempting to render a "yellow, wide, dashed non-flashing line" may, unless modified, inadvertently render blue boxes surrounded by yellow dashed flashing lines, whereas a second customer may want a red box surrounded by no lines, flashing or not. This change of look can be accomplished without changing any of the widget code since the rendering style is a characteristic of the CDS.

The final component which enables portability of graphical applications in an implementation of the FACE Reference Architecture is the Display Manager, which is a required component of Graphics Display Management Services. This component is a UA which is responsible for the final layout of the display during operation. It controls the visibility, size, and location of the graphical display of the connected UAs, and the visibility, size, and location of the External Source Widgets which are used to render OpenGL application graphics (whether the FACE EGL Compositor is deployed or not) and other video or sensor sources. The Display Manager is the control application that ensures non-critical information does not obscure important system warnings or flight-critical data. It may be used to implement the main user interface control system by which the user selects options that control the display, or it may receive information from such a system and cause the various components to behave correctly by controlling UA graphics visibility.

The following lists some elements which must be considered when making a portable graphical application.

- Aspect ratio of application
- Screen size (pixels, PPI)
- Background colors/transparency/haloing
- Use of common colors/configurable color tables (style definitions in 661_comformance.xsd)
- Fonts/text/images (loading/saving from file system/lack thereof)
- Use of OpenGL extensions
- Use of ARINC 661 OEM/custom widgets

5.2.3.2 *OpenGL Considerations*

With a full windowing system, OpenGL rendering can be constrained within a screen region. However, embedded platforms can normally not provide such systems, particularly if certification needs to be taken into account. Resulting from this, the second major decision to support portability of graphical applications was to create the FACE EGL Compositor extension.

This extension allows well-behaving OpenGL applications to render into what each application "thinks" is a complete framebuffer, but is in fact an off-screen buffer that will subsequently be composited with other off-screen buffers from other OpenGL applications to form the final screen display visible to a user. With minimal integration effort, this EGL extension permits an OpenGL application to share the physical screen.

5.2.4 Relationship to the FACE Reference Architecture

Within the PSSS a Graphics Services UoC may access proprietary graphical rendering APIs. The idea is that the PSSS UoC has specific requirements that are platform-specific due to the specific hardware the platform uses. Such graphical requirements may include things such as: the resolution of each display being used, the dot pitch of the screens, the refresh rate of each screen, and similar properties of the graphical hardware. Some Graphics Processing Units (GPUs) may require special initialization or configuration which require GPU-specific APIs. It is the intent of the PSSS graphical services to fulfill the requirements which are platform-specific, which in many cases concerns the rendering hardware as well as the displays to which rendering occurs.

Graphical applications which do not fulfill graphical platform-specific requirements should not be placed in the PSSS Graphical Services and should be built to be reused in different environments. This includes limiting the use of rendering APIs to those allowed for the PCS as well as taking into account many of the items discussed in Section 5.2.3.

OpenGL and EGL in the FACE Reference Architecture reside as part of the OSS API and as such the APIs may be used by both the PCS and PSSS. While the OSS APIs are allowed to be used in all FACE Segments, the use of OpenGL and EGL within the TSS, IOSS, and OSS is not considered to provide reasonable use of these APIs, because these segments should not have any need to render data to a visual output.

Graphical applications rely on the TSS in the following mandatory ways:

- All ARINC 661 communications which flow between any UA and CDS must use the TSS – these communication channels should be tagged as ARINC 661 data streams
- All ARINC 739 communications which flow from client to server must use the TSS – these communication channels should be tagged as ARINC 739 data streams
- All graphical client server data streams such as GLX must flow through the TSS and be represented in a USM

The use of the TSS within graphical applications promotes portability and reusability of graphical applications. The render APIs described here are designed to be asynchronous and separate such that the minimal amount of communications is needed between the applications. For example, in a compositing system the compositor does not need to control when a composited application renders its frame or exactly when the swap happens. However, when non-rendered graphical data does need to flow between graphical applications the applications must use the TSS. This includes information like pointer [cursor] position, selection, and other similar data which is sometimes considered graphical in nature but is normally not.

5.3 Graphics Services

5.3.1 ARINC 661

5.3.1.1 Introduction

ARINC 661 defines an overall architecture with many sub-components governing the creation of display systems. It normalizes the definition of a display server and UA in a Model View Controller (MVC) design pattern. The standard defines the use of a CDS for dynamic graphical image generation, and the communication protocol between the CDS and UAs which manages aircraft avionics functions.

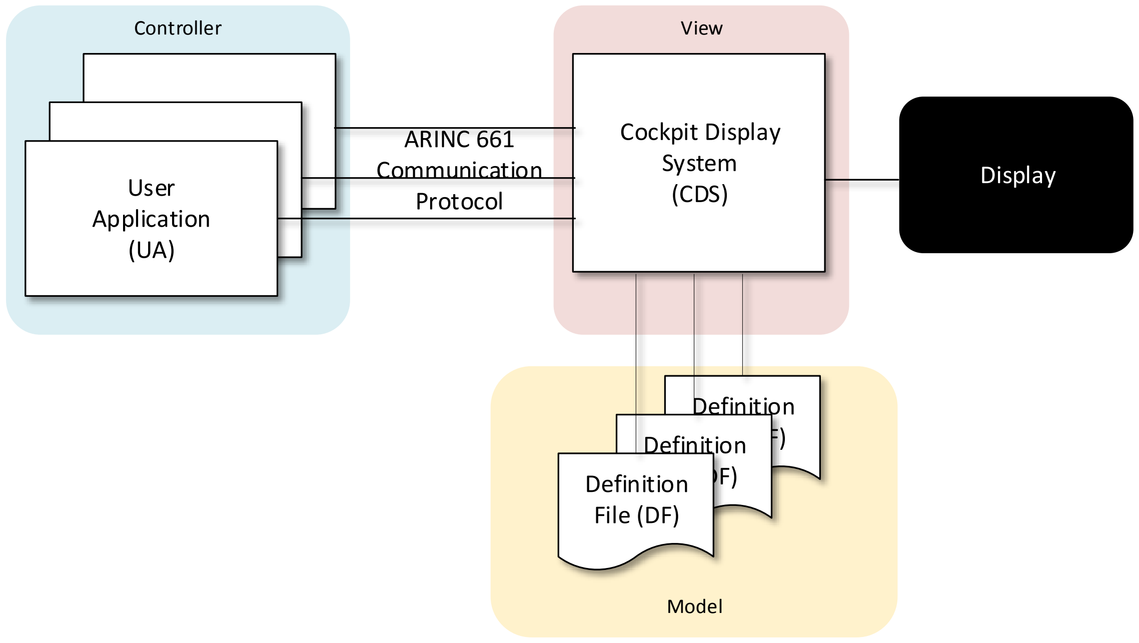


Figure 3: ARINC 661 MVC Overview

5.3.1.2 Technical Description

The MVC design pattern in ARINC 661 separates processing between the information generation sources and the component that draws graphics to a display. The standard gives guidance for how display screens are defined, a protocol to direct control of the display, as well as how to receive user events from a display and pass events back to the controller. Separating these processing elements supports changes in display appearance, such as color or line width. ARINC 661 provides the capability to separate the data entry, display, and computational aspects of a system. The UA can reside in an entirely different CPU to ease processor loading.

The ARINC 661 CDS is responsible for displaying graphical images to the end user using a library of software graphical elements defined by the ARINC 661 Specification. The ARINC 661 Specification defines these elements as widgets. Each widget has attributes describing the mechanisms that direct the appearance and dynamic movement of the widget. At start up, the display elements (widgets) are loaded based on one or more definition files depicting windowing configuration, layering behavior, and initial properties such as position, color, and visibility.

UAs are the control mechanism that drives the CDS via the ARINC 661 communication protocol. This protocol allows UAs to direct dynamic changes to elements of the display by the control of individual attributes associated with specified widgets. Multiple UAs can communicate with the server directing and/or controlling different display components of the displayed screen through run-time attribute changes.

Unlike OpenGL, ARINC 661 is a graphical definition of user-defined displays using the widget components defined in the standard. The ARINC 661 CDS provides a standardized interface for displaying imagery that can be implemented on top of OpenGL as the drawing mechanism. The CDS is not limited to OpenGL for this function.

Widget organization is defined in Definition Files (DFs). DFs are either human-readable XML or binary representations of the creation attributes for widgets on the display. The UA specifies, through the use of run-time modifiable attributes in the DF, characteristics of the instances of each widget to modify when communicating to the CDS. ARINC 661 provides a data schema for the XML DF definition.

The ARINC 661 Specification provides mechanisms whereby OEMs can create new widgets. The OEM widgets are implementation-specific and are therefore not portable to different standardized implementations of the CDS. As such, OEM-specific or extended widgets are not to be included.

5.3.1.3 *Portability Considerations*

ARINC 661 defines a UA that must know certain parameters of the CDS. The Reference Implementation Guide for FACE Technical Standard, Edition 2.1 provided guidance for using common indexes and definitions in addition to suggesting that the CDS could allow for a method to configure the undefined ARINC 661 parameters. In the FACE Technical Standard, Edition 3.0, instead of suggesting that this change be made in some CDS-specific way it defines an XML Schema Definition (XSD) for how a CDS should allow these parameters to be set. This section builds on the work of the Reference Implementation Guide for FACE Technical Standard, Edition 2.1 §7.1.3.1 to explain the parameterization.

The FACE Technical Standard §H.2 defines the XSD used to define the schema for the ARINC 661 UA to communicate the color, font, and style attributes from the UA developer to the CDS. Every UA can provide a UA-specific style configuration file for that UA. Connecting the style configuration file to the UA's DF in the CDS is shown in the example in Section 5.4.3.4 by using the *styleFilePath* attribute of the *ua* element. When a UA does not provide a style definition file the CDS would use its default styles.

There is an ARINC 661 element defined as *colorTableEntry*, which is used to create color index to RGBA mappings for the color indices that ARINC 661 provides. *colorTableEntrys* elements are grouped into a *colorTable* element to express the color index to RGBA mappings for all color indices used by the UA. The *pictureTableEntry* allows for mapping picture reference to file paths such that the CDS can know where to find a specific image without having to load the image into the DF. The table entries, defined in the XSD, allow each index, which the UA uses, to be mapped to specific settings that the UA expects it to be.

5.3.2 OpenGL

5.3.2.1 *Introduction*

OpenGL is a widely adopted group of specifications defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. This document assumes the reader has working knowledge of OpenGL. Readers unfamiliar with OpenGL may consult the OpenGL Reference Manuals, or specifications provided by the Khronos Group at https://www.khronos.org/.

5.3.2.2 *Technical Description*

OpenGL applications may be implemented as either PCS or PSSS components, or both. In the PCS, the EGL interface to the display hardware is abstracted via the EGL_EXT_compositor extension. This abstraction allows an OpenGL application to function within the PCS as though it had control of the platform's graphics services through the EGL interfaces.

A PSSS OpenGL application may access the EGL interfaces directly to create and manage platform graphics services. When the PSSS supports PCS OpenGL applications, it takes the role of a graphics services compositor, managing rendering of the PCS OpenGL application(s), in addition to its own rendering.

OpenGL profiles within the FACE General Purpose Profile are restricted to OpenGL SC 1.0.1, OpenGL SC 2.0, or OpenGL ES 2.0. The OpenGL Safety-Critical profiles (SC 1.0.1 and SC 2.0) are intended for use within the FACE Safety Base and Safety Extended Profiles.

Use of OpenGL within the FACE Security Profile to present secure data poses a potential security concern due to the ability of the *glReadPixels*() API to retrieve data from a display surface. Furthermore, display buffers used to represent the display surface(s) may reside in user-addressable memory, thereby allowing for access to the contents. OpenGL/EGL driver use of Dynamic Memory Access (DMA) buffers, and GPU non-volatile memory may pose further security risks. Caution is therefore recommended when attempting to use OpenGL/EGL within the Security Profile.

5.3.2.3 *Use Cases*

5.3.2.3.1 *OpenGL PCS Example*

Figure 4 shows the most basic example of how OpenGL and EGL will be used in a FACE UoC. The UoC developer will create an application that uses OpenGL and EGL calls directly. Since OpenGL and EGL are assigned to the OSS, the application is not required to use the TS Interface for transmission of graphics commands. The UoC is not required to have any understanding of compositing in the system because of the abstraction provided by the EGL_EXT_compositor extension. The compositing within the system will be occurring during end system integration.

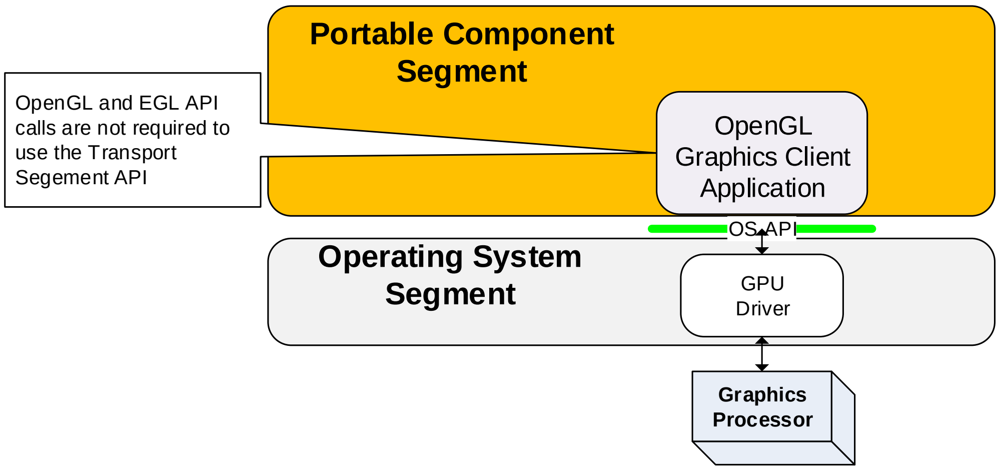

Portable Component Segment
OpenGL and EGL API calls are not required to use the Transport Segement API
OpenGL Graphics Client Application
OS API
Operating System Segment
GPU Driver
Graphics Processor

Figure 4: PCS Graphics Services Example

5.3.2.4 OpenGL PSSS Example

Figure 5 shows three different examples of OpenGL UoC deployment within the PSSS. The primary difference in the figure pertains to how the rendered context is managed. In Example 1, the OpenGL context is created using EGL. In Example 2, the OpenGL context will be created using a graphics framework or toolkit such as Qt or X Windows. In Example 3, the OpenGL context will be provided by an ARINC 661 external source widget.

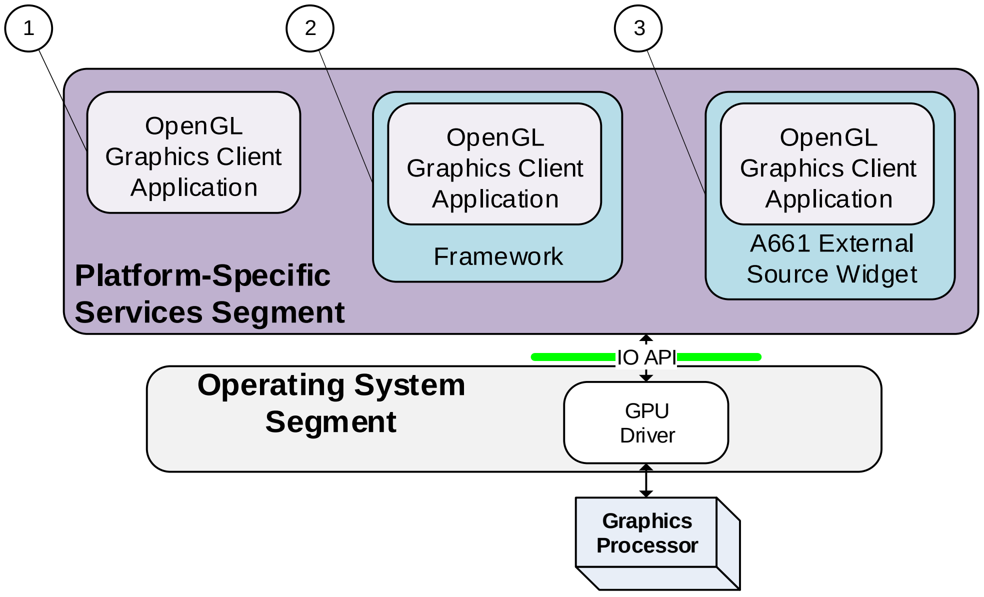

1
2
3
OpenGL Graphics Client Application
OpenGL Graphics Client Application
Framework
OpenGL Graphics Client Application
A661 External Source Widget
Platform-Specific Services Segment
IO API
Operating System Segment
GPU Driver
Graphics Processor

Figure 5: PSSS Graphics Services Example

In Figure 5, the larger blue boxes surrounding the OpenGL Graphics Client Applications in Examples 2 and 3 are abstraction frameworks for graphics management. This means that the UoC will interact with framework-level calls rather than EGL and or OpenGL directly.

5.3.2.5 *Safety-Critical OpenGL Profiles*

The EGL API, *eglCreateContext*(), allows a client application to specify the OpenGL major and minor versions, or to use the ES profile. On embedded targets, this capability typically goes unused, because the OpenGL driver profile is singularly defined for the target platform. OpenGL client applications, in this case, would have knowledge of the available OpenGL profile. Desktop OpenGL implementations, however, typically support a wide variety of OpenGL profiles, thus requiring the OpenGL client application to specify the profile during initialization. The EGL specification defines EGL_OPENGL_ES_BIT, OPENGL_ES2_BIT to enable the ES profiles with a major and minor version, but provides no facility to select an SC profile; as EGL_OPENGL_SC_BIT and EGL_OPENGL_SC2_BIT are undefined. There is no formal method to specify an SC profile with EGL. Fortunately, the SC 1.0.1 and SC 2.0 profiles are subsets of OpenGL 1.3 and OpenGL ES 2.0, respectively. It is recommended that OpenGL client applications specify one of those two those profiles to activate the corresponding OpenGL SC profile.

5.3.2.6 *OpenGL Extensions*

The OpenGL language provides a mechanism, through the *glGetString*() API, to query the OpenGL implementation for any platform-supported extensions. If the platform supports dynamic binding of APIs, a PSSS OpenGL client application may use OpenGL extensions at run-time. PCS OpenGL client applications are not permitted to use OpenGL extensions and remain conformant.

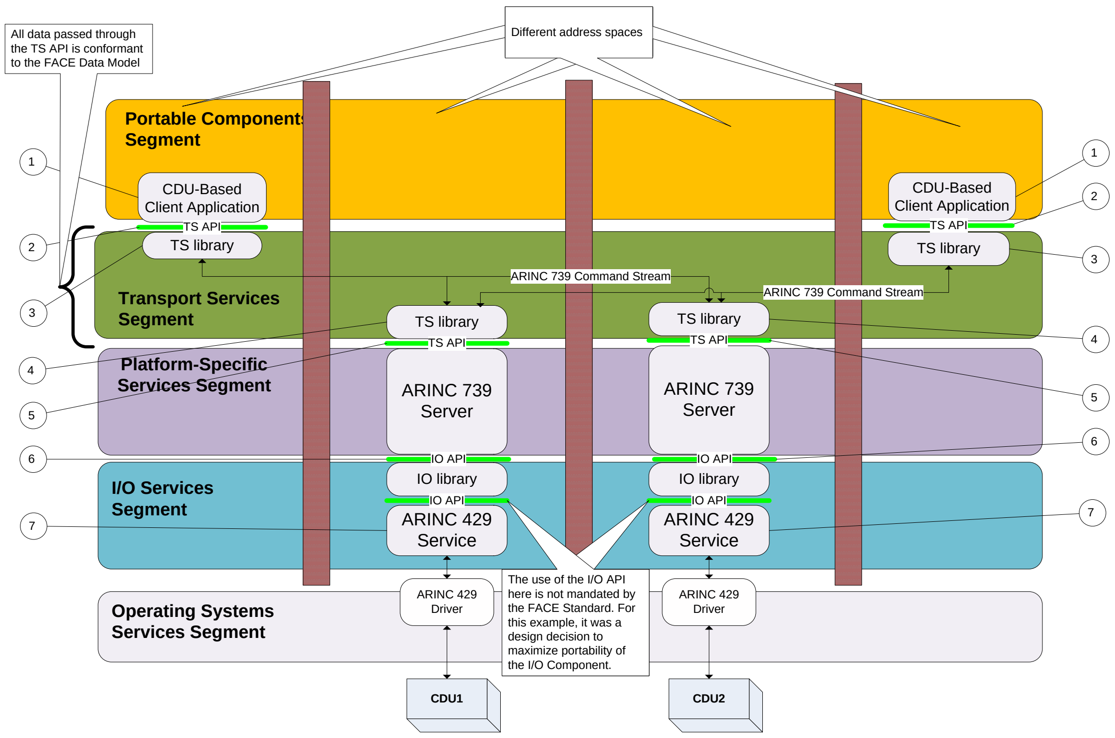
5.3.3 ARINC 739

5.3.3.1 *Introduction*

No change was made to ARINC 739 within the FACE Technical Standard between Editions 2.1 and 3.0.

5.3.3.2 *Technical Description*

Figure 6 shows an example data flow in an ARINC 739 distributed environment. This example shows two Control Display Unit (CDU)-based client applications communicating with two ARINC 739 servers.



All data passed through the TS API is conformant to the FACE Data Model
Different address spaces
Portable Components Segment
CDU-Based Client Application
TS API
TS library
Transport Services Segment
ARINC 739 Command Stream
Platform-Specific Services Segment
ARINC 739 Server
IO API
IO library
I/O Services Segment
ARINC 429 Service
The use of the I/O API here is not mandated by the FACE Standard. For this example, it was a design decision to maximize portability of the I/O Component.
ARINC 429 Driver
Operating Systems Services Segment
CDU1
CDU2
1
2
3
4
5
6
7

Figure 6: ARINC 739 Server Distributed Graphics Communication

The list below describes the sequence of actions performed to send ARINC 739 data across the TS Interface to multiple CDUs:

1. CDU-based client application packs the ARINC 739 command stream data using the FACE Data Architecture (FACE Technical Standard §3.6).
2. CDU-based client application provides packed data to TS library by calling the TS API *Send_Message* function (FACE Technical Standard §E.8).
3. TS library executes the *Send_Message* function using IPC to send the ARINC 739 command stream.
4. TS library receives the ARINC 739 command stream for the ARINC 739 server.
5. TS library handles the callback registered by the ARINC 739 server if required.
6. ARINC 739 servers send the ARINC 739 data by using the ARINC 429 I/O Service (see the Reference Implementation Guide for FACE Technical Standard, Edition 3.0, Volume 2: Computing Environment §6.4 for more information on I/O communication).
7. ARINC 429 service communicates with the non-standardized ARINC 429 drivers to control the CDUs.

5.4 Graphics Display Management Services

5.4.1 Introduction

Display Management Services were created due to the lack of a common windowing system that could be used across the entire range of platforms, operating systems, GPUs, and window managers which exist in the avionics ecosystem. To standardize window management an extension to EGL was created. This allows one or more applications to set up off-screen buffers into which PCS UoCs can draw using only the OpenGL and EGL standard API. This differs from most windowing system integrations of EGL because the client applications no longer need to rely on API calls to the windowing systems such as: XOpenDisplay on X11, CreateWindowEx on the Windows system, or ANativeWindow_setBuffersGeometry on Android™. These are just a subset of the platform-specific APIs which must be called to create on-screen windows using three standard operating system APIs.

To allow PCS UoCs to be conformant and fully portable, the calls to the platform-specific windowing APIs needed to be removed from the application. As such, EGL had to be extended to allow creation of off-screen windows which could then be composited onto a screen using standard EGL and OpenGL function calls. The EGL_EXT_compositor extension was created to allow one or more PSSS UoCs to specify graphical system resource usage, which is specific to the platform on which the applications are deployed.

Since Display Management Services use an ARINC 661 API for control of the windowed surface using the ARINC 661 External Source Widget, the ARINC 661 CDS is responsible for implementation of the Display Management Services through the use of the EGL_EXT_compositor extension provided by the EGL driver. The ARINC 661 CDS uses a display management configuration item which uses the schema described by the XSD defined in the FACE Technical Standard §H.3. Using this configuration item, addition of new graphical applications requires minimal changes to the existing software. It also allows for portability of the existing system to newer hardware which may have other inputs or properties of those inputs which may only require simple configuration changes as opposed to rebuilding the entire system from the original components.

5.4.2 Technical Description

Without the addition of the EGL_EXT_compositor extension PCS UoCs must be able to make calls to platform-specific windowing API calls in order to get on-screen space. This causes an issue with portability because those APIs then tie the applications to a specific windowing system which also normally includes tying the application to a specific operating system, which means it is not portable.

With the addition of the Display Management Services, a common API was created to provide display organization applications to be created to manage the organization of the data being rendered to any given screen.

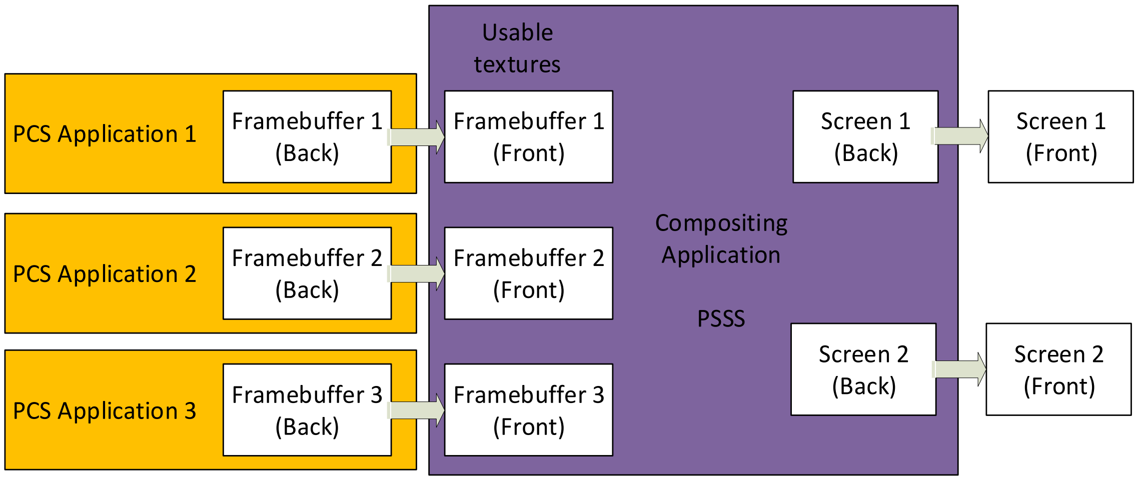


Figure 7: Compositing Architecture

Figure 7 provides a visual representation of how graphical data flows in a system using the EGL_EXT_compositor. Here, three PCS UoCs each render to framebuffers using their own rendering context. When each application is done drawing, it instructs the EGL driver to make its framebuffer data visible to the compositing application as usable textures using the standard eglSwapBuffers call. As usable textures, the compositing application then renders those textures onto the screens according to the Display Management Services requirements of stacking, visibility, and sizing.

In some environments, the framebuffers' data might be copied to the textures backing memory, and others may swap the front and back buffer pointers to achieve the requirements. In either case, the framebuffer data in the front buffer is unaffected by the rendering operations of the back buffer until the PCS UoC notifies the EGL driver that it is done rendering and would like the buffers swapped. This sort of rendering allows the PCS UoCs to render at any frame rate and the compositing application to render at any frame rate it needs to. For example, if PCS UoC 1 only gets new data at 5Hz, and PCS UoC 2 gets new data at 18Hz, PCS UoC 3 renders at 30Hz, and the compositing application is also compositing a 60Hz video stream, then the compositor can run at 60Hz while allowing the other applications to run slower and not overuse rendering resources to render the same data over and over again.

When using an ARINC 661 CDS as a compositing application, as discussed here, the CDS should also be optimized to be able to render its ARINC 661 widgets at the rate they need to be rendered while allowing things like video streams and other external video widgets to be rendered at a faster rate. Without this optimization, the amount of symbology the ARINC 661 CDS must draw could have a direct negative performance impact on the rendering resources of the system.

5.4.3 Usage Examples

5.4.3.1 Generalizing EGL Context Creation in the PCS

This use case describes how to set up an EGL Rendering context and window surfaces for use with the FACE PCS. Any OpenGL application using EGL can either own the GPU, or will be used with the FACE Display Management Services. This results in portability between platforms, as well as the ability to share the screen with other Graphics Services UoCs, when applicable. When the OpenGL/EGL driver supports the EGL_EXT_compositor extension, the window identifier used in eglCreateWindowSurface is an integer read from a configuration item for this UoC. The window size is defined by the Display Management Service. The application only has to create an EGLSurface for that window. The Display Management Service is responsible for defining the number of EGLContext and reference IDs of each EGLContext, which an application can use. The EGL_EXT_compositor extension requires that surfaces and context be matched such that a specific window identifier used in eglCreateWindowSurface matches the same EGL_EXTERNAL_REF_ID_EXT specified for eglCreateContext.

The windows for this platform are set up through a UoC that resides as part of the PSSS Graphics Services. The PCS UoC must query the window size from EGL using eglQuerySurface requesting EGL_HEIGHT and EGL_WIDTH, after the surface has been created. A PCS Graphics Services UoC should be written to adjust rendering based on the given window size. A PCS Graphics Services UoC should also assume that the window size may be changed without explicit notification of the PCS Graphics Services UoC. Therefore, the PCS Graphics Services UoC should query the window size each cycle before rendering, as the surface size may have changed.

Below is a high-level pseudo-code example of the PCS code outline.

```
EGLint dpyId, win, contexId, numCfgs=1, width, height
FACE_BUFFER_SIZE_TYPE bytes
FACE_RETURN_CODE_TYPE ret
EGLDisplay dpy
EGLContext ctx
EGLSurface sfc
EGLConfig cfg
EGLBoolean errorStatus
EGLint contex_attribs[] = {EGL_EXTERNAL_REF_ID_EXT, 0, EGL_NONE}
EGLint config_attribs[] = {EGL_BUFFER_SIZE, 32, EGL_DEPTH_SIZE, 24,
EGL_NONE}

FACE_Configuration_Read(conf_handle, "DisplayId", &dpyId,
sizeof(EGLint), &bytes, &ret)
FACE_Configuration_Read(conf_handle, "WindowId", &win, sizeof(EGLint),
&bytes, &ret)
FACE_Configuration_Read(conf_handle, "ContexId", &contextId,
sizeof(EGLint), &bytes, &ret)

contex_attribs[1] = contextId

dpy = eglGetDisplay(dpyId)
eglInitialize (dpy, 0, 0)
eglChooseConfig(dpy, config_attribs, &cfg, 1, &numCfgs)
sfc = eglCreateWindowSurface(dpy, cfg, win, NULL)
ctx = eglCreateContext(dpy, cfg, EGL_NO_CONTEXT, context_attribs)
/* Must wait for surface and context to be created,
    PSSS UoC may not have setup windows yet */
errorStatus = eglMakeCurrent(dpy, sfc, sfc, ctx)

while(...)
    eglQuerySurface(dpy, sfc, EGL_HEIGHT, &height)
    eglQuerySurface(dpy, sfc, EGL_WIDTH, &width)

    resize(width, height)
    draw()
    eglSwapBuffers(dpy, sfc)
```

It is possible for a PCS Graphics Services UoC to use EGL without the EGL_EXT_compositor, but it must rely on static values for the win parameter, width, height, display id, and context within the EGL Driver/Library. Then, a Software Supplier can claim it as a PCS UoC. Otherwise, the need to call nativeCreateWindow causes it to be a PSSS UoC.

5.4.3.2 *Generalizing EGL Context Creation in the PSSS*

EGL context creation within the PSSS utilizes the platform-specific EGL window creation API to initialize the "win" parameter (EGLNativeWindowType) that is passed to eglCreateWindowSurface.

Below is a high-level X11 pseudo-code outline used to initialize EGL.

```
EGLint dpyId, contexId, numCfgs=1, width, height
FACE_BUFFER_SIZE_TYPE bytes
FACE_RETURN_CODE_TYPE ret
EGLDisplay dpy
EGLContext ctx
EGLSurface sfc
EGLint contex_attribs[] = {EGL_EXTERNAL_REF_ID_EXT, 0, EGL_NONE}
EGLint config_attribs[] = {EGL_BUFFER_SIZE, 32, EGL_DEPTH_SIZE, 24,
EGL_NONE}

/** Start X11 Native Display Initialization */

Window root
XSetWindowAttributes swa
XSetWindowAttributes  xattr
Atom wm_state
Atom fullscreen
XWMHints hints
XEvent xev
EGLConfig ecfg
EGLint num_config
Window win
Display x_display

x_display = XOpenDisplay(NULL)
if ( x_display == NULL )
    return EGL_FALSE

root = DefaultRootWindow(x_display)

swa.event_mask  =  ExposureMask | PointerMotionMask | KeyPressMask
win = XCreateWindow (
    x_display, root,
    0, 0, width, height, 0,
    CopyFromParent, InputOutput,
    CopyFromParent, CWEventMask,
    &swa )

// Make the window visible on the screen
XMapWindow (x_display, win)
XStoreName (x_display, win, title)

// Get identifiers for the provided atom name strings
wm_state = XInternAtom (x_display, "_NET_WM_STATE", FALSE)
```

```
memset ( &xev, 0, sizeof(xev) )
xev.type                 = ClientMessage
xev.xclient.window       = win
xev.xclient.message_type = wm_state
xev.xclient.format       = 32
xev.xclient.data.l[0]    = 1
xev.xclient.data.l[1]    = FALSE
XSendEvent (
    x_display,
    DefaultRootWindow ( x_display ),
    FALSE,
    SubstructureNotifyMask,
    &xev )

/** End X11 Native Display Initialization */

FACE_Configuration_Read(conf_handle, "DisplayId", &dpyId,
sizeof(EGLint), &bytes, &ret)
FACE_Configuration_Read(conf_handle, "ContexId", &contextId,
sizeof(EGLint), &bytes, &ret)

contex_attribs[1] = contextId

dpy = eglGetDisplay(dpyId)
eglInitialize (dpy, 0, 0)
eglChooseConfig(dpy, config_attribs, &cfg, 1, &numCfgs)
sfc = eglCreateWindowSurface(dpy, cfg, win, NULL)
ctx = eglCreateContext(dpy, cfg, EGL_NO_CONTEXT, context_attribs)
errorStatus = eglMakeCurrent(dpy, sfc, sfc, ctx)

while(...)
    eglQuerySurface(dpy, sfc, EGL_HEIGHT, &height)
    eglQuerySurface(dpy, sfc, EGL_WIDTH, &width)

    resize(width, height)
    draw()
    eglSwapBuffers(dpy, sfc)
```

5.4.3.3 *PSSS Graphics Services UoC Window Setup*

When supporting PCS Graphics Services UoCs, a PSSS Graphics Services UoC is required to set up EGL windows. This is done using the functions listed in the EGL_EXT_compositor extension. The Graphics Services UoC is specific to the display surfaces being rendered to, as well as the other Graphical Services UoCs in the system. As such, it can only be a PSSS UoC and may require the use of proprietary GPU APIs during setup. This compositing PSSS UoC should be started before any other Graphics Services UoCs which use OpenGL, because PCS Graphics Services UoCs will have to wait until the PSSS UoC sets up the windows for the PCS Graphics Services UoCs.

The example below shows a high-level framework for specifying four windows which are shared between three different EGLContexts. This allows the PCS Graphics Services UoCs to use the specified contexts and windows.

```
EGLint dpyId, win, contextId, numCfgs=1, width, height
FACE_BUFFER_SIZE_TYPE bytes
```

```
FACE_RETURN_CODE_TYPE ret
EGLDisplay dpy
EGLContext ctx
EGLSurface sfc
EGLConfig cfg
EGLint contex_attribs[] = {EGL_PRIMARY_COMPOSITOR_CONTEXT_EXT,
EGL_NONE}
EGLint config_attribs[] = {EGL_BUFFER_SIZE, 32, EGL_DEPTH_SIZE, 24,
EGL_NONE}

FACE_Configuration_Read(conf_handle, "DisplayId", &dpyId,
sizeof(EGLint), &bytes, &ret)

EGLint contextIds[NUM_CONTEXTS] = {5,2,3}
EGLint windowIds[NUM_CONTEXTS][NUM_WINDOWS] = {2,3,4,5}
EGLint win1Attribs[] = {EGL_WIDTH, 768, EGL_HEIGHT, 512,
                        EGL_VERTICAL_RESOLUTION, 20,
                        EGL_HORIZONRAL_RESOLUTION, 20, EGL_NONE}
EGLint win2Attribs[] = {EGL_WIDTH, 384, EGL_HEIGHT, 512,
                        EGL_VERTICAL_RESOLUTION, 20,
                        EGL_HORIZONRAL_RESOLUTION, 20, EGL_NONE}
GLuint windows[NUM_WINDOWS]

dpy = eglGetDisplay(dpyId)
eglInitialize (dpy, 0, 0)

eglChooseConfig(dpy, config_attribs, &cfg, 1, &numCfgs)
win = nativeCreateWindow(DISPLAY_HEIGHT, DISPLAY_WIDTH) // Platform
specific API
sfc = eglCreateWindowSurface(dpy, cfg, win, NULL)
ctx = eglCreateContext(dpy, cfg, EGL_NO_CONTEXT, context_attribs )
errorStatus = eglMakeCurrent(dpy, sfc, sfc, ctx)

eglCompositorSetContextListEXT(contextIds, NUM_CONTEXTS)
eglCompositorSetWindowListEXT(contextIds[0], &windowIds[0], 1)
eglCompositorSetWindowListEXT(contextIds[1], &windowIds[1], 1)
eglCompositorSetWindowListEXT(contextIds[2], &windowIds[2], 2)

eglCompositorSetWindowAttributesEXT(windowIds[0], win1Attribs, 8)
eglCompositorSetWindowAttributesEXT(windowIds[1], win1Attribs, 8)
eglCompositorSetWindowAttributesEXT(windowIds[2], win2Attribs, 8)
eglCompositorSetWindowAttributesEXT(windowIds[3], win2Attribs, 8)

eglCompositorSwapPolicyEXT(windowIds[0],
EGL_COMPOSITOR_KEEP_NEWEST_FRAME_EXT)
eglCompositorSwapPolicyEXT(windowIds[1],
EGL_COMPOSITOR_KEEP_NEWEST_FRAME_EXT)
eglCompositorSwapPolicyEXT(windowIds[2],
EGL_COMPOSITOR_DROP_NEWEST_FRAME_EXT)
eglCompositorSwapPolicyEXT(windowIds[3],
EGL_COMPOSITOR_DROP_NEWEST_FRAME_EXT)

glGenTexture(4, windows)

for(win = 0; win < NUM_WINDOWS ; ++win)
    glBindTexture(windows[win], GL_TEXTURE_2D)
    eglCompositorBindTexWindowEXT(windowIds[win])
```

```
while(...)
    drawWindows()

    eglSwapBuffers(dpy, sfc)
```

5.4.3.4 *Multiple Contributing Application Window Management*

This section describes a situation where multiple graphical applications are required to draw on the same screen, where each application is a UoP. This situation includes symbology overlay on top of external video.

The applications being used in this example are:

- An ARINC 661 server with window management capabilities
- An ARINC 708 weather radar rendering application built using OpenGL/EGL
- A digital map using OpenGL/EGL
- A map application for flight planning and situational awareness OpenGL/EGL
- An ARINC 661 UA providing control of map external sources
- An ARINC 661 UA providing primary flight guidance
- An ARINC 661 UA providing camera GUI
- An ARINC 661 UA providing display management

The external video sources being used are:

- A (Pan tilt zoom) PTZ forward mounted camera
- A fixed mounted downward looking camera

All applications listed above will be displayed on a 768 x 1024 pixel display in portrait orientation. The display area will be split into two logical areas: the primary flight instruments and the mission sensor area. The mission sensor area will be 768 x 768 pixels with the primary flight area being 768 x 256 pixels. Figure 8 shows two layouts of the display for this example.

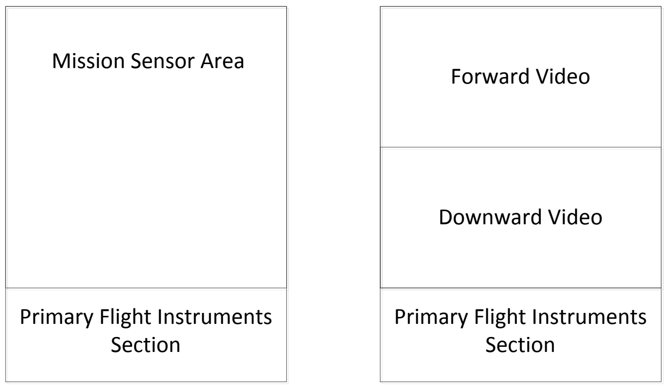

Mission Sensor Area
Primary Flight Instruments Section
Forward Video
Downward Video
Primary Flight Instruments Section

Figure 8: Multiple Graphical UoCs Screen Spaces

Below is an XML configuration file which describes the organization of the window areas of the first display layout shown in Figure 8, defines which DF files to load, and defines the external video sources being used.

External video sources 2 – 4 are references to the OpenGL applications 2 – 4 listed in the application list above. External source 0 and 1 are video cameras. The four UAs share three DF files with the display manager. The display manager is aware of all the layer numbers to control layer visibility of the other three UAs.

```
<configuration>
  <screen id="0">
    <pixelSize width="768" height="1024" />
    <physicalDimensions width="6" height="8" units="inch" />
    <layout id="0" >
       <window id="0" name="PrimaryFlightWindow" >
         <description>
            This window is used for the primary flight instruments
section of the display
         </description>
         <pixelArea x="0" y="0" width="768" height="256" />
         <scalingFactor xScale="19.84" yScale="19.84" baseUnit="screen"
perUnit="pixel" />
       </window>

       <window id="1" name="MissionSensorWindow" >
         <description>
            This window is used for digital map and video displays
         </description>
         <pixelArea x="0" y="256" width="768" height="256" />
         <scalingFactor xScale="20.0" yScale="20.0" baseUnit="screen"
perUnit="pixel" />
       </window>
    </layout>
  </screen>

/<!-- The following window ids reference the windows selected in the
above layouts, and are used to group ARINC 661 DF files within windows
to allow layering or other ARINC 661 interactions between UAs -->
  <ua applicationId="5" window="1" visible="true"
dFPath="/661dfs/map.bin" styleFilePath="/661dfs/map.conf.bin" />
  <ua applicationId="6" window="0" visible="true"
dFPath="/661dfs/pfd.bin" styleFilePath="/661dfs/pfd.conf.bin" />
  <ua applicationId="7" window="1" visible="false"
dFPath="/661dfs/camera.bin" />
  <ua applicationId="8" window="1" visible="false"
dFPath="/661dfs/displayManager.bin" />

<!-- The following ids reference the external source ID parameter for
ARINC 661, see FACE TS 3.0 H.3.7 -->
  <externalSource id="0" name="forward camera" type="SMPTE292" >
     <properties name="input" value="1" />
     <properties name="width" value="1024" />
     <properties name="height" value="768" />
     <properties name="units" value="pixels" />
  </externalSource>
  <externalSource id="1" name="downward camera" type="RS170" >
    <properties name="path" value="/dev/video1" />
  </externalSource>
  <externalSource id="2" name="weather_radar"   type="OpenGL" />
  <externalSource id="3" name="map_application" type="OpenGL" />
  <externalSource id="4" name="map_overlays"    type="OpenGL" />

</configuration>
```

The above XML defines windows using the definition of a window defined in ARINC 661 §2.3. The use of the external source widget can be used to control windowing in a less static method. The use of the external source widget method is used to control the relative content of the mission sensor window.

In the Mission Sensor Area window, the map and camera definition files are loaded with the map being visible by default, and the camera being not visible on load. The display management UA is responsible for changing which layers in the DF files are visible at any specific time.

The Map DF file contains three external source widgets. External source widget 1 will be the map background with a source reference of 3. External source widget 2 has an external source reference of 2, so it will be the weather radar external source widget. External source widget 3 is the map overlays with a source reference of 1.

ARINC 661 defines widget and object stacking through the order they are listed in the DF. This puts the map background in the back, the weather radar next, and the symbology overlays in front of the weather radar and map background.

By using the size and positioning attributes of the external source widget, the weather radar can be moved and zoomed to fit the map properly. Using similar map projections for both OpenGL applications, the symbology is sized to allow for proper placement of the overlay.

A key to making this work is that a background color of the OpenGL applications is known by the ARINC 661 server, or uses a transparent background. The weather radar and symbology overlay applications use a transparent background. A basic assumption for this example is that these applications are drawn using the same GPU and the backgrounds are fully transparent.

When the user wants a video displayed instead of a map, the display management UA makes the map layer invisible and the camera layer visible. The camera layer is made up of multiple basic containers of external source widgets. The first is an external source filling the entire area, with an external source reference of 0 to display the forward-looking camera in the whole space. The second is an external source widget filling the entire area with an external source reference of 1. This allows the downward looking camera to fill the entire screen space. The third has two external source widgets. The positioning and sizing is shown in the right side of Figure 3. The camera control UA controls which camera layout is active at any specific time allowing the user to change as requested. The camera control UA could allow zooming of a specific external video.

The primary flight instruments window provides full isolation of the Primary Flight Display (PFD) functionality and because it uses ARINC 661 as its draw language, it requires no additional explanation. If the PFD is an OpenGL application, a single external source widget can be used to place the OpenGL application output in the window area.

5.5 Graphics Requirements Decomposition

It is important to understand that the FACE Technical Standard does not levy requirements on a platform. The FACE Technical Standard only levies requirements on UoCs. When developing a platform that implements the FACE Reference Architecture, it must be understood that the platform should be developed to host the broadest set of capabilities expected to be needed by the UoCs. While a FACE Computing Environment is not required to support any specific graphics services, UoCs may be expecting their availability. Software Suppliers should consider the roadmap of any product and provide the largest set of FACE Graphics Services expected to be required. The commentary below is intended to provide the FACE developer with the intention of the authors as they developed the FACE Technical Standard and how best to stay within the "Spirit of the FACE Technical Standard".

Note: The following subsections feature tables containing requirements, as written, from FACE Technical Standard, Edition 3.0. All references to chapters or sections in the "Requirement Text" column apply to those found in the FACE Technical Standard and not this Reference Implementation Guide.

5.5.1 OSS Requirements for Graphics Services

Table 1 shows the OSS requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 1: OSS Requirements for Graphics Services

Edition 3.0 Section	Requirement Text
5.5.1.1	When supporting OpenGL, an OSS UoC shall provide OpenGL drivers compatible with OpenGL SC 1.0.1 or OpenGL SC 2.0, or OpenGL ES 2.0.
5.5.1.2	When supporting EGL, an OSS UoC shall provide EGL drivers compatible with EGL, Version 1.4 or EGL, Version 1.4 with the EGL_EXT_compositor extension.
5.5.1.3	When supporting Graphics Display Management Services, an OSS UoC shall provide an EGL driver with the EGL_EXT_compositor extension. Note: An OpenGL or EGL driver can be provided as part of the OSS. OpenGL and EGL drivers expected to support Graphics UoCs in the PCS or PSSS will need to be able to support OpenGL SC 1.0.1 or OpenGL SC 2.0, or OpenGL ES 2.0 or EGL, Version 1.4, or EGL, Version 1.4 with EGL_EXT_compositor extensions.

5.5.1.1

When a platform has an OpenGL requirement, this does not mean that the platform cannot support a superset of the selected OpenGL version. It just means that a Software Supplier must, at a minimum, choose a version that has all of the calls in the selected conformant version of OpenGL. Platforms without safety-critical or security requirements would be advised to choose OpenGL ES 2.0. If a Software Supplier does have safety-critical or security requirements, they should choose one of the safety-critical versions of OpenGL. OpenGL is not a requirement for a platform. OpenGL and EGL are OSS APIs that can be accessed from either PCS or PSSS. If a

Software Supplier chooses to use another API, they are limited to only deploy to the PSSS or to represent the data in a USM it if deployed to the PCS.

5.5.1.2

UoCs that use OpenGL are required to use EGL as their context management API. The EGL_EXT_compositor extension is not required by the platform but it should be used in all systems that intend to deploy more than one UoC to share the screen or support OpenGL applications in the PCS. All modern display systems should use this extension. Legacy OpenGL applications will require minor modifications to support the compositor extension.

There is a known issue in that no version of the EGL specification supports OpenGL SC as a client API. During context creation the client API is typically specified in *eglBindAPI*(). This poses an issue for cross-platform development where multiple OpenGL APIs are present. The recommendation is to choose the version of the API that acts as a superset of the calls.

5.5.1.3

The EGL_EXT_compositor extension is not required by the platform but it should be used in all systems that intend to deploy more than one UoC to share the screen or support OpenGL applications in the PCS. All modern display systems should use this extension. Legacy OpenGL applications will require minor modifications to support the compositor extension. It remains within the spirit of the FACE Technical Standard to use the compositor extension without a CDS being deployed. The EGL_EXT_compositor extension used with a CDS provides a naturally partitioned graphics approach that eases the certification burden later. Using the EGL_EXT_compositor without a CDS is a better approach than integrating a bunch of tightly coupled OpenGL applications that require knowledge of one another.

5.5.2 PCS Requirements for Graphics Services

5.5.2.1 *PCS User Application Requirements*

Table 2 shows the PCS User Application Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 2: PCS User Application Requirements for Graphics Services

Section	**Requirement Text**
5.5.2.1.1	A Graphics Services UoC implementing ARINC 661 UAs in the PCS shall satisfy the requirements in Section 3.10.
5.5.2.1.2	A Graphics Services UoC implementing an ARINC 661 UA shall use the TS Interface to communicate ARINC 661 data.
5.5.2.1.3	A Graphics Services UoC implementing an ARINC 661 UA shall document the set of ARINC 661 widgets it uses.
5.5.2.1.4	A Graphics Services UoC implementing an ARINC 661 UA shall have an associated ARINC 661 DF.

5.5.2.1.1

This requirement requires that ARINC 661 UAs that are deployed to the PCS conform to the PCS requirements.

5.5.2.1.2

This is a clarifying requirement to specify that ARINC 661 UA must communicate to the ARINC 661 CDS via the TSS interface; this was not clear in FACE Technical Standard, Edition 2.1. It should be known that the ARINC 661 interface is not required to be data modeled per J.2.4.20.

5.5.2.1.3

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases an unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms. The UA can utilize widgets beyond this list but will need to document this list so a CDS's capability can be matched to the UAs need.

5.5.2.1.4

Each individual UA will require an associated Binary Definition File that defines the layout of the HMI. The final composed HMI will be made up of a collection of various BDF files controlled by a Display Manager which will be developed by the end System Integrator. Structure of the BDF is defined in the ARINC 661 Specification and the FACE Technical Standard does not modify this in any way.

5.5.2.2 PCS Cockpit Display System Requirements

Table 3 shows the PCS Cockpit Display System Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 3: PCS Cockpit Display System Requirements for Graphics Services

Section	**Requirement Text**
5.5.2.2.1	The ARINC 661 CDS UoC shall provide OpenGL windowing using the external source widget when configured to support Graphics Display Management Services.
5.5.2.2.2	A Graphics Services UoC implementing an ARINC 661 CDS in the PCS shall support the logic specified in ARINC 661-5 for widgets it provides.
5.5.2.2.3	A Graphics Services UoC implementing an ARINC 661 CDS in the PCS shall provide the minimum widget subset as specified in Table 10. Note: The ARINC 661 CDS may provide additional widgets.
5.5.2.2.4	A Graphics Services UoC implementing an ARINC 661 CDS in the PCS shall satisfy the requirements in Section 3.10.

Section	Requirement Text
5.5.2.2.5	A Graphics Services UoC implementing an ARINC 661 CDS shall use the TS Interface to communicate ARINC 661 data.
5.5.2.2.6	A Graphics Services UoC implementing an ARINC 661 CDS shall use the XSD defined in Section H.2 for its style data configuration.
5.5.2.2.7	A Graphics Services UoC implementing an ARINC 661 CDS shall use the XSD defined in Section H.3 for its display management configuration parameters.
5.5.2.2.8	A Graphics Services UoC implementing an ARINC 661 CDS shall document the set of ARINC 661 widgets it provides.

It is the intent that a CDS is a PSSS service but it is possible when using OpenGL or a data modeled graphics interface that a CDS can exist in the PCS. This set of requirements is to govern that use, but it is recommended that the CDS be defined as a PSSS UoC.

5.5.2.2.1

The intent of this requirement is to tell the CDS implementer that the OpenGL render context should be presented and managed inside of the CDS via the external source widget. This way a scene can be fully composed within the confines of the CDS for control of visibility, positioning, and size. This is so that legacy OpenGL, video, and sensor data can be displayed simultaneously. The purpose of the external source widget within ARINC 661 is to allow the display and control of non-ARINC 661 render sources. This is the key widget that allows for true portability of Graphical UoCs from system to system.

5.5.2.2.2

This requirement says that a Software Supplier needs to implement the widgets per the ARINC 661-5 Specification.

5.5.2.2.3

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms, but it is expected that all CDS implementations will tailor a superset of widgets for the specific platform capabilities. This being the case, CDS providers should provide a documented list of widgets for their platform-specific CDS.

5.5.2.2.4

This requirement requires that an ARINC 661 CDS that is deployed to the PCS conforms to the PCS requirements.

5.5.2.2.5

This is a clarifying requirement to specify that a ARINC 661 CDS must communicate to the ARINC 661 UAs via the TSS interface; this was not clear in FACE Technical Standard, Edition

2.1. It should be known that the ARINC 661 interface is not required to be data modeled per J.2.4.20.

5.5.2.2.6

The ARINC 661 Specification does not provide a means to configure the reference handles for things like lines styles, colors, etc. This XSD is our means to provide the configuration required for a consistent representation. For example, a UA can be assured that the handle for a given color is the same on all platforms.

5.5.2.2.7

The ARINC 661 Specification does not provide a means to configure physical attributes of the hardware. This XSD gives users of the FACE Technical Standard a standardized way to configure these elements. This file would best be described as the Master Integrator’s file.

5.5.2.2.8

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms, but it is expected that all CDS implementations will tailor a superset of widgets for the specific platform capabilities. This being the case, CDS providers should provide a documented list of widgets for their platform-specific CDS.

5.5.2.3 ARINC 739A Requirements

Table 4 shows the PCS ARINC 739A Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 4: PCS ARINC 739A Requirements for Graphics Services

Section	**Requirement Text**
5.5.2.3.1	A Graphics Services UoC shall use ARINC 739A messages defined in Section 3.7 of ARINC 739-1 or ARINC 739A-1 when communicating with the ARINC 739A Services.
5.5.2.3.2	ARINC 739 Client UoCs deployed to the PCS shall satisfy the requirements in Section 3.10.

5.5.2.3.1

This requirement is stating that when using ARINC 739 there is a defined message set that needs to be used for it to work properly. This message set is specified in an externally referenced standard.

5.5.2.3.2

This requirement requires that an ARINC 739 client that is deployed to the PCS conforms to the PCS requirements.

5.5.2.4 *OpenGL Requirements*

Table 5 shows the PCS OpenGL Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 5: PCS OpenGL Requirements for Graphics Services

Section	**Requirement Text**
5.5.2.4.1	A Graphics Services UoC using OpenGL shall also use EGL, Version 1.4. Note: A Graphics Services UoC may only use extended EGL as specified in the OSS graphics requirements.
5.5.2.4.2	A Graphics Services UoC using OpenGL in the: a. General Purpose Profile shall use OpenGL SC 1.0.1, OpenGL SC 2.0, or OpenGL ES 2.0. b. Safety Profile shall use OpenGL SC 1.0.1 or OpenGL SC 2.0. c. Security Profile shall use OpenGL SC 1.0.1 or OpenGL SC 2.0. Note: Graphics Services UoC may not use extended OpenGL and be conformant.
5.5.2.4.3	A Graphics Services UoC shall be restricted to the core profile for the OpenGL version being used. Note: This does not preclude use of dynamic binding to use OpenGL extensions, however the UoC must not rely on the existence of any OpenGL extensions.
5.5.2.4.4	A Graphics Services UoC using *eglGetDisplay* shall use a configurable parameter for the *display_id* input argument. Note: The configurable parameter, for example, could be read from a file or passed in at startup of the software component. See Configuration Services in Section 3.2.5.
5.5.2.4.5	A Graphics Services UoC using *eglCreateWindowSurface* shall use a configurable parameter for win input argument when using EGL_EXT_compositor as off-screen windows. Note: The configurable parameter, for example, could be read from a file or passed in at startup of the software component. See Configuration Services in Section 3.2.5.
5.5.2.4.6	A Graphics Services UoC using *eglCreateContext* shall use a configurable parameter for the EGL_EXTERNAL_REF_ID_EXT attribute during context creation when not using the EGL_PRIMARY_COMPOSITOR_CONTEXT_EXT attribute.

5.5.2.4.1

This requirement is specifying to the UoC developer that they are limited to using EGL, Version 1.4. The use of the compositor extension should be constrained to the CDS or standalone PSSS OpenGL display manager application.

EGL, Version 1.4 was chosen as a compromise between backwards compatibility with 1.2 while still moving towards a more recent version of EGL. EGL, Version 1.4 provided the framework necessary to implement the compositor extension.

5.5.2.4.2

The Safety and Security Profiles did not allow the use of OpenGL ES 2.0 because OpenGL SC 2.0 is a safety tailored subset of OpenGL ES 2.0. Any UoC developer who would want to utilize OpenGL ES 2.0 capabilities in these profiles would be required to do a similar tailoring exercise to achieve safety and/or security. Allowing OpenGL ES 2.0 in these two profiles would be redundant.

5.5.2.4.3

This requirement was created to support application portability. The note is specifying that the application is allowed to query OpenGL for extensions but must be able to properly operate if no extension is detected. If choosing to attempt to dynamically bind OpenGL extensions, achieving the conformance will become more difficult.

5.5.2.4.4

This is for UoC compatibility with the compositor extension.

5.5.2.4.5

This is for UoC compatibility with the compositor extension.

5.5.2.4.6

This is for UoC compatibility with the compositor extension.

5.5.3 PSSS Requirements for Graphics Services

5.5.3.1 PSGS Sub-Segment Requirements

Table 5 shows the PSGS Sub-Segment Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 6: PSGS Sub-Section Requirements for Graphics Services

Section	Requirement Text
5.5.3.1.1	When a PSSS Graphics Service UoC renders using a method other than OpenGL, as defined in Section 3.12.7, the Graphics Service UoC shall use the graphics driver API. Note: This allows a PSSS Graphics Service UoC to use interfaces not defined in the IOS or OSS APIs for graphics rendering.

5.5.3.1.1

The intent of this requirement is to define that Graphics Services implemented in the PSSS layer can use other non-standardized graphics APIs not referenced in this standard. The requirement was originally added to allow ARINC 661 CDS implementations the ability to be implemented without using OpenGL. It has since transformed into allowing frameworks such as QT, GLX, and others. This requirement also does allow Graphics UoCs to be developed in the PSSS using proprietary graphics APIs. This practice is not in the spirit of the FACE Technical Standard and does not create portability, so should be avoided.

5.5.3.2 *PSSS Cockpit Display System Requirements*

Table 7 shows the PSSS Cockpit Display System Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 7: PSSS Cockpit Display System Requirements for Graphics Services

Section	**Requirement Text**
5.5.3.2.1	The ARINC 661 CDS UoC shall provide OpenGL windowing using the external source widget when configured to support Graphics Display Management Services.
5.5.3.2.2	A Graphics Services UoC implementing an ARINC 661 CDS in the PSSS shall support the logic specified in ARINC 661-5 for widgets it provides.
5.5.3.2.3	A Graphics Services UoC implementing an ARINC 661 CDS in the PSSS shall provide the minimum widget subset as specified in Table 10. Note: The ARINC 661 CDS may provide additional widgets.
5.5.3.2.4	A Graphics Services UoC implementing an ARINC 661 CDS shall use the TS Interface to communicate ARINC 661 data.
5.5.3.2.5	A Graphics Services UoC implementing an ARINC 661 CDS shall use the XSD defined in Section H.2 for its style data configuration.
5.5.3.2.6	A Graphics Services UoC implementing an ARINC 661 CDS shall use the XSD defined in Section H.3 for its display management configuration parameters. Note: The Graphics Services UoC implementing an ARINC 661 CDS in the PSSS may have direct access to any proprietary graphics hardware drivers and APIs.
5.5.3.2.7	A Graphics Services UoC implementing an ARINC 661 Display Management UA shall have an associated ARINC 661 DF.
5.5.3.2.8	A Graphics Services UoC implementing an ARINC 661 CDS shall document the set of ARINC 661 widgets it provides.

5.5.3.2.1

The intent of this requirement is to define the ARINC 661 CDS as the window compositor in the system. In practice this would be used with the EGL compositor extension and a multi-partition-capable OpenGL device driver.

5.5.3.2.2

This requirement specifies to the ARINC 661 CDS developer that ARINC 661-5 is the standard that should be used in development. At the time of the writing the ARINC 661-6 had not been released. The ARINC 661 will continue to evolve more rapidly than the FACE Technical Standard. UoC developers are allowed to use this new additional capability but will not be guaranteed that the CDS will supply it. This will need to be worked out between platform providers and UoC developers during the requirements capture phase.

5.5.3.2.3

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms, but it is expected that all CDS implementations will tailor a superset of widgets for the specific platform capabilities. This being the case, CDS providers should provide a documented list of widgets for their platform-specific CDS.

5.5.3.2.4

This is a clarifying requirement to specify that an ARINC 661 CDS must communicate to the ARINC 661 UAs via the TSS interface; this was not clear in FACE Technical Standard, Edition 2.1. It should be known that the ARINC 661 interface is not required to be data modeled per J.2.4.20.

5.5.3.2.5

The ARINC 661 Specification does not provide a means to configure the reference handles for things like lines styles, colors, etc. This XSD is our means to provide the configuration required for a consistent representation. For example, a UA can be assured that the handle for a given color is the same on all platforms.

5.5.3.2.6

The ARINC 661 Specification does not provide a means to configure physical attributes of the hardware. This XSD gives users of the FACE Technical Standard a standardized way to configure these elements. This file would best be described as the Master Integrators file.

5.5.3.2.7

Each individual UA will require an associated Binary Definition File that defines the layout of the HMI. The final composed HMI will be made up of a collection of various BDF files controlled by a Display Manager which will be developed by the end System Integrator. Structure of the BDF is defined in the ARINC 661 Specification and the FACE Technical Standard does not modify this in any way.

5.5.3.2.8

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms, but it is expected that all CDS implementations will tailor a superset of widgets for the specific platform capabilities. This being the case, CDS providers should provide a documented list of widgets for their platform-specific CDS.

5.5.3.3 *PSSS User Application Requirements*

Table 8 shows the PSSS User Application Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 8: PSSS User Application Requirements for Graphics Services

Section	Requirement Text
5.5.3.3.1	A Graphics Services UoC implementing an ARINC 661 UA shall use the TS Interface to communicate ARINC 661 data.
5.5.3.3.2	A Graphics Services UoC implementing an ARINC 661 UA shall document the set of ARINC 661 widgets it uses.

5.5.3.3.1

This is a clarifying requirement to specify that an ARINC 661 CDS must communicate to the ARINC 661 UAs via the TSS interface; this was not clear in FACE Technical Standard, Edition 2.1. It should be known that the ARINC 661 interface is not required to be data modeled per J.2.4.20.

5.5.3.3.2

The ARINC 661 Specification does not require that all ARINC 661 widgets be implemented to be conformant. This would be a costly and in many cases unnecessary task. The FACE Technical Standard, Edition 3.0 does specify a minimal list of widgets (Table 10) that UA providers can be sure are available on all platforms, but it is expected that all CDS implementations will tailor a superset of widgets for the specific platform capabilities. This being the case, CDS providers should provide a documented list of widgets for their platform-specific CDS.

5.5.3.4 *ARINC 739A Requirements*

Table 9 shows the PSSS ARINC 739A Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 9: PSSS ARINC 739A Requirements for Graphics Services

Section	Requirement Text
5.5.3.4.1	A Graphics Services UoC shall use ARINC 739A messages defined in Section 3.7 of ARINC 739-1 or ARINC 739A-1 when communicating with the ARINC 739A Services.

5.5.3.4.1

This requirement is stating that when using ARINC 739, there is a defined message set that needs to be used for it to work properly. This message set is specified in an externally referenced standard.

5.5.3.5 OpenGL Requirements

Table 10 shows the PSSS OpenGL Requirements for Graphics Services. The corresponding subsequent subsections contain details on guidance for adherence to the requirements.

Table 10: PSSS OpenGL Requirements for Graphics Services

Section	**Requirement Text**
5.5.3.5.1	A Graphics Services UoC using OpenGL shall also use EGL, Version 1.4. Note: A Graphics Services UoC may only use extended EGL as specified in the OSS graphics requirements.
5.5.3.5.2	A Graphics Services UoC using OpenGL in the: a. General Purpose Profile shall use OpenGL SC 1.0.1, OpenGL SC 2.0, or OpenGL ES 2.0. b. Safety Profile shall use OpenGL SC 1.0.1 or OpenGL SC 2.0. c. Security Profile shall use OpenGL SC 1.0.1 or OpenGL SC 2.0. Note: Graphics Services UoC may not use extended OpenGL and be conformant.
5.5.3.5.3	A Graphics Services UoC shall be restricted to the core profile for the OpenGL version being used. Note: This does not preclude use of dynamic binding to use OpenGL extensions; however, the UoC must not rely on the existence of any OpenGL extensions.
5.5.3.5.4	A Graphics Services UoC using *eglGetDisplay* shall use a configurable parameter for the *display_id* input argument. Note: The configurable parameter, for example, could be read from a file or passed in at startup of the software component. See Configuration Services in Section 3.2.5.
5.5.3.5.5	A Graphics Services UoC using *eglCreateWindowSurface* shall use a configurable parameter for win input argument when using EGL_EXT_compositor as off-screen windows. Note: The configurable parameter, for example, could be read from a file or passed in at startup of the software component. See Configuration Services in Section 3.2.5.
5.5.3.5.6	A Graphics Services UoC using *eglCreateContext* shall use a configurable parameter for the EGL_EXTERNAL_REF_ID_EXT attribute during context creation when not using the EGL_PRIMARY_COMPOSITOR_CONTEXT_EXT attribute.

5.5.3.5.1

This requirement is specifying to the UoC developer that they are limited to using EGL, Version 1.4. The use of the compositor extension should be constrained to the CDS or standalone PSSS OpenGL display manager application.

EGL, Version 1.4 was chosen as a compromise between backwards compatibility with 1.2 while still moving towards a more recent version of EGL. EGL, Version 1.4 provided the framework necessary to implement the compositor extension.

5.5.3.5.2

The Safety and Security Profiles did not allow the use of OpenGL ES 2.0 because OpenGL SC 2.0 is a safety tailored subset of OpenGL ES 2.0. Any UoC developer that would want to utilize OpenGL ES 2.0 capabilities in these profiles would be required to do a similar tailoring exercise to achieve safety and/or security. Allowing OpenGL ES 2.0 in these two profiles would be redundant.

5.5.3.5.3

This requirement was created to support application portability. The note is specifying that the application is allowed to query OpenGL for extensions but must be able to properly operate if no extension is detected. If choosing to attempt to dynamically bind OpenGL extensions, achieving the conformance will become more difficult.

5.5.3.5.4

This is for UoC compatibility with the compositor extension.

5.5.3.5.5

This is for UoC compatibility with the compositor extension.

5.5.3.5.6

This is for UoC compatibility with the compositor extension.

6 Safety Guidance

A formal safety program may require Configuration Management (CM) of all software life cycle artifacts including requirements, design, code, test reports, and supporting documents. This requirement is driven by the Airworthiness Authority. Safety is assessed at the system level. Safety certifications are not given to FACE UoCs, services, UoPs, or packages. The cognizant Airworthiness Authority should be involved during all stages of safety-critical software development and test.

The Design Assurance Level (DAL) of the OSS must be equal to or greater than the highest DAL required by the system. Any component in a safety-critical partition should have artifacts for certification at the level of the partition. FACE UoCs use APIs defined in the FACE Safety Profile to support system-level safety requirements.

The FACE Safety Profile is described in the FACE Technical Standard §3.2.1.4. The FACE Safety Profile is defined for all the FACE Segments: OSS, IOSS, PSSS, TSS, and PCS. For each of the segments, the Safety Profile limits the set of APIs necessary in developing safety-related software components.

While software itself is an abstraction, and thus has no substance and cannot directly harm people, property, or the environment – it is the nature of the software's sensing and control of physical components within a system and its environment that render that system safe or not. Safety of software, then, depends on the combined interactions of the software with hardware, system operator, and external factors. Leveson's (1995) definition of software system safety, while consistent with the Naval Sea Systems Command (NAVSEA) definition of software safety, is more explicit about this dependency:

"Software System Safety implies that the software will execute within a system context without contributing to hazards."

6.1 Safety Overview

In very general terms, system developers must convince a qualification authority that they achieve a level of safety appropriate to the potential hazards inherent in the system in order to attain authorization to field avionics that include a software base. A developer must establish the following at a level of detail that matches the criticality established through safety analysis:

- The system software results from the correct set of high-level requirements, decomposed into lower-level requirements, from which code was generated to implement the functions defined thereby

  Further, no functionality is implemented in the software that is not described by the requirements. The requirements must (all) be shown to be represented by a set of test procedures, execution of which verifies that the software correctly implements the requirements; it must be documented that the tests were executed with acceptable results. The software must be proven to produce exactly one result when presented with a given set of inputs. Traceability of the requirements to the software and to the tests must be documented in a manner conducive to evaluation of these concerns.

- The documentation shows that proper development procedure, CM, and Quality Assurance (QA) is incorporated into the program
- A plan is established to address software maintenance and revision procedures that protects the integrity of the aforementioned considerations

The documentation artifacts required by an Airworthiness Authority are not goals in and of themselves; they are documentation showing that the proper goals were achieved during the program. Early generation of the appropriate plans and coordination of their content with the cognizant Airworthiness Authority increases the likelihood of efficient achievement.

It is absolutely impractical to properly achieve the goals associated with this documentation in retrospect at the program's end. Early, in-depth involvement of the qualifying authority is key to achieving efficiency in this process through:

- Providing engineering cognizance of the various development processes
- Establishing agreement as to task requirements that lead most directly to qualification
- Agreement that goals are achieved at the appropriate program phase

Conformance to the FACE Safety Profile is intended to impose the following key attributes that support qualification, depending on the evaluated criticality level:

- Non-interference between applications
- Non-interference between applications and the operational environment
- Non-interference between applications and external interfaces
- Software fault isolation
- Software fault recovery
- Reliable information flow between applications
- Reliable information flow between applications and external interfaces
- Integrated health monitoring and management
- Standards-based system configuration data

6.2 Purpose

This text applies to reuse of portable software components, particularly those developed specifically for reuse. Integrators accept the software component, its associated documentation plan, and perform the technical work to integrate that software component into a software system. Integrators then seek a certification of airworthiness from the appropriate Airworthiness Authority, based on the evaluation of a documented qualification effort. To make the case for qualification of the software system that incorporates the component, the integrator offers documentation that was delivered with the software component, and also provides substantiation that all additional qualification objectives have been satisfied.

6.3 Applicability

In addition to the airworthiness qualification requirements defined by the cognizant military authority, systems incorporating FACE UoCs may fall under the requirements applied by Civil authorities. DoD aircraft operating in the National Airspace (NAS) must consider those requirements, and similarly must address the requirements of international regulatory authorities for airspace in which they operate; e.g., International Civil Aviation Organization (ICAO), etc.

In general, these Civil requirements are more stringent than those imposed by DoD agencies, although, for obvious reasons, they are somewhat in alignment.

6.4 Safety Assessment and Design Assurance Level

Safety-related software components are those for which a safety analysis has determined that a high level of criticality exists, requiring achievement of qualification tasks and documentation of development practices consistent with a high level of rigor. Since this qualification potentially represents significant impact in terms of cost and schedule, it is important that criticality be properly established for prioritization and planning of qualification early during the program. At a high level, the following sections address safety concerns for software:

6.4.1 Start Assessment Very Early and Update as the Program Progresses

Every system development must include a safety program, which necessarily includes a Safety Assessment (SA) process.

The SA process provides a methodology to evaluate aircraft functions and the design of systems performing these functions, and seeks to establish that the associated hazards have been addressed.

The SA process for a system must establish that all failure conditions and significant combinations of failures have been considered, including additional complexities arising from integrated system architectures.

At a top level, stated in common terminology, the SA process comprises a Functional Hazard Assessment (FHA), a Preliminary System Safety Assessment (PSSA), and a final System Safety Assessment (SSA). The SA process is iterative, tracking the system development as it progresses, and should be executed as an inherent part of the process. As the program progresses, it is expected that each analysis will be revised. The software development plan can be used to document this strategy.

The SA process begins with concept design, and derives any safety requirements for the design. These derived safety requirements are documented as part of the software requirements, and are expected to trace directly to source code and test cases for verification of implementation.

The endpoint of the SA process is definition of the safety and safety-related requirements as input to evaluation of criticality for software components. The SA process concludes with verification that the design meets the safety requirements.

The FHA identifies and classifies the failure conditions associated with the aircraft functions (and combinations thereof) at the beginning of the development cycle. Classification of failure conditions (establishing criticality) establishes the safety objectives. The output of the FHA is input to the PSSA.

The PSSA is a systematic examination of the proposed system architecture(s) to determine how failures can cause the functional hazards identified by the FHA. The goal is to establish system safety requirements; these are derived requirements (not traced directly from higher-level requirements), that require verification as part of qualification to determine compliance with the system safety plan.

The SSA is a systematic, comprehensive evaluation of the implemented system to show that safety objectives from the FHA and derived safety requirements from the PSSA are met. Requirements at all levels that trace to safety-related functionality must be identified, since they impact criticality evaluations, engineering trade space decisions, and regression testing required by software maintenance and revision.

A software Failure Modes Effects Analysis (FMEA) is also a systematic and comprehensive evaluation from a bottom-up view of the system. The effects of failures within the system on the end result are determined. An FMEA is encouraged to understand the ramifications of such errors.

6.5 DAL and Qualification Cost are Directly Proportional

6.5.1 Assess Criticality During First Application, or as Part of a Notional Application

Limitations of cost and schedule drive software developers to plan and execute qualification at the lowest practical level of criticality. For that reason, proper assessment of criticality is key to integrating the proper processes, qualification activities, and documentation into the engineering effort from the start (refer, for example, to the qualification objectives listed for criticality levels A through E in the RTCA DO-178B/C). Developers face the challenge of cost-effectively engineering and marketing portable software for integration into systems where the criticality level is not known. If it is not practical for a developer to work for qualification at the highest level of criticality, then a software component must be developed as part of some real or notional system of “typical” criticality.

It is not possible to demonstrate some aspects of qualification in the absence of a supporting system. In fact, it is not likely that all of the development can be done without some notion of a system around a component (e.g., debugging logic, unit testing).

Although most FACE UoCs and UoPs might be developed as part of complete systems, the context of the FACE Technical Standard does open up a market for software developed specifically as portable components, and suppliers should consider the qualification implications for the integrators of their software to maximize reuse.

6.5.2 Plan for Future Use of Functionality at a Higher DAL

The expense associated with qualification varies significantly (~25% to 150% of the project cost) with the assessed criticality of the functionality provided by software. The immediate benefits of qualification at the initially assessed level of criticality may be outweighed, however, by the long-term benefit of increased use of software if accompanied by artifacts that support qualification at higher levels. Therefore, suppliers of FACE UoCs should consider achievement of a higher level of qualification than immediately necessary, and to reflect a higher level of documentation in the FACE Registry.

6.5.3 Safety Assessment May Vary by Context

Integrators of FACE UoCs should be aware that the hazard evaluation of functionality in their system may result in the need to qualify those components at a higher level of criticality than can be substantiated by the artifacts available from the component supplier. This situation represents a significant risk in terms of cost and schedule if artifacts to support qualification at a higher level must be reverse engineered, or if increased levels of test or structural/coverage analysis are necessary. If artifacts are not available, it may not be possible to generate them, leaving a significant deficit in an integrator's qualification picture when seeking Airworthiness Release (AWR). Suppliers of FACE UoCs and subsequent integrators should cooperate early to guard against this, and should coordinate with Airworthiness Authorities to document and execute a qualification program that leads to successful flight release and fielding.

6.5.4 Plan for the Most Critical Application

FACE UoCs are particularly likely to see use in new contexts that were not foreseen by their developers. This is likely to be true when UoCs provide support elements or generic services. It is also possible that a UoC provides functionality that has safety implications in a new integration that are higher than those provided in the initial system. The functionality provided by FACE UoCs could be evaluated to have a higher criticality by a different Airworthiness Authority considering a subsequent integration. To ensure the broadest audience for FACE UoCs, a developer should consider, and evaluate the expense of, qualifying their software at higher levels of rigor if they intend to provide qualification artifacts to aid subsequent integrators when they seek qualification for their systems.

6.5.5 Safety Assessment of Configurable Software

Implications of configurable software require special consideration during the SA to ensure that consequences of incorrect operation are considered. The qualification strategy for complex software may include the concept of configurability at run-time. This configurability is usually based on the contents of small configuration files, read by the software during start-up, that cause operation to take a variety of configurations. This alleviates the generation and maintenance of many different software loads, rather than one larger application that can accommodate these configurations, although the qualification effort may be somewhat greater because of the increased complexity. Other methods of configurability include hardware strapping and user interface via entry of codes or option selections while the software is in maintenance mode; particular attention to allowances for "dead and deactivated" code is advised, and early coordination of approach is key to efficiency in development and qualification.

Developers and users of software components may need to address configurability as they seek flight release for their systems in their plan for qualification, which must address the variability associated with configurability. Application of this concept is likely to result in a code base that includes "deactivated code" that traces to software requirements, is documented, tested, and managed with the rest of the software, but which is not always executed for specific operational configurations. Consensus with the Airworthiness Authority for this approach is required. Also, additional testing must verify the proper operation of the configuration mechanisms of the software, and must include robustness testing that proves the fault tolerance of the final product. Response of the software to invalid configuration information (e.g., out-of-bounds and incorrect values, etc.) must be shown to be in conformance with formal requirements, with no adverse effects.

Qualification of an integration that includes configurable software components can become a complex issue that requires careful coordination. Careful design is required to achieve the perceived benefits.

6.6 Qualification Advantage of Various Software Reuse Strategies

6.6.1 Basic Principles of Software Qualification

In order to ensure safety in Civil aviation, the Federal Aviation Administration (FAA) evaluates a supplier's plans, execution, and documentation of their processes for the development, verification, CM, and maintenance of software in their products. Suppliers of software used in military avionics must consider more than safety, including items relevant to ensuring mission support. This results in the imposition of more qualification requirements. Suppliers of software who participate in military acquisitions will be subject to a mix of these requirements, based on situation, including Civil applicability, Airworthiness Authority agreements, etc.

As technology progresses, the tools available to generate software improve, resulting in an exponential growth in the volume and complexity of software systems. Consequently, it has become impractical or impossible to generate a complete set of tests to verify an implementation in every respect. Therefore, the evaluation of criticality, imposition of various levels of rigor in verification, and additional focus on QA, configuration control, tool quality, requirements traceability, and documentation is necessary to achieve acceptable levels of safety and to ensure mission capability to the appropriate level of rigor.

Core elements that comprise the qualification picture include:

- Definition of user need
- Definition of system requirements
- Safety analyses to establish criticality and therefore the required level of verification rigor

  Often safety analysis results in the determination of a DAL for a system that can map more directly into certification objectives defined in DO-178 or DO-254, for example.
- Identification of safety-related requirements
- Decomposition of system requirements into detailed software requirements
- Decomposition of requirements into a software design
- Generation of software source code that implements the software requirements
- Generation of a sufficient set of tests to show that the software implements its requirements
- Test report proving that the software implements its requirements ("pass")
- Traceability document(s) or evidence showing the mapping of requirements to design, to source code, to test cases/results including proof that the traceability is "bidirectional" Version Description/Configuration Management

- Life cycle support plan (error collection/correction and distribution)
- All other artifacts that provide the details required by the cognizant Airworthiness Authority

The level of rigor defines the set of particular artifacts that are required, and definition of the circumstances under which they must be developed, verified, delivered, and maintained. Achievement of proof of qualification represents a significant expense in terms of cost and schedule that can increase dramatically with an increasing level of criticality. It is therefore important to properly perform functional and hazard analyses early in the development cycle to establish the qualification objectives that must be achieved, since many of the requirements are best executed at the appropriate stages of development (early) rather than by post-documentation at later program stages. Some artifacts are effectively useless if written late in a program simply to satisfy documentation requirements (i.e., a software development plan).

The qualification effort should consist of executing a valid engineering process to generate system software and documenting that process, and its achievements. Therefore, this effort is best executed according to a formal agreement, established early with the correct qualification authority and maintained at each phase, rather than through a flurry of document generation immediately before seeking a flight release. The latter approach is less efficient, less accurate, and more expensive.

For FACE UoCs or UoPs, the qualification process is complicated by the intent to generate software components that can be integrated into many software systems (i.e., portable software). In order to achieve the stated goal of reducing development effort, integration costs, and time-to-field avionics capabilities, suppliers of FACE UoCs or UoPs should consider the qualification implications of their products, and coordinate with the appropriate release authority in the initial stages of software development or early stages of programs that incorporate FACE UoCs or UoPs into the system .

6.6.2 Additional Qualification Considerations

6.6.2.1 Memory Management

Proper memory management of code requires ensuring any memory allocated is properly deallocated to return memory to the heap when no longer needed or used by the program. If dynamic memory allocation is used, improper management of memory may result in memory leaks, resulting in anomalous behavior. For safety-critical code, static allocation of memory should be employed so that memory requirements are defined before, during, and after program execution. Stack and heap areas of memory cannot be allowed to overflow the memory space allocated, which often propagates execution errors.

Proper pointer management is also critical to ensure correct operation. Use of dynamic memory allocation is not recommended beyond UoC initialization due to possible access conflicts, although mechanisms can be incorporated and qualified to alleviate this concern.

Proper management of cache use must be substantiated. It is likely that this issue will present qualification challenges in multicore applications in the future.

In this regard, the guidelines for restricting use of specific high-level languages should be applied to support qualification at the level of rigor associated with the FHA. Examples would be: adherence to the Motor Industry Software Reliability Association (MISRA) restrictions for C, the Ravenscar restrictions for Ada, and similar subset restrictions for C++ and Java®.

Some level of requirements-based and robustness testing should be done to assure proper memory management.

6.6.2.2 File Management

File system management is also critical to proper operation when opening files, providing read/write access, and closing files. If a file is opened for operations then it must be closed properly. If multiple readers and writers are operating on the file then proper locking mechanisms should be managed to prevent corrupting the files. Inadvertent closure of files during operation should be prevented or handled properly by exception processing. The integrity of data within the file should be ensured via a cyclic redundancy check or checksum. Some level of requirements-based and robustness testing should be done to assure proper file handling. This guidance implies there should be requirements defining the system's characteristics in these areas.

6.6.2.3 Concurrency Management (Semaphores, Mutexes)

The spurious effects of a multi-threaded system without proper concurrency management can lead to indeterminate states. Mutexes and semaphores must be properly employed. Some level of requirements-based and robustness testing should be done to assure proper concurrency handling, implying that detailed requirements should exist to define these aspects of the software.

6.7 Overview of the Principles of Portable Software

Given the qualification overview above, the reuse of software at any criticality level where artifacts are to be reused or any qualification credit is sought requires special attention. While it is beneficial to reuse and repurpose software, reuse of certification data may present some challenges when a software component is inserted into another platform. The following sections document considerations when using or reusing FACE UoCs or UoPs with a previously-approved certification basis.

6.7.1 Documentation of Early Agreement

As with any software qualification, but especially true when a reuse strategy includes flight release requirements, early coordination of a reuse approach amongst the stakeholders is key to achieving an efficient qualification. Stakeholders include the developer, first integrator (if different), the cognizant Airworthiness Authority, and any other entity involved in the qualification effort.

6.7.2 Specific Achievements for Portable Software during the First Qualification

Developers of software components can significantly impact the marketability of their components by planning a qualification effort that includes the precepts of reuse that apply to their organization. Since these vary by service and situation, a developer should have a plan and seek coordination early, adjusting course as necessary to maximize effective reuse. Developers should also document reusable components separately and coordinate "qualification with reuse" as a specific goal.

6.7.3 Achievements Remaining for Re-Application of Portable Software

Developers should evaluate the qualification objectives that cannot be met during qualification of their component(s), and document these for use by subsequent integrators (e.g., customers). Prudent integrators will insist on the existence of proper artifacts to ensure their success in their own qualification efforts, and savvy suppliers of FACE UoCs or UoPs will work to accommodate them.

6.7.4 Qualification of Portable Software as Part of a System Qualification

Civil guidance for qualification of software components with the intent of reuse includes qualification of the software components first as part of a software system. This seemingly prohibitive requirement makes sense when considering that the nature of qualification pervasively concerns itself with verification of implementation of all software requirements, and this implies testing the implementation of the requirements that define the interfaces of the software component. Without qualification of the interfaces, the operation of the software cannot be fully verified and, therefore, the qualification would be significantly deficient. Suppliers who do not qualify their FACE UoCs or UoPs before posting to the FACE Registry for consumption should plan to support the qualification efforts of the integrators of their software, and integrators should be aware of the significant qualification implications of using software with no qualification pedigree.

6.7.5 Separate Documentation Requirements for Portable Software

FACE UoCs or UoPs that are first safety-qualified as part of a complete software system should be separately documented. Submission of system qualification artifacts with FACE UoCs or UoPs to the FACE Registry puts the integrators of those FACE UoCs or UoPs at risk by requiring that they disentangle the qualification picture for the software components from the documentation of the entire system of which they were originally a part.

Those who reuse existing software components in subsequent system developments must consider installation, safety, security, operational, functional, and performance issues for each project. The integration of software components into new systems has both technical and qualification implications. Integrators should expect that their Airworthiness Authority will require them to substantiate not only the qualification of the FACE UoCs or UoPs they integrate, but the quality of the integration. Moreover, integrators need to address the suitability of the reused components, from both a functional and a safety perspective to application in their systems. Analysis of performance may also be required to show suitability in a given integration.

6.8 Types of Reuse

6.8.1 Binary Reuse

Software designed for binary portability and reuse is not a quality attribute of FACE UoCs or UoPs. Reuse of binary software modules, while offering the greatest qualification advantage, imposes stringent documentation of configuration and documentation of specific qualification objectives. Binary reuse restricts use to those software operating environments that match those present when the software component was originally qualified. Alteration of the binary of this type of software reuse module tends to negate the stated qualification advantages, and is subject to review and concurrence by the appropriate authority. Software reuse at a binary level requires careful coordination with the certifying Airworthiness Authority.

6.8.2 Source Code Reuse

FACE UoCs or UoPs are ideally designed for source code reuse, as source code portability is a goal of the FACE Technical Standard. Reuse of code at this level broadens the range of reuse by accommodating additional hardware configurations, software environments, compilers, libraries, frameworks, and virtual machines. While qualification of the final binary is required, great advantage is available through reuse of artifacts such as high and low-level requirements, development documentation, algorithmic verifications, and particularly reuse of traceability information that substantiates the decomposition of the design requirements as they relate to the software functions. If properly developed and documented, and especially when properly automated, the verification test suite can be made reusable as well.

6.8.3 Reuse of Artifacts

Reuse of software at any level lends itself to the reuse of documentation artifacts that substantiate the engineering and qualification process. For software developments where the component is intended for reuse, developers should generate these artifacts separately for the reusable component(s). This allows them to be part of the delivery, and clarifies the parts of each artifact that apply to the reuse case. Reuse of artifacts that describe the whole software development that were used during initial qualification are not specific to the reusable component, and the extra content would lead to confusion.

6.9 Types of Artifacts

6.9.1 Requirements

Well-written requirements are the basis for the development, documentation, and qualification efforts of software components. Reuse increases the importance of quality requirements, in that subsequent integrators must rely on the requirements to communicate the characteristics of the software component for their own use, and for use in their qualification efforts.

The DO-178 objectives identify key characteristics for software requirements. These include:

- Testable – verifiable
- Code-able – all requirements must trace to code
- Singular – one "shall" per paragraph
- Self-contained – requirements do not inherit requirements from referenced sources, nor do they depend on definitions nor specifications stated elsewhere
- Imposes concise statements – "system shall have as a goal" is not a meaningful requirement
- Consistent – requirements are not conflicting
- Complete
- Unambiguous

  Note: Unambiguous implies specific enough to be testable and useful in established test coverage; for example, "system X shall operate when the landing gear is down" may mean that a positive gear down indication is read from the hardware, but could be misconstrued as: the “gear up and locked” indicator is “off”. These are two distinctly different conditions if the gear did not fully deploy.

- Positive

  Note: Requirements should not be stated in the negative. For example, “diagnostics shall not impact system performance” is both vague and untestable, in that it cannot be proven that there will never be an impact, and it is not specified what an “impact” consists of.

Consequences of poorly written requirements include:

- Poor product quality
- Not meeting customer needs
- Increased development/rework cost
- Requirements volatility
- Program delays – rework, retest, field maintenance
- Incomplete/wrong validation and verification – rework, retest, reduced safety, functionality
- Increased rework/maintenance cost

Analysis has shown that the cost to fix a software problem increases exponentially as it is discovered later in the life cycle. Good requirements result in better implementation, unit testing, and earlier problem resolution.

In order to properly reuse a software component, detailed information must be available as to the functionality that it provides. When qualifying a software system that incorporates a particular portable software component, the requirements for the portable component will be central to the verification of the implementation in software for that component. Each requirement must have a program-unique identifier for use in establishing traceability from high to low-level requirements, from requirements to verification, and from requirements to code. Also, each requirement involved in the implementation of safety-critical functionality must be identified. During regression testing, the software that implements safety-critical functionality must be retested.

A key characteristic of FACE UoCs or UoPs that are engineered for reuse is that their set of requirements can be used to document the component in the context of a new integration with no modification, or possibly with little modification.

6.9.2 Tests

Test documentation typically consists of a test plan, test description, and test results which may include analysis of source or object code coverage. In general, the test plan provides an overview and the general terms of verification testing, and the test description gives the specific details of the tests that will be executed to verify the software implementation. The test plan may be generic enough to describe testing for the portable software component as well as for the entire software suite (during initial qualification), but the test description must separately detail the test steps defined to verify the implementation of each requirement in the FACE UoCs or UoPs.

The test plan should document the procedure for recording changes to the test sequence required during testing, and the procedure for feeding these changes back into the test sequence to ensure accurate testing in the future. This is particularly critical to test plans and descriptions that accompany portable software components for integration testing or verification of compiled code during the qualification activities for subsequent integrations.

Testing that verifies the operation of interfaces to the FACE UoCs or UoPs must also be detailed in the test description or in other documentation. This is critical to the qualification of the new integration of a software component into a new software system, and represents a critical component of the qualification objectives that remain to be done by an integrator.

6.9.3 Traceability

Traceability refers to an evaluator's ability to analytically establish a clear connection between a software specification, its design and implementation, and a corresponding set of test results.

A system designer produces a set of system requirements that are decomposed into a set of detailed high-level and low-level software requirements, from which software source code can be generated. To verify traceability, a sufficient set of accurate tests must be defined and executed to establish that all of the software requirements were correctly implemented.

Bi-directional traceability means that tracing can be both up and down from the software requirements, through the code and all the way to the test cases and results for each requirement. Tracing upwards means starting with the test results and ultimately arriving at a software requirement while tracing through the implementation and design.

In order to evaluate qualification status for a certification decision, an evaluator must be able to establish traceability using the documentation provided with the software. Reusable software components (as defined in FAA Advisory Circular 20-148) must be provided with traceability documentation specific to the software component, not documentation of the entire system of which the component was originally a part.

In order for a qualification analysis to include substantiation that a test suite has been executed that verifies the implementation of a FACE UoC's or UoP's detailed requirements, the traceability of the requirements to the software and to the set of tests must be provided.

Again, each requirement must have a program-unique identifier for use in establishing traceability from high to low-level requirements, from requirements to verification, and from requirements to code, and each requirement involved in the implementation of safety-critical functionality must be so identified.

6.9.4 IRS/IDD/ICD

Key documentation for reuse of software components is the interface description, including Interface Requirements Specification (IRS), Interface Design Description (IDD), and ICD. The Data Item Descriptions (DIDs) referenced by the Contract Data Requirements Lists (CDRLs) commonly found in DoD acquisitions describe various formats for the content required to define the interfaces in a system. Documentation of interfaces is key to the successful reuse of FACE UoCs or UoPs, as they provide value through the capabilities they provide to the systems into which they are integrated. Interface definitions must accurately depict not only the capability provided by a software component, but the format in which it is provided. As is the case with software requirements, a software component such as a UoC or UoP has little value unless accompanied by formal documentation that specifies its interfaces. Innovations in acquisition methodology may rely on complete definition of a component’s interfaces, especially if that interface is properly modeled in a format that can be used to accurately specify, and perhaps validate, its interface characteristics.

6.9.5 Miscellaneous Software Documentation

Airworthiness qualification of software will entail more than generation of requirements and test data. The software development life cycle called out in many standards (DO-178, MIL-HDBK-516, etc.) requires a planning process, development process, configuration management process, quality process, and more. The planning process will often result in the creation of the following plans and standards:

- Software Certification or Qualification Plan
- Software Development Plan
- Software Verification Plan
- Software Configuration Management Plan
- Software Quality Assurance Plan
- Software Requirements Standard
- Software Design Standard
- Software Coding Standard

  Note: The list above is derived from civilian airworthiness guidance and may nor may not be required for any given program of record.

The software development process will result in the creation and review of the requirements and design documents as well as reviews of the source code. For FACE UoCs or UoPs, it may also include a software vulnerability or hazard analysis so that integrators get insight into potential failure conditions and if the FACE UoCs or UoPs can mitigate or isolate failures. If the FACE UoCs or UoPs cannot by themselves mitigate failure conditions, the hazard analysis should provide guidance to the integrator on other ways to mitigate a given failure.

6.9.6 Software Configuration Index or Version Description Document

The software configuration index or version description document should capture the final configuration of all software life cycle data produced during the qualification process. This would include configuration data of documents, tools, source code, binary files, test files, test results, and more. The configuration information (version numbers, CM labels, path names, etc.) should be captured in sufficient detail that it can be identified and audited by an airworthiness representative should the need arise.

6.9.7 Software Accomplishment Summary or Qualification Summary

The Software Accomplishment Summary (SAS) describes the developer’s evaluation of the qualification objectives that were met during qualification of the system software for the portable software component in particular. The SAS for the portable component is separate from the SAS for the system. The terms under which a certification authority will consider qualification of a software system that integrates one or more software components must be established by consensus. This evaluation is based in part on the content of the SAS. Given that the developer has the best technical appreciation for the software's functionality, implementation, and for the qualification activities, the developer must document the qualification objectives that were considered to have been partially met during the initial qualification, and those that are considered to be required for completion by each integrator. In addition, a description of the activities necessary to complete the objectives must be included. The qualification objectives not addressed for the software component during the initial qualification must be listed along with a summary of activities for the integrator to fulfill these remaining objectives.

The accomplishments summary or qualification summary should also provide a list of open defects documented at the time of qualification. Justification for all open defects should be provided that demonstrate no untoward safety impacts on the approved software baseline.

Documentation of these details by the developer and their evaluation by the certification authority will be essential to evaluation of the qualification merits of a software reuse component during assessment of each subsequent integration. If the same Airworthiness Authority evaluates a subsequent integration, the qualification picture is clearer than in cases where a different Airworthiness Authority considers the merits of software reuse; careful documentation is necessary to maximize the benefits of software reuse in the latter case. Coordination between Airworthiness Authorities is advantageous if possible.

FACE UoCs or UoPs should be accompanied with documentation that definitively frames the conditions for their reuse, which should be coordinated in advance with the qualification authority for a system that integrates these components.

6.10 Reuse of Tests

As alluded to previously, reuse of tests can be advantageous for software efforts – having a known set of tests that can be conducted again on code in association with changes, or when it is integrated into new environments. This is applicable to manual or automated test cases.

6.10.1 Reuse of Manual Tests

Quality sets of requirements-based and robustness tests that are well-documented and accepted for qualification establishes repeatable procedures for qualifying software as the life cycle progresses. While the labor of manual tests is not reduced, the existence of well-documented, traceable procedures is beneficial for testing in that these procedures need not be recreated. These tests can also be used as the basis for creating automated tests. Making reusable tests available to subsequent System Integrators is advantageous for inclusion in the system-level tests to ensure that the functionality has not been altered by the integration, and suppliers of FACE UoCs or UoPs should consider this in association with their offerings.

6.10.2 Reuse of Automated Tests

After a proven set of manually conducted tests is established, the efficiency with which those tests can be conducted may be increased by converting the manual tests into automated tests. The initial costs of automation may be significant, and must be analyzed for feasibility. However, consideration of long-term benefits for use in regression testing and for system integration testing should be considered. Automation typically removes the randomness of human-conducted tests and allows verification engineers to converge on determining the continued airworthiness merits of the software.

Developers seeking flight release for their software should remain aware that automated testing must be documented in sufficient detail to enable an evaluator to ascertain that the test sequence properly verifies implementation of the requirements in software. Automated testing has little qualification merit if the test procedure merely lists the title of the automated test script that purports to verify that a specific requirement is properly implemented in code. Technical details must be available to show that the verification is sufficient. This visibility is particularly important in the reuse case, where the tests are also reused.

6.11 Consideration of Archival of Artifacts in the FACE Repository or Reference

Suppliers of FACE UoCs should consider maximizing the availability of qualification artifacts when deciding the terms under which their components will be offered through the FACE Registry.

In order to achieve the FACE goal of software portability through the attributes of modularity, portability, and interoperability the cognizant qualification authority must have access to documentation identified in Section 6.9 above that substantiates achievement of qualification tasks that are useful in evaluating a qualification decision.

The following general categories of artifacts are typically required by an authority responsible for flight release:

- Planning Documents
- Configuration Management/QA Artifacts
- Design and Specifications
- Source and Executable Object Code
- Traceability Artifacts
- Safety Analysis/Hazard Assessment
- Review Artifacts
- Test and Coverage Artifacts (including manual and automated tests)
- Analysis Artifacts

Availability of artifacts and the specifics of their content may significantly impact the cost and schedule associated with reuse of software components and, once again, early coordination of a qualification approach with the integrator’s Airworthiness Authority is advisable to achieve reuse goals.

6.12 Coordination of Qualification

6.12.1 Early Agreement and Documentation Facilitates Efficient Qualification

6.12.1.1 Agreement on Credit Sought

A plan that includes reuse of FACE UoCs should include early coordination with the Airworthiness Authority to achieve consensus as to any qualification credit that will be sought for that reuse, and the documentation that will be required to substantiate that credit. Also, agreement as to the extent of verification that will be necessary for the balance of the qualification is important to achieve efficient qualification and flight release. If no credit is sought, early consensus as to the type, number, and content of artifacts that must be available for review during airworthiness qualification of the resulting system should be established to avoid potential conflicts over documentation requirements for that portion of the software implementation. Description of the credit sought should appear in an Airworthiness Qualification Specification (AQS) or Plan for Software Aspects of Certification (PSAC).

6.12.1.2 Agreement on Completion Tasks

The supplier of FACE UoCs should provide documentation to support the extent to which qualification objectives have been met, and be able to provide artifacts as evidence. Of equal importance is a statement of the qualification tasks that remain to be accomplished by subsequent users, and any definition of those tasks offered. These should be used by integrators when presenting qualification evidence for their composite system software in an SAS, an Airworthiness Qualification Substantiation Report (AQSR), or similar documentation.

6.12.1.3 Documentation of Results

Integrators of FACE UoCs should perform and document the completion tasks as indicated by the component supplier when seeking airworthiness release, and any other qualification tasks identified during the early coordination with their certifying authority.

6.13 Segregation of Software Components

When designing a software architecture based on ARINC 653 partitioning, the engineering trade-offs of timing, resource sharing, latency, etc. should be considered. If substantial variation of criticality exists in significant portions of a software application, or if whole applications having broadly different levels of criticality are to reside on a single processing platform, an analysis should be performed to evaluate whether potential savings in qualification of the software domain having the lower criticality outweigh the expense and complexity associated with the use of an environment that can provide time and space partitioning of the software applications.

Even though qualification at higher levels of rigor can imply significant increase in cost and schedule (possibly amounting to a significant increase in the cost of a project), many factors enter into the decision regarding use of a partitioning operating system. This is an engineering decision that should be based on analysis. Partitioning, in and of itself, does not imply an increase in system safety or reliability, and could result in a decrease in either or both of these if partitioning is not properly implemented (considering balance of resources, interfaces, etc.). On the other hand, partitioning is the only way to segregate software components within a single computing environment in order to achieve qualification at different criticality levels. Without robust time and spatial partitioning, a lower-level criticality application may interfere with a higher-level application.

Memory, hardware resources, and processing requirements for applications that share a computing platform are significant factors in the decision to partition software and in balancing the use of resources if partitioning is implemented.

In order to achieve credit for segregation of software components with regard to qualification at varying levels of criticality, a partitioning system must possess the following key attributes:

- Temporal Separation (schedule control)
- Spatial Separation (memory control)
- Resource Sharing and Separation
- Controlled Interfaces

The mechanisms that supply this segregation must be qualified to the highest level of criticality that applies to any software application they protect.

6.13.1 Segregation for Variability

DoD acquisition of material that incorporates software commonly follows an incremental development process wherein periodic formal releases incorporate change requests and implementation of additional functionality. Given typical release schedules and frequencies, full qualification test of that software is not generally considered necessary. Qualification of incremental releases is typically achieved by regression testing, which should consist of the following tests for all software that executes within the same processing environment (i.e., formal partition):

1. Test of software that implements new functionality
2. Test of software that is impacted by new functionality
3. Test of software that implements requirements that have been identified as safety-related during the SA process

Test of software described by (1) is implicit, because the software has changed. Testing of software of types (2) and (3) is a consequence of the fact that the new/altered software could impact previously qualified software. This is particularly important in the case of software that implements safety-related functionality, which drives qualification of all software executing within a particular computing space (partition or processor) to the level of rigor of equivalent to the level of the most critical software function. Therefore, the extent of regression testing in an incremental software development plan is a cost consideration in architecting a partitioning design.

6.13.2 Segregation for Criticality

6.13.2.1 Multiple Levels of Safety Qualification

Qualification rigor has significant programmatic impact with regard to cost and schedule. Software should be engineered to achieve a proper balance between qualification requirements and the additional complexity and cost of segregation (partitioning) of software components, selection and availability of API calls, and other factors, as these impact the portability of software and, therefore, attainment of the goals of the FACE Reference Architecture. These decisions should be based on evaluation of hazards using a process appropriate to the situation (i.e., joint or service-specific).

6.13.2.2 FACE Profiles and Criticality Level

Software that is evaluated to have no airworthiness qualification requirements is consistent with the definition of RTCA DO-178B Level E. This software may conform to the FACE Technical Standard by meeting the requirements of the FACE General Purpose Profile.

Software evaluated to have any higher criticality should conform to a FACE Safety Profile, requirements for which are defined in the FACE Technical Standard. Meeting any FACE Safety Profile does not imply that any airworthiness qualification requirements are met. These are specified by the cognizant qualification authority. Terms for meeting qualification requirements should be coordinated early with the authority.

The FACE Technical Standard §3.2.1 describes two sub-profiles for the use of the POSIX standard in conformance with the FACE Safety Profile: the Base and Extended Sub-profiles. The Extended Sub-profile is a superset of the Base Sub-profile, and includes POSIX API calls that require special review by a program's safety qualification authority, as the calls may not be considered a candidate for qualification in some contexts. Refer to the FACE Technical Standard §3.2.1 and Appendix A for more specific information before choosing to use the Extended Sub-profile. Additional guidance may be found in Volume 2: Computing Environment. Any component using the API calls specified for any FACE Profile is subject to the qualification requirements established by the cognizant authority. The implementation of the APIs themselves is also subject to qualification.

6.13.2.3 Other FACE OS Interfaces

The FACE Technical Standard includes capability sets for language run-times (C, C++, Ada) and a framework for Java (currently OSGi). These are specified for use in the various FACE Profiles, and, like the ARINC 653 and POSIX interfaces already described, may impact the partitioning strategy in an implementation of the FACE Reference Architecture.

When used with run-times or frameworks as specified in the FACE Technical Standard certain language constructs may not be considered a candidate for airworthiness qualification by particular Airworthiness Authorities. A number of initiatives on defining "safe" language subsets for C++, Ada, and Java have been published and could be used if approved by the Airworthiness Authorities.

6.13.3 Spatial/Temporal Partitioning (ARINC 653)

In order to achieve credit for segregation of software components with regard to qualification at varying levels of criticality, a partitioning system must possess key attributes:

- Time Separation (schedule control)
- Spatial Separation (memory control)
- Resource Separation
- Controlled Interfaces

The OS, hardware, and support elements that provide the formal segregation on which qualification to different levels of criticality is based must be shown to provide this to a level of criticality equivalent to, or greater than, the highest level of criticality of the software functionality it supports. Users of partitioning mechanisms must be aware of the qualification implications of any given configuration.

Selection, implementation, and qualification of the partitioning mechanisms are particularly relevant to achievement of the FACE Safety and Security Profile attributes:

- Non-interference between applications
- Non-interference between applications and the operational environment
- Non-interference between applications and external interfaces
- Software Fault Isolation
- Software Fault Recovery
- Reliable Information Flow between applications
- Reliable Information Flow between applications and external interfaces
- Integrated health monitoring and management
- Standards-based system configuration data
- Non-interference with safety-related functions
- Security Function Isolation
- Security Function Recovery
- Security Function Safe Failure
- Trusted Information Flow between applications
- Trusted Information Flow between applications and external interfaces
- Protection of Data in Processing
- Protection of Data in Transit
- Protection of Data at Rest

6.14 Limiting Qualification Impacts Through Configurability

6.14.1 Safety and Configuration Files (using Deactivated Code)

Software that configures itself based on the contents of a simply formatted, user-editable external file (such as XML), either at start-up or at specific times during operation, allows a system to operate in various configurations which can be qualified as safe and reliable. While this can make software more versatile in various deployments, it leads to special considerations with regard to qualification. Functionality of the software must be proven accurate, deterministic, and robust, at a level commensurate with the functionality that it ultimately controls in the system. Tools that produce and alter configuration may need to be qualified as development tools meaning they will likely carry the same level of assurance as the software on which they operate.

For historical information regarding Civil policy relating to safety-identified requirements, see RTCA/DO-178B §2.4 item e, User Selectable Option Software, and §5.2.3 as a suggestion of the types of considerations an Airworthiness Authority is likely to evaluate. RTCA/DO-178C includes specific objectives regarding the use of parameter data files that are used to modify the run-time behavior of software based on configuration data.

Implications of configurable software require special consideration during safety assessment to ensure that consequences of incorrect operation are considered.

Software that modifies behavior based on an external configuration file must implement that functionality in accordance with stated requirements, and must be verified by test cases that trace to those requirements.

6.14.2 Pre-Qualified Selection of Options at Run-Time (including Initialization)

The concept of deactivated code makes possible the development of software that can be configured in various ways, with code segments remaining unused in varying application environments, hardware/software configurations, missions, etc. This allows software applications to read simply formatted external configuration files to alter their behavior without recompilation from source. When the software is qualified to operate in this manner, the broadest application of software components is facilitated. The best way to avoid cost in the qualification of modified code is to minimize the exposure of changes to the software base.

Another benefit of this strategy is that software fixes in the "common code" are automatically included for every configuration represented, rather than requiring implementation in separate software versions that support various configurations. This single advantage can represent significant programmatic savings with regard to regression testing, although caution is warranted to ensure that changes made to support one configuration do not degrade operation for any other configuration.

6.14.3 Dead Code *versus* Deactivated Code

Most qualification guidance for software prohibits the existence of code segments that implement functionality not specified explicitly by requirements. These code segments are often only found through testing and coverage analysis. These code segments can exist to handle exception conditions or may be present to support a feature that may no longer be used.

Sometimes referred to as "dead code" these code segments represent a risk of reduced safety and reliability in software systems for several reasons:

- If accidentally executed, system operational impact is unpredictable
- Dead code consumes valuable system resources (e.g., memory and its protection mechanisms, data transfer systems, load times)
- Dead code remains untested, since it does not trace to requirements (or, therefore, tests)

Deactivated code is software that is traceable to requirements and, by design, is either not intended to be executed, or is only executable in certain configurations of the current system.

Existence of deactivated code that is not intended for use in the current application is subject to additional qualification activities, including proof of the mechanisms that prevent its execution.

6.14.4 Application of Qualification Requirements to Configuration Mechanisms

A possible drawback of this strategy is additional qualification effort for configurable software. It must be demonstrated that the mechanisms that read and use the configuration information operate reliably, and protect both the used and unused functionality. The software that implements the configuration functionality must trace to detailed requirements and be covered by tests that verify the implementation, and which include robustness testing that verifies correct operation when the content of the configuration file contains errors.

Safeguards must be incorporated when implementing configurable software:

- Verification of proper design of error handling algorithms in software that reads the configuration
- Verification of proper execution of error handling algorithms in accordance with the specified requirements
- Qualification of the system configuration algorithms that use the configuration file content, at a criticality level commensurate with that of the system, and of the functions implemented by the software being configured

Run-time configuration of software requires creation of source code to implement the configuration function. The software design must incorporate these derived requirements, and the software implementation must trace to those requirements and to test sequences that verify correctness.

6.15 Test Programs

Advanced test programs may seek to automate testing, thus making regression testing more efficient, accurate, and possibly applicable to subsequent integrations. Key points to consider are:

- Qualification and configuration management of the test environment is important to ensure confidence in the validity of testing of the software
- Dry run of the test cases before the official run-for-record testing is needed to assure a clean run-for-record with valid, robust test procedures
- Demonstration and documentation of the extent of test coverage of requirements and structural coverage of software elements
- Generation of documentation that clearly and fully exposes the details of testing, necessary for evaluation of sufficiency of the test suite to establish verification – automated test scripts must be fully documented and verifiable
- Accuracy in the test sequence; “pass” without real-time changes during test execution, and without multiple execution of steps is a key aspect of determinism

6.15.1 Requirements-Based Testing

In order to verify implementation of requirements in software, analysis and mapping of requirements to the detailed test cases and the test steps they comprise must be achieved. This allows a qualification consideration to include evaluation of the extent to which the test suite verifies the implementation. The default metric for requirements test coverage is 100%.

6.15.2 Low-Level Software Requirements Testing

High-level requirements are traceable to low-level (detailed implementation) requirements. Low-level requirements testing ensures that the lowest level of operation has been tested to properly implement the low-level requirements. This may require white box testing in which the developer has to connect a debugger to invoke non-reachable states. Also, depending on the level of criticality, the code may require instrumentation to perform structural coverage analysis. DO-178 associates a level of structural coverage with the DAL. The following table summarizes the level of required coverage for DALs C through A. Level D has no structural coverage requirements.

Table 11: DO-178 Design Assurance Levels

DO-178 Design Assurance Level	**Associated Coverage Specified**
DAL C	Statement Coverage
DAL B	Decision-level Coverage
DAL A	Modified Condition Decision Coverage[1]

6.15.3 High-Level Software Requirements Testing

High-level software requirements are derived from system-level requirements. These are typically used to test or demonstrate that the software's functionality meets the intent of the high-level requirements. Usually, this involves testing at the interface level, through black box type testing of the software. Some high-level testing may be achieved via hardware integration testing. Collectively, the various test regimes must add up to a complete qualification suite. Programs may also claim credit for aspects of production acceptance testing, where those test steps verify specific software component functionality sufficiently (subject to approval).

6.15.4 System Integration Testing

System integration testing seeks to verify that all components operate correctly with respect to each other, and so is especially relevant to the incorporation of software components into avionics systems. Implementation of their interfaces, timing, performance, integrity, fault tolerance, robustness, and defined segregation requirements must be verified as correct. It is the responsibility of the integrator to engineer a composite solution that achieves these attributes and provides the required functionality, even if the integration uses software components that have been part of software systems that have previously achieved qualification.

[1] Each condition that has an independent effect on the outcome of a decision must be tested.

6.15.4.1 *Integration of Applications with the Software Operating Environment*

A specialized form of integration testing applies to the interaction of the application software with the infrastructure that supports its execution. Suppliers of software that is part of the support infrastructure should plan and execute an appropriate approach to qualification of these components, including documentation that helps subsequent users engineer solutions based on them, and that supports their efforts to establish qualification. Depending on the requirements of the certification authority, full documentation may be required for these elements. It is unlikely that missing documentation can be reverse-engineered by an integrator, especially in the case of planning documents that are intended to guide the software engineering process.

6.15.4.2 *Integration of Applications with Support Components (Run-Times, Frameworks)*

Software components tested in isolation do not always consider the effects of dependence on other software components. Integration testing is often required to ensure correct operation of applications when supported by elements of the software infrastructure. The more complex these support elements are, the more effort will be required to verify their operation at the level of criticality required by the functions supported. Various analyses may apply, as described in SAE Document ARP4754A/4761, for example.

Not only must the integration of the application with the run-time or framework be verified, but the implementation of the run-time or framework is subject to qualification. While any particular run-time or framework may have an established history of correct operation, the implementation for a particular system (processor, resources, configuration, etc.) has a bearing on qualification, and must be verified for flight release. If qualification artifacts are not available for these specific parts of a FACE Computing Environment, it is unlikely that they can be reverse-engineered. If it is widely acknowledged that a given run-time or framework presents qualification challenges, an integrator must exercise caution when basing a computing architecture on these.

6.15.4.3 *Partial Qualification Credit for Bundled Applications*

The concept of a FACE UoC seeks, in part, to foster integrity by allowing suppliers to combine software support elements with the applications that need them. Presumably, prudent suppliers will perform verification of the interactions between the application and these support elements, and will offer documentation with the UoC that assists integrators with their qualification efforts for the resulting system. To achieve FACE conformance, a UoC must implement, or use, interfaces defined in the FACE Technical Standard for its integration into a FACE Computing Environment; if a supplier provides qualification evidence for the application and for the interface to the FACE Computing Environment, the qualification case for a system having previously qualified FACE interfaces is strengthened. Airworthiness evaluation is likely to include the merits of both the FACE UoC and the FACE Computing Environment supporting it.

6.15.5 Hardware Integration Testing

Software to hardware integration testing is conducted in an environment equivalent to the intended target hardware. While a software vendor creating FACE conformant software may not know the hardware context where their software will be applied, they do know the limitations and resources required by their software. Therefore, they should test the software in some representative hardware environment. The true software to hardware integration is the responsibility of each subsequent integrator of the FACE UoCs.

6.15.6 Configuration and Setup Testing

It is important that the initialization and run-time configuration of software be tested to ensure proper operation. These test cases must appear as part of the qualification test suite.

6.15.6.1 Default Configuration Setup After Power-Up

The requirements should clearly indicate the initial state of all configuration parameters so that a known state is achieved upon power-up. The initial states of the variables should be verified to meet these requirements during qualification testing.

6.15.6.2 Nominal and Out of Boundary Configuration Verification (Robustness)

Any configuration of parameters within the valid range for each parameter should be checked. Also, values outside the range should be tested to ensure proper handling by the software.

6.15.7 Data Parameter Configuration Testing

External database load files that determine the run-time configuration of the software should be tested within bounds and out of bounds.

6.15.8 Information Interface Testing

All interfaces should be well-defined in the requirements with their proper protocol, data types, and little/big-endian configuration. Also, data integrity and checking of the data transferred by the interface should be tested.

6.15.9 Type Consistency Testing

The data that is exchanged through an API or communication service should be tested for type consistency (booleans, integers, reals, etc.). Moreover, robustness testing must be included to check for proper response per requirements as well as system response when type inconsistencies occur.

6.15.10 Protocol Header Testing

The protocols used on each interface should be clearly defined on the interface and tested for nominal and off-nominal conditions.

6.15.11 Data Integrity Testing

The data that is exchanged on an interface should be tested for integrity by an application through some defined mechanism (e.g., Cyclical Redundancy Checking (CRC), Error Checking and Correction (ECC), checksum). Nominal and corrupted non-nominal testing should occur to detect proper data flow operation. Sequencing of data should also be checked with out of order, delayed, and variable delayed conditions tested as robustness test cases.

6.15.12 Robustness Testing

Qualification testing must include not only normative test cases that show correctness in the expected realm of operation, but also test cases that verify a software system's response to out-of-bounds conditions. Due to challenges relating to the qualification of complex software, an argument cannot be made that any given software component will only receive the expected range of inputs, nor can it be stated that operation outside the designed operating space will not occur, without definitive proof. Testing must therefore expand beyond normal cases, especially for software that implements functionality evaluated to have a high level of criticality.

6.15.12.1 *Boundary Conditions*

Inability to exhaustively test the operation of software elements where ranges of input values may occur can be mitigated by selecting test cases where values both just inside, just outside, and exactly on the defined boundaries demonstrate correct operation. Intelligent selection of other input values can provide verification evidence without exhaustive testing; the supplier or integrator should document the strategy used.

6.15.12.2 *Error Checking*

In addition to boundary value checking, error checking provides demonstration of robustness. Examples include properly formatted configuration files, error messages, data exchanges, checksum/CRC failures, etc. This will be particularly true with regard to configuration file content where modes of operation are based on the values read; errors in format and range should be verified for proper system response, per specific requirements (presuming that derived requirements were formulated to define this functionality).

6.15.12.3 *Exception Processing (Health Monitoring/Fault Management (HMFM))*

The FACE Technical Standard requires health monitoring and fault management at the OS level and at higher levels. Verification of the operation of these strengthen the qualification case for software systems seeking flight release, and may offer mitigations that increase the calculated system reliability, depending on their coverage and verification. Suppliers should document these features and the associated verifications as part of the artifact set for consideration of qualification.

The following tests should be considered in testing:

- **Load testing**

  The behavior of software in a system with a nominal loading of processor, memory, or data bus is expected to be tested for verification. It is when the component or system is under a load that anomalous behavior may occur.

- **Duration testing**

  Long duration testing of software can be useful in determining whether memory leaks or file mismanagement generate errors over time. It is advisable that automated testing be employed to stress the system (load test) be conducted over a sufficient duration to reveal effects over time.

- **Memory leak testing**

  Conducting tests to cause the system to make multiple memory allocations is important to ensure that memory is properly managed. It is encouraged that testing be automated to cause software to generate latent errors that might manifest after multiple allocations, extensive cache accesses, etc. If available, automated unit tests can snapshot the amount of available memory using OS services not defined by the FACE Technical Standard. It is then necessary to ensure that the memory available does not change during run-time of the tests.

- **Fixed/variable delay (jitter) testing**

  Delaying information at an interface can be detrimental to operation of a software component or system. Varying this delay can produce additional errors in some applications such as data-intensive real-time video or voice operations, for example.

- **Configuration testing**

  This type of testing seeks to determine the behavior using illegal input values. User configuration through a user interface or through an external configuration parameter file should be performed under test conditions to ensure that proper error checking is done to prevent anomalous operation. System response to these conditions should be verified to be conformant to the stated requirements.

6.16 Life Cycle Support

Given a current definition of "complex" software – e.g., that for which each and every execution path and range of input values cannot be exhaustively tested – the likelihood that fielded software incorporates problems is high. While verification programs seek to ensure functionality and safety, the focus must be on areas of high criticality, and those seeking qualification must acknowledge that complete coverage cannot be achieved through exhaustive testing. Since problems are likely to surface after deployment, military programs must plan and execute procedures for problem reporting, tracking, and correction that address the requirements of the Airworthiness Authorities, security, and satisfy the conditions for continued airworthiness.

6.16.1 Distribution of Software Maintenance Releases

The life cycle support picture becomes complicated when portable software components are part of systems that are developed and deployed by multiple integrators. It is necessary to have a plan that maintains the acceptable level of safety established during the qualification of a system that incorporates FACE UoCs or UoPs. The original supplier of a UoC or UoP could act as the hub of a network of users that gathers problem reports, generates or receives "fixes", and distributes corrected software to all users. This parallels the guidance stated for Civil software reuse (Advisory Circular 20-148). An example of this is where suppliers of OS software accept input and provide updates. For suppliers of FACE UoPs or UoCs, life cycle support may involve contractual complications that must be solved, but ultimately it will be up to the integrators of FACE UoPs or UoCs to address this aspect of system qualification with their respective Airworthiness Authorities.

6.16.2 Collection of Problem Reports

The success of life cycle software support is based on the collection of problem information from fielded systems. Whether formal or informal, this process is simplified when software is deployed within an organization, and is more complex in the context of systems comprising FACE UoPs or UoCs. Both suppliers and consumers of FACE UoPs or UoCs should address the issue of feedback as part of the engineering process to maximize efficiency in this area.

6.16.3 Dissemination of Problem Reports

A general problem with reused software components, whether they be open source or proprietary, is how to triage and disposition software defects. Not all defects impact all users in the same way. This problem will still exist for integrators of FACE UoPs or UoCs. In addition to distribution of software updates, a process should be established by which notification of problem reports takes place rapidly and pervasively, especially for software issues that have been shown by safety analysis to impact system safety and security. Application of FACE UoPs or UoCs in multiple integrations and various contexts makes distribution of problem reports both more complicated and more important, and a formal statement of a support plan should represent a distribution advantage for FACE suppliers.

6.16.4 Software Loading Instructions

The integrator is responsible for integrating FACE UoCs and setting up the software. Guidelines provided with the software ensure a successful integration.

6.16.5 Software Compilation Instructions

Any special tools, preprocessing, or compilation instructions should be clearly spelled out in documentation accompanying software components. Any advice concerning optimization parameters or particulars of the compiler, target processor environment, and memory requirements should be provided.

6.16.6 Software User's Manual

All of the instructions for software loading, compilation, and any other environmental aspects should be contained in a Software User's Manual.

6.17 Documentation

Qualification artifacts are critical to the evaluation of airworthiness, and cognizant Airworthiness Authorities are likely to base their evaluation on those available for FACE UoCs within avionics systems based on software. Suppliers should evaluate the merits of making these artifacts available to System Integrators of their products to support qualification. For integrators of FACE UoPs or UoCs these artifacts will be key to qualification of systems that incorporate them. Existence of artifacts may be listed in the FACE Registry or available directly from a Software Supplier.

These general categories and types of artifacts are typically required by an authority responsible for flight releases:

- Technical Planning Documents:
  - — Design and Specification Documents
  - — Requirements
- Programmatic Planning Documents:
  - — Process Descriptions
  - — Configuration Management Plans
  - — Quality Assurance Plans
  - — Reviews

- Safety Analyses:

  — Hazard Analyses

  — Functional Analyses

  — Failure Mode Effects and Criticality

- Test Artifacts:

  — Test Procedures/Plans

  — Test Results

  — Robustness Verification and Documentation of Supporting Elements

  — Test Coverage Reporting

  — Traceability

More information on the specifics of the artifact listing and content is available in MIL-STD-498 (cancelled 1998), MIL-HDBK-516, and RTCA DO-178B/C, et al. Suppliers and integrators of FACE UoPs or UoCs should refer to the appropriate Airworthiness Authorities for definitive guidance for their particular situations.

6.18 Model-Based Engineering of Software

The use of Model-Based Engineering (MBE) potentially enhances the integrity and safety of a software product, as well as presenting opportunities to improve efficiency in the areas of development and qualification. Other benefits such as succinct communication of requirements and verification of design before implementation may be available as well. Successful approaches will carefully consider the benefits of a tool chain that facilitates each phase of the software engineering process from requirements management to model development, to generation of source code, and formulation of a set of tests that verify the implementation. Effective tools will produce output that is useful for evaluation of qualification, including traceability data that substantiates the integrity of the system, and coverage of the test suite.

In cases where qualification credit for an application is sought due to the use of software tools, such tools must be qualified according to the level of rigor associated with the software functions they are used to implement or verify. Many tools have not been qualified. Those that have been qualified must still be proven to operate effectively for the development under consideration, and so users of these tools should plan for additional effort. To aid in this, the tool supplier may offer a set of resources to support qualification of the tools for a particular use.

In cases where no qualification credit is sought due to the use of specific tools, the tool set should be planned and used in such a way that typical qualification artifacts will be produced that support evaluation of qualification according to the appropriate established processes. The development approach should be accurately documented so that a qualification evaluation can consider the merits of the tool chain and the documentation produced.

6.19 Special Considerations for the Use of Run-Times and Frameworks

6.19.1 Language Run-Times

Like any other software components that comprise a complex avionic system, language run-times that are part of a qualification consideration are subject to the terms of qualification as determined by the cognizant Airworthiness Authority.

Refer to the FACE Technical Standard §3.2.3 for detailed information regarding the use of language run-times within the FACE Reference Architecture.

Definition of a subset of APIs does not constitute qualification, which includes verification of the implementation of those functions on the particular processing engine/electronic hardware, and within the software environment, that supports execution of an application that relies on them. Verification of the APIs' executable code is required to establish the software quality attributes that are defined by the Airworthiness Authority (i.e., MIL-HDBK-516 or DO-178).

Credit for service history must be based on documented service records showing that the software possesses the necessary quality attributes, and evidence supporting the claim that the operating environment for the system seeking certification is equivalent to the environment(s) on which the service history was established.

It is commonplace that restrictions have been defined for the use of language run-times that improve the qualification picture by seeking to limit violation of the FACE Technical Standard. Applicability of these restrictions should be coordinated early with the Airworthiness Authority prior to implementation or integration of existing software components to avoid expensive and time-consuming re-qualifications. Use of some languages and their run-time libraries without these restrictions may make qualification problematic.

Implementations that disregard these limiting definitions should carefully evaluate this strategy before placing limitations on the applicability of their products.

6.19.2 Frameworks

The FACE Technical Standard allows frameworks as part of the Operating System Segment of a FACE conformant UoC.

For a description of the FACE concepts that apply to use of frameworks, refer to the FACE Technical Standard §3.2.4. In short, the distinction between language run-times and frameworks is that run-time libraries are called (and instantiated) by the applications they support, but a framework calls the applications as the arbitrator of execution.

Within the definition of the FACE Technical Standard use of frameworks in both the General Purpose and Safety Profiles consists of framework features described in OSGi Service Platform Release 6 Core for Java-based systems In addition to qualification of application software that provides aviation-specific functionality, military acquisitions require qualification of the software infrastructure that supports those applications. A framework that arbitrates execution of avionics applications is of particular interest to an airworthiness qualification agency, since this software directly affects the areas listed in Chapter 6, and other concerns relevant to integrity, determinism, robustness, etc. as described extensively herein. Depending on the evaluated criticality of the functions that are implemented, qualification of software may require demonstration of determinism by complete decision path testing, with documented evidence of coverage. This testing and analysis represents a significant investment in time and labor, and should be carefully considered when an architecture is engineered to include a framework. Once the implementation of such an infrastructure is qualified and documented, it may be possible to leverage that qualification for future systems, provided the hardware and software environment can be proven to be exactly equivalent, and documentation is provided. As with any software, a qualification authority is likely to require proof of correct operation of any new integration, as well, including proof of correct functionality, hard real-time scheduling where required, singularity of outcome (determinism), robustness (edge conditions), fault-tolerance, and other characteristics as required by the certification authority.

Frameworks that arbitrate the execution of applications, or bundles of applications, must be proven to provide the separation necessary to ensure their correct and timely operation, especially frameworks that seek to build systems that emerge from joining different components that have no pre-existing knowledge of each other, and systems which arise dynamically by combining the available applications and services as needed. Further attention must be applied to frameworks that arbitrate the provision of services from one application or group to another, and removes them asynchronously, requiring the consumers of those services to adjust their operation. The goal of integration testing in military avionics qualification programs is to ensure that systems comprising various applications function correctly in every operant configuration; therefore, dynamic configuration of software components in avionics is only acceptable if every configuration is qualified, and every schedule of dynamic configuration is tested and proven to meet real-time determinism and reliability. The mechanisms that provide this functionality are also subject to qualification.

An Airworthiness Authority will seldom base a qualification decision solely on the fact that a given software component has been previously used, is said to have been qualified elsewhere, or has an established service record, without artifacts that document its quality, and the conditions under which these achievements were attained. Equivalence of the conditions of application (including operating environment), as well as documented service history, will be required by a certification authority to consider qualification credit for previous use *in lieu* of other artifacts; early coordination with the appropriate agency is advisable to establish consensus as to the plan for certification.

6.20 Conclusion

Engineering of software components for reuse represents enhanced market availability. Reuse of such software components presents opportunities for savings in terms of cost and schedule. Each strategy, however, has some associated cost and schedule implications of its own to consider.

All parts of a software product are subject to qualification, and an integration program that incorporates existing software components will at least be required to verify correct integration and operation of that system, and at most will be required to re-qualify those reused components, especially if the components are not properly documented.

Execution of a safety program is required, including analyses to establish the criticality of all software that comprises an avionic system, which then defines the level of rigor to which that software must be qualified.

Review of documentation of the software engineering effort by an Airworthiness Authority is key to a qualification evaluation for flight release. Documentation should journal the work, not historically report its accomplishments.

Early coordination with the cognizant Airworthiness Authority is advisable in any development or acquisition of avionics where flight release is required, so that an agreeable course is taken to substantiate a case for qualification. This will result in the most efficient qualification possible.

Software without life cycle support is not effective in the long term.

7 Security Guidance

The following sections include guidance, clarification, and examples of security considerations associated with implementing the FACE Technical Standard. While this information is targeted to all the security stakeholders, it is not intended to address all potential conformant implementations of the Security Profile.

7.1 Introduction

Security is a core consideration in developing robust avionics systems. While security enforcing functions can span all segments of the FACE Reference Architecture, a large portion of the critical security functions reside in the Operating System Segment and one or more high-assurance partitions per processor core (e.g., data flow policy enforcement and time/space isolation). The FACE Security Profile provides a minimal API subset that facilitates evaluation to a high assurance level.

While the Security Profile can provide a framework for all assurance levels, given the restrictions of the available APIs it is intended to be used primarily for high-assurance components that provide security enforcing functions. Other FACE Profiles may provide similar assurance but a separate Security Profile is needed to enable systems that require a minimal API subset for security analysis and certification purposes. For systems with lower security assurance requirements, it is possible to use either of the other FACE Profiles as long as the associated process and assurance requirements are satisfied.

It is important to reiterate that conformance to the Security Profile APIs does not guarantee a FACE UoC is certifiable to any security standard. Depending on the target assurance level and certification process for the air platform as defined by the Authorizing Official, the design may not support the security requirements and subsequently may need modifications. Additional design artifacts may be needed to achieve certification.

The certification process and target assurance levels will vary between air platforms whereas the FACE Technical Standard provides a common architectural framework that can be leveraged to help lower the life cycle costs associated with security certification. While there will always be costs associated with certifying a system for each air platform, the FACE Technical Standard helps minimize this by enabling software reuse and portability between platforms. Additionally, this architecture minimizes the impact of hardware obsolescence while providing competitive life cycle costs without sacrificing security. Because FACE requirements do not explicitly specify hardware components, it is the responsibility of the System Integrator to ensure appropriate hardware is used to support the security controls of the composed system including partitions, if any.

7.1.1 Basic Security

The FACE Security Profile is focused on security-relevant software components targeted for high-assurance designs. However, all software components in a secure system should meet a basic level of security assurance unless they can be explicitly shown to not interfere with or compromise the correct operation of system security. The following is a minimum set of data that should be generated for each software component that may support or contribute to the system security. Since one of the primary goals of the FACE Technical Standard is to enable portability of software between computing platforms, the following items should be applied to all UoCs if they are to be reused in a secure environment:

- Functional requirements associated with the software component
- Description of all data flows into and out of the software component
- Description of the computing resources required for the software component to operate
- Description of design and test artifacts for the software component
- Description of software component development history

7.1.2 Security Principles

This section provides information and guidance on the mapping between the FACE Reference Architecture and the security principles as described in Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems (NIST SP 800-160), Appendix F. This may be useful in designing secure systems and FACE UoCs.

Some principles are inherent in the FACE Reference Architecture as described by the FACE Technical Standard. Other principles require additional effort whether by the system design or implementation of a FACE UoC. A summary of all the principles is given in Table 12 along with guidance on whether the principle is considered inherent, enabled by or supported by the FACE Technical Standard, with clarifications as needed for the UoC and the higher-level system design implementations. Additional details on each principle are provided in subsequent sections.

Table 12: Security Principles Inherent to the FACE Reference Architecture

Principle	**FACE Technical Standard Support**
Clear Abstractions	Inherent
Least Common Mechanism	Inherent
Modularity and Layering	Inherent
Partially Ordered Dependencies (POD)	Inherent
Efficiently Mediated Access	Inherent
Minimized Sharing	Inherent
Reduced Complexity	Inherent
Secure Evolvability	Enables
Trusted Components	Limited to the OSS APIs for the Security Profile
Hierarchical Trust	Not inherent

Principle	**FACE Technical Standard Support**
Inverse Modification Threshold	Limited to the OSS APIs for the Security Profile
Hierarchical Protection	Limited to the OSS APIs for the Security Profile
Minimized Security Elements	Not inherent
Least Privilege	Not inherent
Predicate Permission	Not inherent
Self-Reliant Trustworthiness	Not inherent
Secure Distributed Composition	Not inherent
Trusted Communication Channels	Not inherent

The following discusses the relevance of these principles for implementations of the FACE Technical Standard. Italicized text is paraphrased from NIST SP 800-160 Appendix F, with additional descriptions added for further clarification. Implementation of a system should consider these principles.

Clear Abstractions

The principle of clear abstractions states that a system should have simple, well-defined interfaces and functions that provide a consistent and intuitive view of the data and how it is managed.

The FACE Technical Standard establishes clear and well-defined interfaces for each segment. All data exchanged between UoCs using the TSS must be modeled in accordance with the FACE Data Architecture.

Least Common Mechanism

The principle of least common mechanism states that, if multiple components in the system require the same function or mechanism, the function or mechanism should be factored into a single mechanism that can be used by all of them.

The FACE Technical Standard enables the construction of components that provide common services.

Modularity and Layering

The principles of modularity and layering are fundamental across system engineering disciplines. Modularity and layering derived from functional decomposition are effective in managing system complexity, by making it possible to comprehend the structure of the system.

The FACE Technical Standard decomposes the system into five major segments: OSS, IOSS, PSSS, TSS, and PCS. The standard defines the functional scope of each segment.

Partially Ordered Dependencies

The principle of partially ordered dependencies states that the calling, synchronization, and other dependencies in the system should be partially ordered. A fundamental concept in system design is layering, whereby the system is organized into well-defined, functionally related modules or components.

The FACE Technical Standard adheres to a strict modular and layered approach with a hierarchical layering from OSS, IOSS, PSSS, TSS, and PCS. The standard clearly defines the system components and anticipated functionality.

Efficiently Mediated Access

The principle of efficiently mediated access states that policy enforcement mechanisms should utilize the least common mechanism available while satisfying stakeholder requirements within expressed constraints.

IOSS mediates all access to the hardware device drivers; OSS mediates all access to CPU, memory, and other processor resources. A properly constructed system maintains this access and does not violate this principle.

Minimized Sharing

The principle of minimized sharing states that no computer resource should be shared between system components (e.g., subjects, processes, functions) unless it is absolutely necessary to do so.

The FACE Safety and the Security Profiles enable mediating access to the Memory Management Unit (MMU) and memory resources such that a UoC only accesses its allocated resources in its partition. This mechanism is provided at the lowest layer in the system within the OSS that controls all access to the MMU and system processor memory resources. This preserves separation of the resources, and prevents unauthorized access, disclosure, use, or modification by another mechanism from another partition. Care is required to avoid undesirable performance impacts and to understand potential covert timing and storage channels due to misuse of time and space partitioning.

Reduced Complexity

The principle of reduced complexity states that the system design should be as simple and small as possible. A small and simple design will be more understandable, more analyzable, and less prone to error. This principle applies to any aspect of a system, but it has particular importance for security due to the various analyses performed to obtain evidence about the emergent security property of the system.

The FACE Security Profile is based on the Security Kernel concept that follows the NEAT principle (i.e., Non-bypassable, Evaluatable, Always enabled, and Tamper resistant). It contains the least amount of software that has been purposefully developed such that it can be fully evaluated and contain the fewest vulnerabilities. A system which implements a security policy, that requires similar high assurances, should apply the same concept of reduced complexity in its architecture and construction of its UoCs.

Secure Evolvability

The principle of secure evolvability states that a system should be developed to facilitate the maintenance of its security properties when there are changes to its functionality structure, interfaces, and interconnections (i.e., system architecture) or its functionality configuration (i.e., security policy enforcement).

While the concept of a FACE UoC and modularity enables the replacement with a revised component that can be verified independently of the rest of the system, the ability to securely upgrade the system may be dependent upon other design principles herein: Modularity and Layering, Efficiently Mediated Access, Minimized Sharing, Hierarchical Trust, and Reduced Complexity.

Trusted Components

The principle of trusted components states that a component must be trustworthy to at least a level commensurate with the security dependencies it supports (i.e., how much it is trusted to perform its security functions by other components). This principle enables the composition of components such that trustworthiness is not inadvertently diminished and where consequently the trust is not misplaced.

An implementation of a trusted system must consider the trustworthiness of all the UoCs that implement a security function to a level commensurate with the security dependencies it supports. The architecture of the system will require careful placement of the security function and its dependencies on the other UoCs in the system. In addition, adherence to the following principles will help achieve the goals of a trusted implementation: Modularity and Layering, Efficiently Mediated Access, Minimized Sharing, Hierarchical Trust, and Reduced Complexity.

Hierarchical Trust

The principle of hierarchical trust for components builds on the principle of trusted components and states that the security dependencies in a system will form a partial ordering if they preserve the principle of trusted components. The partial ordering provides the basis for trustworthiness reasoning when composing a secure system from heterogeneously trustworthy components.

An implementation of a trusted system must consider the hierarchical dependencies of trust. For the system to achieve trustworthiness of a security function, it should build upon trusted layers. A trusted UoC may only be as trusted as its foundational UoCs.

Inverse Modification Threshold

The principle of inverse modification threshold builds on the principle of trusted components and the principle of hierarchical trust, and states that the degree of protection provided to a component must be commensurate with its trustworthiness.

Trusted UoCs should be protected from unauthorized modification with sufficient level of measure such that the security function of the trusted UoC cannot be bypassed. Care must be taken that the UoC cannot be modified by a less trusted UoC or from unauthorized access via its API.

One example of the potential for an API that may allow for unauthorized access is the LCM interfaces that can change the state of a UoC. All UoCs that have access to this API are presumed to be authorized to modify the LCM behavior of the UoC.

Another example of unauthorized modification (including replacement) is the modification of the loaded UoC (binary).

Other unauthorized access of a UoC include direct access to the UoC's memory and resources.

Hierarchical Protection

The principle of hierarchical protection states that a component need not be protected from more trustworthy components. In the degenerate case of the most trusted component, it must protect itself from all other components.

The FACE Security Profile includes an OSS that uses processor control mechanisms to restrict FACE UoCs to their required computing resources. Likewise, the other UoC's trustworthiness needs to be considered in the use of partitioning that leverages the OSS protections. UoCs in a partition may be only as trustworthy as the lowest trustworthy UoC.

Minimized Security Elements

The principle of minimized security elements states that the system should not have extraneous trusted components. This principle has two aspects: the overall cost of security analysis and the complexity of security analysis.

Trusted components are costly and resource (time and skills)-intensive to develop, procure, and maintain. Care should be taken to minimize the security component size and number of components in a system. This may require cost tradeoffs during the architecture and design phase for the system.

Least Privilege

The principle of least privilege states that each component should be allocated sufficient privileges to accomplish its specified functions, but no more.

Least privilege includes access privileges for an interface and its underlying functionality. Privileges to functionality may be limited to each interface (similar to user *versus* administrative, for instance). The FACE Technical Standard does not inherently provide least privilege mechanisms, but does allow a variety of implementations that can be used to ensure appropriate privileges. Specific to the FACE Security Profile OSS APIs, there are permission restrictions that can be used to implement some aspects of least privilege, but it is the responsibility of the integrator to configure them appropriately. For example, an audit function would require least privilege for the interfaces described in Table 13. UoCs that create or write audit events would have a different level of privilege from UoCs that can read audit events or even modify.

Table 13: Implement System Interfaces with Least Privilege

Interface	Purpose	Access Privilege
System Events (Input)	Capture events	Create/Write

Interface	**Purpose**	**Access Privilege**
Audit Log (Output)	Retrieve/Review Audits	Read
Clear Log	Sanitize the Audit Log	Modify/Read/Verify

Predicate Permission

The principle of predicate permission states that system designers should consider requiring multiple authorized entities to provide consent before a highly critical operation or access to highly sensitive data, information, or resources can proceed.

Predicate permission may be needed in a system to address:

- Hazardous conditions, firing of ordnance (launch/drop weapons)
- Irrevocable security decisions including zeroization of the crypto material or operational flight programs
- Software loading to ensure the system is in a valid safe state to accept a configuration change

FACE UoCs and interfaces should consider separation of the functionality and interfaces that implement predicate permissions.

Self-Reliant Trustworthiness

The principle of self-reliant trustworthiness states that systems should minimize their reliance on other systems for their own trustworthiness.

This principle provides guidance that when a UoC is dependent on another it should retain its trustworthiness. This may include the need for a UoC to protect itself from the UoCs with which it interfaces when it is interfaced with less trustworthy UoCs.

Secure Distributed Composition

The principle of secure distributed composition states that the composition of distributed components that enforce the same security policy should result in a system that enforces that policy at least as well as the individual components do.

Security policy must be considered in the context of the system and should include a system-level architecture to ensure the security policy is properly implemented, not bypassed, and universally applied across all utilized UoCs.

Trusted Communication Channels

The principle of trusted communication channels states that when composing a system where there is a potential threat to communications between components (i.e., the interconnections between components), each communication channel must be trustworthy to a level commensurate with the security dependencies it supports (i.e., how much it is trusted by other components to perform its security functions).

Trusted communications may be needed in a system to provide privilege access to information and/or the system. Trusted communication channels may be protected to:

- Prevent disclosure of the information transmitted:

  — Through isolation from untrusted communications channels

  — Encryption of the communication channel over untrusted infrastructure or channels

- Prevent denial of the information transmitted:

  — Protect the channel from being blocked, halted, or disrupted

- Prevent modification of the information transmitted such as through authentication and digital signatures

It is the responsibility of the System Integrator to determine the implementation of trusted communication channels and properly select the UoCs. It some cases it may be appropriate to utilize a FACE Security Transformation to provide protection of the communication channel.

7.2 System Security Architecture

Security is not an afterthought. Failure to understand and define the security functions of a system will lead to a wide range of issues that may pose significant if not impossible barriers to security certification. While the specifics will vary between implementations, documenting the security architecture for each security function is a critical process step common for all secure systems.

As per the FACE Technical Standard, the FACE Security Profile provides a wide range of security-enabling functions and attributes. These functions and attributes provide support for implementations of single and multi-level security functions, protecting data in process, transit, and at rest, isolation, failure recovery and reporting, and trusted information flows while guaranteeing non-interference between the portable software components and the operating environment and external interfaces. To reinforce a statement from the FACE Technical Standard, these enablers do not in themselves satisfy the system-level characteristics of Confidentiality, Integrity, and Availability. However, they do provide the necessary framework to achieve these.

As stated previously, the Security Profile APIs do not inherently make a secure system; additional data and context must be provided. Security architecture will be a significant aspect to addressing the desired security properties of the system. NIST SP 800-160 addresses a systems security engineering approach in the development of the secure system.

7.3 Process

7.3.1 Risk Management

Security risk management is the responsibility of each aircraft platform to use the applicable risk tool to address risk/issues and opportunities which cover security-specific topics. Each OEM will have the responsibility to analyze the subsystem architecture design with the Aircraft Platform System Architecture and Security Engineering team.

7.3.2 Software Assurance

The CNSS 4009 IA Glossary defines software assurance as: "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner". Software assurance is attained through a combination of activities performed during development, procurement, implementation, verification, test, and maintenance. Ultimately, the goal of software assurance for a FACE UoC is to reduce security flaws and defects in the component's software through consistent application of diagnostic tools, proper process, and education of stakeholders for proper implementation of assurance plans.

Software assurance of FACE UoCs will be a part of the overall system security assurance. As systems are integrated, components and FACE UoCs are integrated into air platforms. These integrated systems are subject to security certification processes defined for the air platform and are not addressed specifically by the FACE Technical Standard.

7.3.3 Security Controls

Understanding the platform security requirements is a critical first step for avionics systems design. While each platform has unique security requirements, they can be ultimately traced back to a set or catalog of security controls. The set of applicable security controls for a platform is typically tailored for specific types of functions, technologies, and operational environments. It is the responsibility of the platform to allocate the required set of security controls to the FACE Computing Environment and associated UoCs based on their implementation.

Prior to the release of the FACE Technical Standard, Edition 3.0, an informal evaluation of the FACE Reference Architecture was performed by the FACE Consortium Technical Working Group (TWG). The purpose of this evaluation was to identify potential impacts to platform security certification requirements and help validate the robustness of the FACE Technical Standard. The security control analysis did not take into account network communications. The informal evaluation was executed using the security control catalog from NIST SP 800-53: Security and Privacy Controls for Federal Information Systems and Organizations. NIST SP 800-53 was selected given its range of applicability across a variety of communities of interest and relevancy to both government and commercial efforts. The catalog of security controls was tailored based on the applicability to avionics information systems which was further reviewed against DoD 8500.2 and CNNSI 1253. Each identified control was mapped to FACE Segments where the control could be implemented and evaluated for impacts to segment requirements or profile characteristics. While not all targeted controls are explicitly supported, the results of this informal evaluation helped validate that the FACE Technical Standard can enable implementation of a wide range of security-relevant functions.

One example of a NIST SP 800-53 control that was evaluated was SC-8 Transmission Confidentiality and Integrity from the System and Communications Protection (SC) family. To enable support for the Cryptographic or Alternate Physical Protection enhancement SC-8(1), the FACE Technical Standard provides a Security Transformation capability. A Security Transformation can be used in multiple segments (i.e., PCS, PSSS, and TSS UoCs) to implement encryption or related functions to protect data confidentiality and integrity during the transmission process. A more detailed description of the Security Transformation is included in Section 7.5.

A security control may also apply to the air platform, such as AC-4 Information Flow Enforcement, that spans multiple segment layers. The elaboration of a specific control would then create security requirements for one or more FACE UoCs. When security evaluators determine whether a given control is satisfied, they would then use the aggregation of evidence provided by each of the UoCs to satisfy the control requirements.

SC-27 Platform-Independent Applications embodies one of the key tenets of the FACE Technical Standard: the separation of UoCs from the platform into which they are integrated.

7.4 Security Design Considerations

To support the concept of isolation where security functions are separate from non-security functions, it is recommended that a set of core security services be provided as common services. These common services should be accessible by applicable UoCs to maximize reuse and portability. These common services should include core security functions such as Audit Logging, Authentication, Access Authorization, Certificate Management, Cryptography, Key Management, Labeling, and Tagging.

7.4.1 Challenges of Reusing UoCs

There are many security challenges to reusing UoCs. It is recommended that analysis occurs to identify common security functions, attributes, and architectures of the target environments for which reuse is being targeted. This may result in varying security controls for a single UoC integrated into different environments. One method for addressing these variation points is through the FACE Configuration Services. Another method would be to develop UoCs for targeted subsets of the environments.

Another challenge would be if a UoC interface contains security properties that are not well-defined and understood. Reusing in a new environment could introduce a new vulnerability that would require full system testing to uncover. The FACE Data Architecture can be used to precisely define the interface to aid in reuse and reduces some potential vulnerabilities. This does not automatically mean the UoC can be reused in another security context without an assessment to show its applicability.

It is the platform integrator’s responsibility to determine the viability of reusing UoCs.

7.4.2 Verification of System Configuration

FACE Configuration Services enable adaption of FACE UoCs for different computing environments. This service requires verification of the configuration data. This verification is not capable of providing end-to-end assurance that the configuration data is from an authorized source and has not been modified. A more complete set of checks are required such as:

- Trusted Delivery
- Trusted Data Load
- Trusted Startup
- Trusted Boot
- Attestation

The design of these checks is implementation-dependent. A FACE UoC could satisfy some of these checks. These types of trusted solutions should be developed with the end application in mind. Some end applications will require a security certification. Experience in security certification and guidance from the Authorizing Official is invaluable. Well-recognized best practices, such as appropriate use of cryptography and Public Key Infrastructure (PKI), can ease these hurtles.

7.4.3 Auditing

To meet the goal of reusability UoCs should utilize common services. One of the services critical in the support of security is an Audit Log. For that reason, we will identify the characteristics of a notional Audit Log as a model for its implementation. An Audit Log is capable of capturing security events in addition to relevant real-time system states, modes, messages, and data flow for future analysis. To support an effective analysis, all records should be time-stamped with an accurate system date and time. The log can provide an Auditor or Security Officer with the information needed to determine the behavior of the system relevant to any number of security-related concerns. The Audit Log could be used to determine if the system has been exposed to a data breach or it could support the documentation required in an official IA audit. The Audit Log is a data source which must be protected. Encryption/decryption for data at rest may be needed to protect persistent data storage of the security log. Likewise, encryption/decryption protections for data in transit may also be required. When the data is no longer needed, a sanitization function may be required.

The FACE Technical Standard, like most existing deployed systems, does not provide a single interface that explicitly meets all the requirements to satisfy the Audit family of controls in NIST SP 800-53. The existing FACE interfaces can be used to develop custom auditing solutions for individual air systems, but those solutions could be overly complex and limit reuse. The FACE Technical Standard interfaces were reviewed as candidates and their initial assessments are briefly described below.

The TSS supports many of the Audit controls. It provides well-defined and constrained interfaces to almost all FACE Segments and can implement one or more Security Transformations to protect the data and provide appropriate access control. However, there is no standardized FACE approach to logging, the TSS does not have direct interfaces to the IOSS, and there is a lack of file system support in the FACE Security Profile. Each of these challenges could be overcome by the implementer.

The HMFM interface also satisfies some of the controls associated with audit logging. HMFM has interactions with all segments, and provides a significant amount of oversight and reporting via ARINC 653. It has the same logging limitations as TSS and lacks access control protections for audit information (e.g., the optional REPORT_APPLICATION_MESSAGE from ARINC 653 is processed by a single handler and is implementation-specific in its payload structure). Furthermore, HMFM is excluded from directly utilizing Security Transformations which may limit data protections. Similar to TSS, these challenges could be overcome depending on the platform security environment.

Since neither approach is complete, a more robust capability will be explored in a future FACE Technical Standard edition. Until then, each platform integrator must work with the UoC supplier(s) to implement audit logging as required to meet the end system needs.

7.4.4 Privileged Processes

The term “privileged process” generally refers to a software thread of execution with access rights that permit it to have expanded system-wide effects. FACE UoCs do not typically require system-wide effect to meet their requirements and few FACE interfaces reflect the concept of "privilege" or "permission". By the security principle of Least Privilege, FACE UoCs should therefore not typically be privileged processes.

Privileged processes are a very useful construct in designing a system software architecture. They can be "high-level", such as an application, or "low-level", such as a device driver. It is not the process' place in the software stack *per se* that makes it "privileged", but rather its ability to have a system-wide impact.

There are various mechanisms or techniques that can be employed to give a process extra privilege or, in the case of FACE UoCs, avoided to limit its privilege. Some examples include:

- Startup processes and kernel processes may have access to the entire physical or virtual address space
- Run-time access to processor or I/O hardware through address mapping or privileged instructions
- Run-time access to other processes' address space through memory mapping
- Run-time access to system services that control system-wide state, such as system reset and system time
- Run-time access to system configuration data

Privileged processes are often the target of cyber-attacks and hosted FACE UoCs are indirectly at risk because of this threat. These cyber-attacks usually either seek to gain control of a privileged process or elevate a malicious or compromised process to privileged status. The scenarios for these attacks are almost endless.

Systems that host FACE UoCs should consider and mitigate this threat even though the FACE UoCs do not themselves typically contain privileged processes by design. This threat and the security controls to mitigate are not new or unique to FACE UoCs. The threat can be mitigated by limiting the privilege of FACE UoCs and applying currently accepted security approaches, such as the NIST Risk Management Framework and the associated SP 800-53 security controls to the larger system as appropriate.

7.4.5 System State Management

7.4.5.1 Nominal Operational Conditions

It is the responsibility of the platform integrator to ensure that prior to any instantiation of any FACE UoC, the system must be in a known secure and safe state. While not explicitly addressed in the FACE Technical Standard, the system boot process must be secure. After the initial boot process, there may also be inter-partition state management operations beyond the scope of the FACE Technical Standard, which must also be secure.

The FACE Technical Standard includes intra-partition provisions for LCM capabilities that can be used to control UoC initialization, configuration, connection, and state transitions. While these capabilities are optional, they are extremely important to maintain a secure system and great care should be taken to ensure these do not pose a risk to the system. Therefore, any state management functions should be controlled by trusted components or data.

As an example, if a UoC contains critical functionality, it should only be controlled by a trusted process or data and may require additional authentication or unique configurations before the UoC functionality is enabled. Also, once a critical UoC is enabled, it should not be disabled unless a trusted process determines that there is a failure condition impacting the operational integrity of the UoC, or if the system is transitioning into a non-operational state. To support this, the FACE Technical Standard allows the UoC provider to tailor configuration and initialization parameters, as well as use the LCM Stateful interface to transition the UoC.

While security is not explicitly included in these features, there are additional security controls that could be added to provide additional confidence in managing these state transitions (e.g., authentication or hash included with the LCM configuration parameters).

7.4.5.2 Failure Conditions

HMFM provides traditional fault management monitoring and reporting; however, this may not be adequate for all security-relevant events. For UoCs that include security-relevant events, it is the responsibility of the platform integrator to design the security event detection, containment, reporting, and recovery capability. This may be a combination of HMFM and custom designed, but FACE conformant components.

7.5 Security Transformations

7.5.1 Introduction

Transformations are one type of a security control to protect the confidentiality and integrity of data in transit and at rest within a secure system. Examples include authentication, encryption, and labeling. A Security Transformation may also be used to protect Critical Program Information (CPI). This approach supports isolation of security-enforcing functions which may ease a security evaluation by limiting the elements that would necessitate an assessment. Security Transformations are intended exclusively for security capabilities designed to meet system-level security controls. Information on the Security Transformation needs to be captured to ensure portability of PCS, PSSS, and TSS UoCs. Given the sensitivity of both the data and transformation mechanism, there may be restrictions on availability and distribution of this information.

7.5.2 Data Modeling Requirements for Security Transformation

To maintain an open interface, the data in a Security Transformation is required to be represented in a USM per the FACE Data Architecture. The USM for any Security Transformation should include any necessary configuration data, and control data as well as the data being transformed.

It is important to note that the model for the transformed data does not require specific knowledge of the data or how it is used; it is only required to be modeled as a generic payload of data (such as an octet sequence of encrypted data) on which the transform will operate. The USM should include the configuration and control information (such as synchronous or asynchronous control of payload processing) necessary to guarantee correct operation by the recipient of the transform as well as interoperability between peer users of the data. Figure 9 illustrates options for where the UoP USM would exist pre and post-transformation based on the location of the Security Transformation.

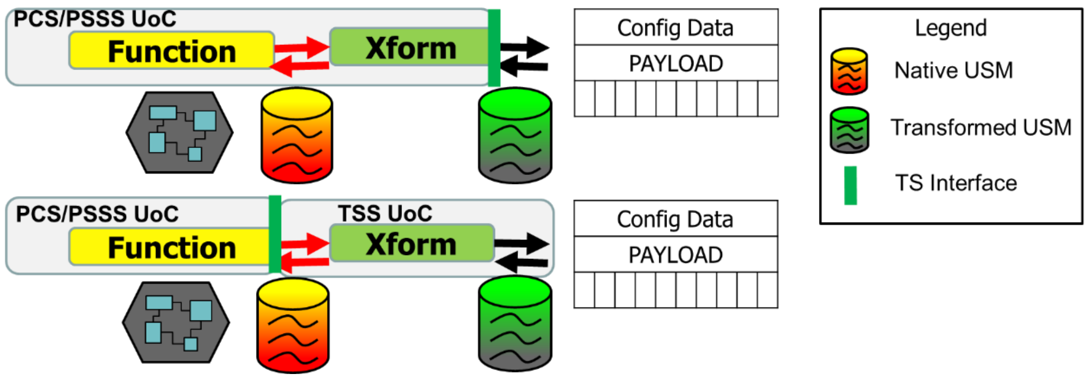


Figure 9: Security Transformation Modeling Examples

7.5.3 Security Transformation Considerations

A variety of factors will influence within which segment a Security Transformation should reside, such as overall system design, hardware architecture, software architecture, security architecture, latency requirements, human factors, etc. Table 14 identifies some of the considerations when determining which segment to implement a security function that requires a Security Transformation.

Table 14: Security Transformation Segment Considerations

Segment	Considerations
PCS	Portable and reusable security services (authentication server, key management), certifying authority restrictions
PSSS	Interactions with physical devices (storage, trusted processing modules, hardware-based accelerators)
TSS	Performance requirements, reduced number of enforcement points, deployment of security as infrastructure services (data mediation, data transforms, labeling, logging)

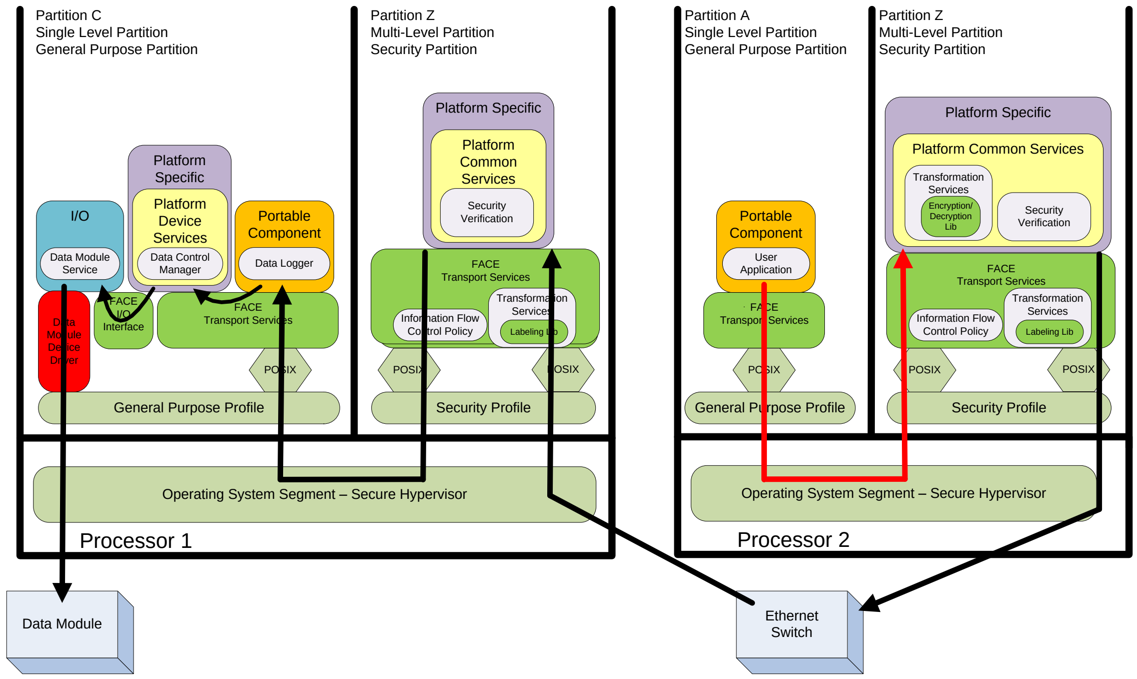


Figure 10: Security Transformation Example

7.6 Example Scenarios – Team

The following example is based on Scenario B from the NSA SSE-100-1. This document is used to illustrate the security implications of the FACE Profiles against a publicly available document. It is recommended that NSA SSE-100-1 be referenced for further details not provided in the sections below.

NSA SSE-100-1 describes four operational scenarios utilizing a separation kernel-based RTOS and IA guidance for security-critical functions. For the purposes of our implementation discussion, we have elected to only present one of the four given the potential applicability to avionics platforms. Scenario B is utilized because it describes an embedded system with multiple processors such as an airborne system with limited network connectivity. The system is comprised of four processors operating on a partitioned RTOS and shows the horizontal communication between each partition. The partitions are operating from a single domain to multiple domains requiring varying levels of Robustness, as defined in NSA SSE-100-1, from Basic to High. The components for which High Robustness is recommended are all components where a failure within that component alone could result in the compromise of classified data (a breach in confidentiality). High Robustness is appropriate for this scenario because the highest classification of the data is identified as Top Secret and restricts user access in the system.

7.6.1 Scenario B

The example system provides Top Secret cleared users with fused situational awareness using data from within the system and from external input. The system is comprised of four independent processors each utilizing a partitioned RTOS and communicating via a Commercial Off-The-Shelf (COTS) Ethernet switch. The RTOS instance provides partitions supporting both single domain and multiple-domains with varying levels of Robustness from Basic to High, as shown in Figure 11.

Basic to High Robustness for these components would require partition isolation and information flow control between partitions on a processor, with the necessary monitoring controls to prevent failures that result in Top Secret data being available to unclassified users. Within Scenario B the cross-domain guard (identified in partition 1B) should achieve High Robustness.

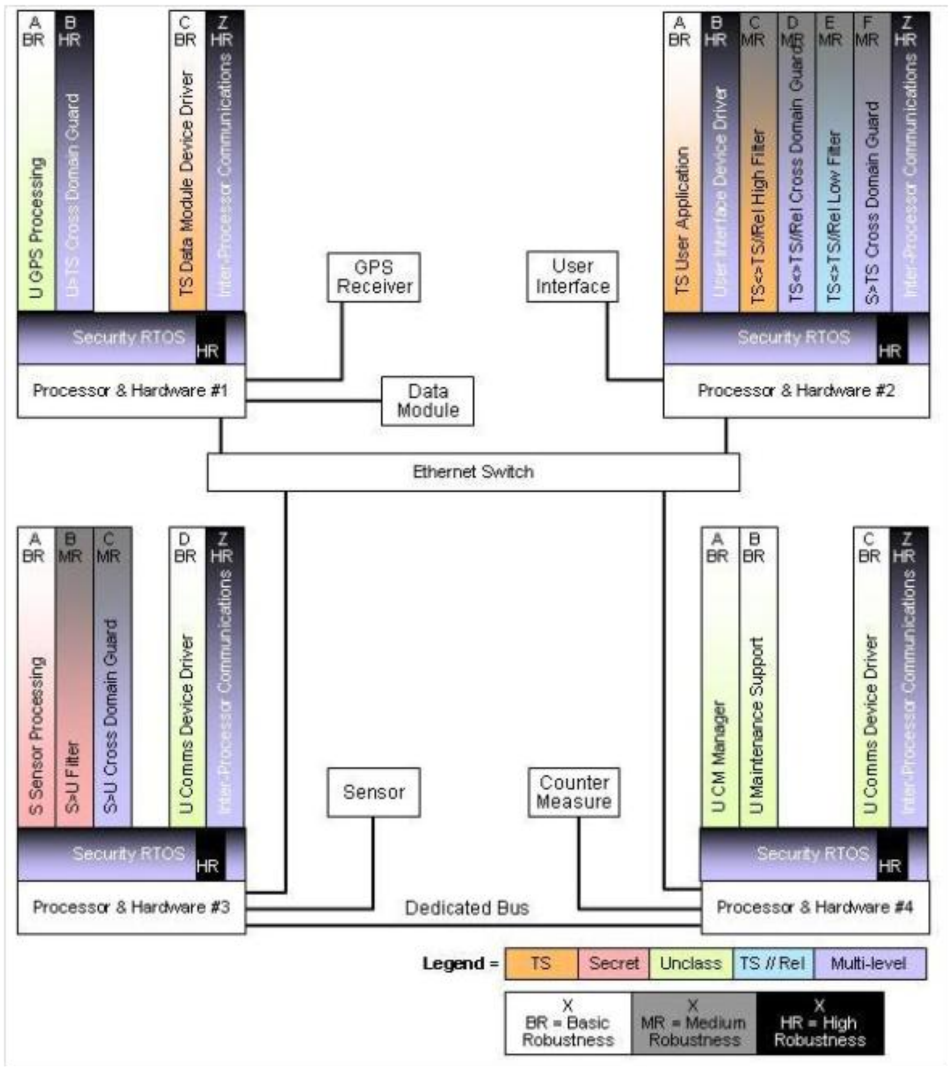

A BR U GPS Processing
B HR U>TS Cross Domain Guard
C BR TS Data Module Device Driver
Z HR Inter-Processor Communications
Security RTOS HR
Processor & Hardware #1
GPS Receiver
Data Module
User Interface
A BR TS User Application
B HR User Interface Device Driver
C MR TS<>TS//Rel High Filter
D MR TS<>TS//Rel Cross Domain Guard
E MR TS<>TS//Rel Low Filter
F MR S>TS Cross Domain Guard
Z HR Inter-Processor Communications
Processor & Hardware #2
Ethernet Switch
A BR S Sensor Processing
B MR S>U Filter
C MR S>U Cross Domain Guard
D BR U Comms Device Driver
Z HR Inter-Processor Communications
Processor & Hardware #3
Sensor
Counter Measure
Dedicated Bus
A BR U CM Manager
B BR U Maintenance Support
C BR U Comms Device Driver
Z HR Inter-Processor Communications
Processor & Hardware #4
Legend = TS Secret Unclass TS // Rel Multi-level
X BR = Basic Robustness
X MR = Medium Robustness
X HR = High Robustness

Figure 11: Scenario B Component Robustness Levels

To reiterate, the focus of the FACE Security Profile is to address the security-critical functions of the architecture that require High Robustness.

7.7 Security Standard References

The following recognized security standards, or guidance documents, are provided for informational purposes and include descriptions of each document, as well as their uses.

7.7.1 Software Engineering Institute Secure Coding Standard

The Software Engineering Institute (SEI) Secure Coding Standard provides developers with a set of rules and recommendations for the construction of secure code. The standard supports commonly used programming languages including C, C++, Java, and Perl. The standard identifies common application vulnerabilities and remedies that can be incorporated into the source code. A risk assessment is conducted for each rule/recommendation where it is evaluated for severity, likelihood, remediation cost, priority, and level.

7.7.2 Common Weakness Enumeration

International in scope and free for public use, Common Weakness Enumeration (CWE) provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

7.7.3 Open Web Application Security Project

The Open Web Application Security Project (OWASP) is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security. Even though avionics applications do not operate on the web, many of the vulnerabilities that apply to the web also apply to the avionics domain.

7.7.4 ISO/IEC 27034

ISO/IEC 27034 offers guidance on information security to those specifying, designing, and programming or procuring, implementing, and using application systems; in other words, business and IT managers, developers and auditors, and ultimately the end users of ICT. The aim is to ensure that computer applications deliver the desired or necessary level of security in support of the organization’s Information Security Management System, adequately addressing many ICT security risks.

7.7.5 Risk Management Framework for DoD IT

The Risk Management Framework (RMF) is a risk-based approach to IT cycle management by the DoD as captured in NIST SP 800-37: Guide for Applying the Risk Management Framework to Federal Information Systems. At a high level, the RMF is implemented in six (6) basic steps as follows:

1. Categorize information system
2. Select security controls
3. Implement security controls
4. Assess security controls
5. Authorize information system
6. Monitor security controls

For the most part, no one FACE UoC will proceed through this process alone, but rather the RMF applies to the whole system.

7.7.6 CMU/SEI-2009-SR-001

CMU/SEI-2009-SR-001: Making the Business Case for Software Assurance is a special report published by the Carnegie Mellon Software Engineering Institute. The intended audience for this guide is primarily software developers and software managers with an interest in assurance and people from a security department who work with developers. While generic in nature, this document provides a very good overview of security considerations and key steps to ensure robust and secure software designs.

7.7.7 Program Protection Plan

Each platform that has security requirements will need to develop a Program Protection Plan (PPP). The U.S. DoD has published guidance on the organization and content of this in both the Program Protection Plan Outline & Guidance and Software Assurance Countermeasures in Program Protection Planning.

7.7.8 Systems Security Engineering

NIST SP 800-160 addresses the systems engineering required to develop secure systems. It targets the system’s life cycle processes for new systems, systems upgrades, or repurposing systems for new uses. It is not a how-to guide implementation guide, but addresses considerations for security within the context of systems engineering.

8 Implementation Scenario Examples

8.1 Communications Manager Scenario

8.1.1 Introduction

Communications Management (COMM) is the management and control system for all of the communications equipment on-board the air platform. For the purposes of this implementation scenario, the COMM function receives control information (i.e., frequencies, channels, presets, keys) from the CDU and displays communications information back on the CDU. The COMM function also manages the communications devices. This scenario begins after the Data Transfer Device (DTD) has already been loaded onto the aircraft. This scenario excludes the pre-flight and post-flight mission planning activities.

8.1.2 Scope

8.1.2.1 Operational View

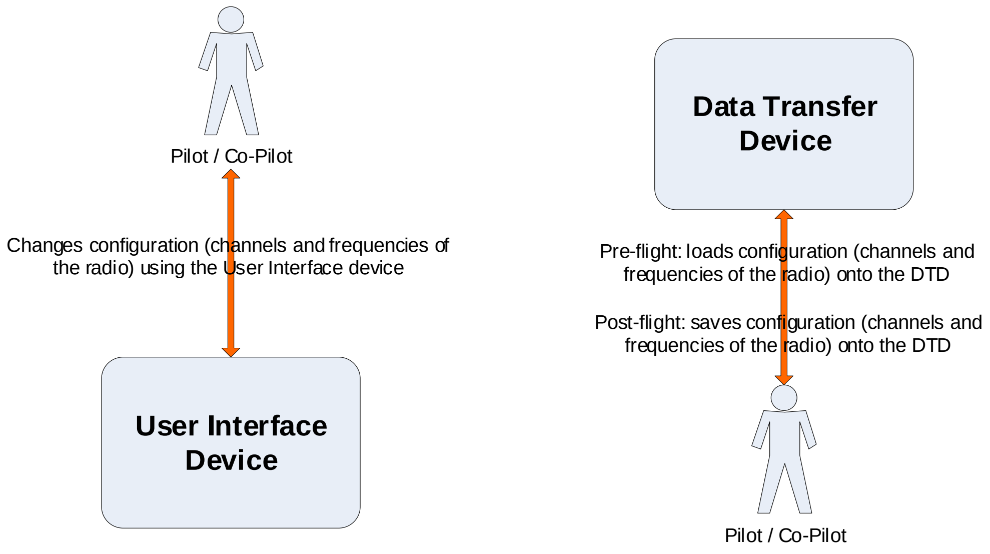

Pilot / Co-Pilot
Changes configuration (channels and frequencies of the radio) using the User Interface device
User Interface Device
Data Transfer Device
Pre-flight: loads configuration (channels and frequencies of the radio) onto the DTD
Post-flight: saves configuration (channels and frequencies of the radio) onto the DTD
Pilot / Co-Pilot

Figure 12: Communication Manager Operational View

The user (Pilot/Co-Pilot) interfaces with the radio by interactions with the User Interface Device. The user is capable of changing channels, frequencies, presets, and antenna selection of the radio using this user input device.

The user interfaces through the User Interface Device with the DTD by loading presets, and operational configuration files. The user interfaces through the User Interface Device with the Data Transfer Device to save presets and operational configuration files.

8.1.2.2 *Functional View*

The view in Figure 13 is shown from the point after the DTD has already been loaded onto the aircraft. This excludes the pre-flight and post-flight actions.

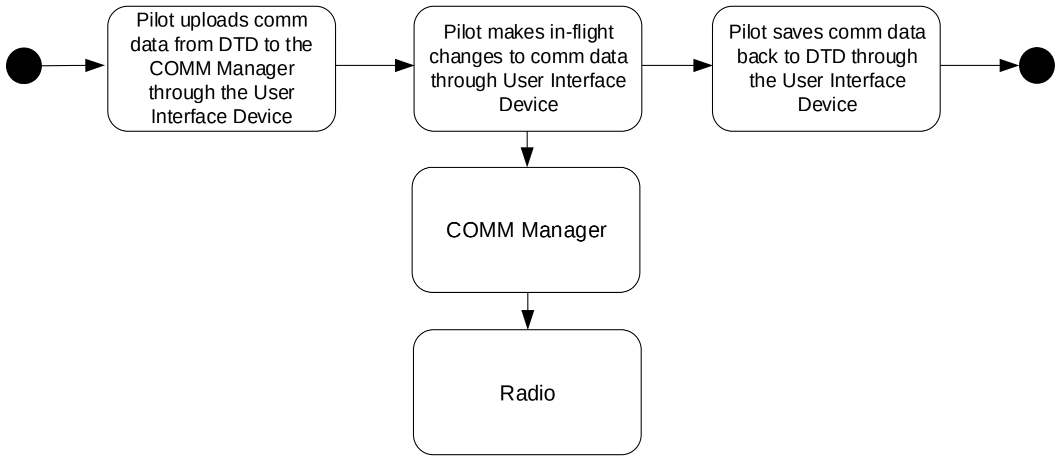

Pilot uploads comm data from DTD to the COMM Manager through the User Interface Device
Pilot makes in-flight changes to comm data through User Interface Device
Pilot saves comm data back to DTD through the User Interface Device
COMM Manager
Radio

Figure 13: Communication Manager Functional View

8.1.2.3 *Physical View*

Figure 14 shows the example FACE Computing Environment and associated UoCs separated by their respective FACE Segments.

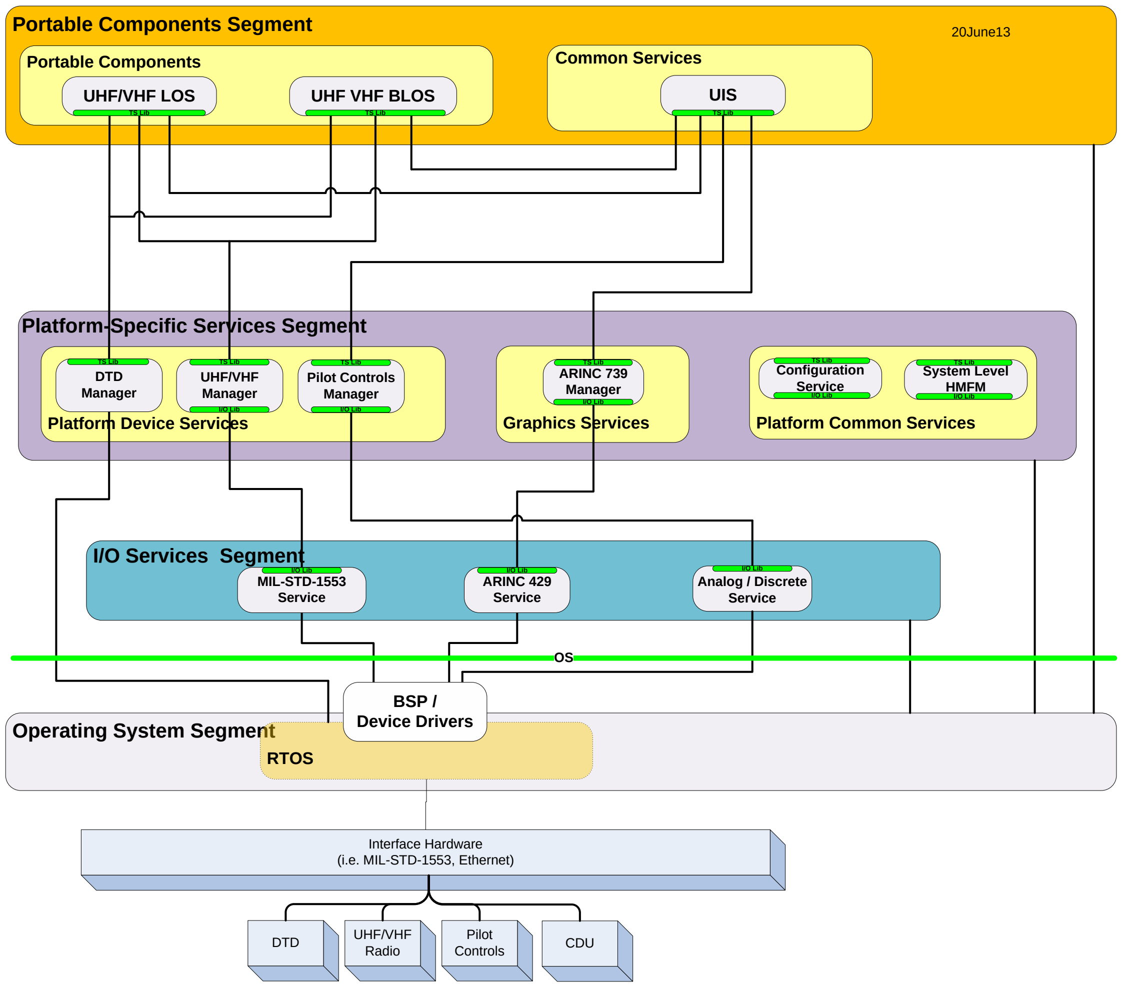


Figure 14: Communication Manager Physical View

8.1.2.4 *Partitioned View*

Figure 15 features a partitioned view of the FACE Computing Environment and associated UoCs defined in this example.

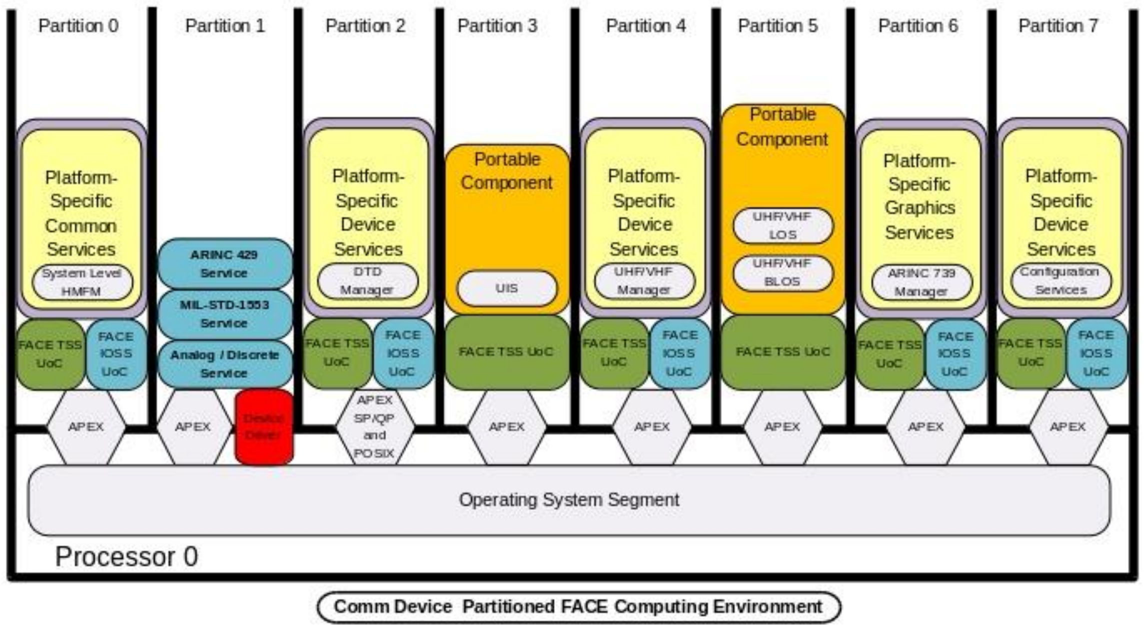


Figure 15: Communications Manager Partitioned View

8.1.3 Assumptions

The devices that are depicted in this scenario are a DTD, Ultra High Frequency (UHF)/Very High Frequency (VHF) Radio, Pilot Controls, and CDU. The CDU uses the ARINC 739 protocol and is connected over an ARINC 429 bus. A UHF/VHF Radio is connected over a MIL-STD-1553 bus. The Pilot Controls use analog and discrete inputs. The DTD is connected over Ethernet.

8.1.3.1 Profile

The software is run on a single processor in eight different partitions. The General Purpose Profile is being used with both ARINC 653 (APEX) partitions and POSIX partitions.

8.1.3.2 Performance

No strict real-time requirements. The User Interface Service (UIS) must provide suitable human response times.

8.1.4 Computing Environment

8.1.4.1 Operating System Segment

The OSS provides and controls access to the computing platform for the other FACE Segments.

8.1.4.1.1 The POSIX Standard

The OSS implements the FACE General Purpose Profile for the POSIX standard as defined in the FACE Technical Standard §3.11.3.

8.1.4.1.2 ARINC 653

The OSS implements the FACE General Purpose Profile for ARINC 653 (APEX) as defined in the FACE Technical Standard §3.11.3.

8.1.4.1.3 Board Support Package/Device Drivers

The OSS provides the appropriate Board Support Package (BSP) and/or device drivers for the MIL-STD-1553, ARINC 429, Ethernet, and discrete I/O interfaces.

8.1.4.2 I/O Services Segment

The IOSS provides a common interface for I/O devices using the FACE interfaces.

8.1.4.2.1 MIL-STD-1553 I/O Service

The MIL-STD-1553 I/O Service normalizes the interface between the BSP and software component(s) communicating with the platform device over the MIL-STD-1553 bus.

8.1.4.2.2 ARINC 429 I/O Service

The ARINC 429 I/O Service normalizes the interface between the BSP and software component(s) communicating with the platform device over ARINC 429.

8.1.4.2.3 Discrete I/O Service

The Discrete I/O Service normalizes the interface between the BSP and software component(s) requiring the discrete inputs over the corresponding discrete connections.

8.1.4.3 Platform-Specific Services Segment

The PSSS allows for platform-unique software component combinations corresponding to the specific platform’s requirements.

8.1.4.3.1 Platform-Specific Device Services

The Platform-Specific Device Service (PDS) includes components that provide control for, receive data from, and send data to platform devices or external systems.

UHF/VHF Manager

- Communicates with the UHF/VHF device over the FACE I/O Interface according to the UHF/VHF device’s ICD
- Receives data from the other FACE UoCs and translates the data from the FACE USM into messages the UHF/VHF device can understand
- Translates the ICD-defined data into the FACE USM for communication with other FACE UoCs:
  - — Provides health and status of the UHF/VHF device – Periodic Built-In Test (PBIT), Continuous Built-In Test (CBIT), Initiated Built-In Test (IBIT)
  - — Provides command and control of the UHF/VHF device

Pilot Controls Manager

- Communicates with the Pilot Controls (e.g., stick, throttle, collective, cyclic) using discrete inputs over the FACE I/O Interface
- Receives data from the other FACE UoCs and translates the data from the FACE USM into messages the Pilot Controls device can understand
- Translates inputs made from the Pilot Controls device into the FACE USM for communication to other software components in the FACE Computing Environment:

  — Provides health and status of the Pilot Controls device (PBIT, CBIT, IBIT)

  — Provides command and control of the Pilot Controls device

DTD Manager

- Communicates with the DTD using Ethernet
- Receives data from the other FACE UoCs and translates data from the FACE USM into messages the DTD can understand
- Translates inputs from the DTD into the FACE USM for communication to other software components in the FACE Computing Environment:

  — Provides health and status of the DTD (PBIT, CBIT, IBIT)

  — Provides command and control of the DTD device

8.1.4.3.2 Platform-Specific Graphic Services

ARINC 739 Manager

- Provides a standardized interface for client software components to display text-based information on a CDU
- Manages the currently active subsystems
- Controls the subsystem menu display
- Provides subsystem fault detection
- Provides an abstraction for non-standard ARINC 739 hardware

8.1.4.3.3 Platform-Specific Common Services

The Platform-Specific Common Services provides common services to all other FACE Computing Environment segments per system requirements. The Platform-Specific Common Services support Centralized Configuration Services and System-level Health Monitoring for this implementation are as defined in the FACE Technical Standard §3.6.3.1.4 and §3.6.3.1.5, respectively.

8.1.4.4 Transport Services Segment

8.1.4.4.1 Components

In this implementation, there are no stand-alone services needed to handle the TS. The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. Refer to Section 8.1.4.7 for more implementation-specific details.

8.1.4.5 Portable Components Segment

The PCS is a portion of a FACE solution whose contents are entirely independent from other FACE Segments. PCS components exclusively use the TS Interface for data exchanges.

8.1.4.5.1 Common Services

The UIS is a configurable ARINC 739 client software component which provides a service for software components to display and format information on the CDU. The UIS communicates with the ARINC 739 Graphics Service (in PSSS) using the ARINC 739 command stream over the TS Interface, as described in the FACE Technical Standard §3.12.9. In addition to ARINC 739, the UIS can also communicate with other PSSS services (e.g., Pilot Controls Manager) to be provided user inputs from other sources. The UIS communicates with other FACE UoCs over the TS Interface. The actual information (e.g., menus, text fields, options) which is displayed on the UIS is exchanged with other FACE UoCs and is retrieved using the FACE Configuration Services. The data configuring the display is retrieved using FACE Centralized Configuration Services using the TS Interface and then provided to the ARINC 739 manager using the TS Interface.

8.1.4.5.2 Portable Applications

Two similar portable applications are included in the example. They could be combined into a single UoP, but have been separated to demonstrate the potential for additional reusability. If one of the components was useful in a future FACE Computing Environment, it could be reused without requiring modification.

UHF/VHF Line of Sight

The UHF/VHF Line of Sight (LOS) portable application manages frequency data, channel information, and radio status for UHF/VHF LOS radios on the platform. It communicates with actual radios through a UHF/VHF Manager located in the PSSS over the TS Interface using the FACE Data Architecture. The UHF/VHF Manager provides any necessary translation between the USM and the specific UHF/VHF Radio ICD. The UHF/VHF LOS portable application also provides information to be displayed by the UIS over the TS Interface.

UHF/VHF Beyond Line of Sight

The UHF/VHF Beyond Line of Sight (BLOS) portable application manages frequency data, channel information, and radio status for UHF/VHF BLOS radios on the platform. It communicates with actual radios through a UHF/VHF Manager located in the PSSS over the TS Interface using the FACE Data Architecture. The UHF/VHF Manager provides any necessary translation between the USM and the specific UHF/VHF Radio ICD. The UHF/VHF BLOS portable application also provides information to be displayed by the UIS over the TS Interface.

8.1.4.6 *IOSS Interface*

The IOSS Interface is instantiated within an IOSS UoC which contains the business logic for the specific transport mechanism. This is only used between IOSS and PSSS.

8.1.4.6.1 *Analog and Discrete I/O Services to Pilot Controls Manager*

The Analog and Discrete I/O Services request data from the device driver included in the BSP. The Analog and Discrete I/O Services then send the command to the Pilot Control Manager, which then makes those inputs available to the rest of the FACE Computing Environment via the TS Interface.

8.1.4.6.2 *ARINC 429 Service to ARINC 739 Manager*

The ARINC 429 I/O Service sends/receives ARINC 739 command stream data to/from the ARINC 429 device driver included in the BSP. The ARINC 429 I/O Service then sends/receives the data to the ARINC 739 manager, which then makes the ARINC 739 interface available to other FACE UoCs over the TS Interface using the FACE Data Architecture.

8.1.4.6.3 *MIL-STD-1553 Service to Radio Manager*

The MIL-STD-1553 I/O Service sends/receives radio data, per the radio ICD, to/from the MIL-STD-1553 device driver included in the BSP. The MIL-STD-1553 I/O Service then sends/receives the data to the UHF/VHF Manager, which then makes the radio data available to other FACE UoCs over the TS Interface using the FACE Data Architecture.

8.1.4.6.4 *System-Level HMFM ↔ **

All services in the IOSS communicate their health status to the System-Level HMFM through the IOSS Interface. Figure 14 does not show the communication lines between FACE Configuration Services and all other components. This was intentional to minimize the complexity of the diagram.

8.1.4.6.5 *Configuration Services ↔ **

I/O Services are configured using the FACE Configuration Services Interface.

8.1.4.7 *TS Interface*

The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. This is only used between PCS and PSSS. All data passed through the TS Interface uses the FACE Data Architecture.

The source and destination for connections are used by the TS library and are defined in the TS library configuration data; refer to the FACE Technical Standard §3.7.

8.1.4.7.1 *PSSS ↔ PCS*

System-Level HMFM ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS communicate their health status to the System-Level HMFM through the TS Interface. Figure 14 does not show the communication lines between Centralized Configuration Services and all other components. This was intentional to minimize complexity of the diagram.

Centralized Configuration Services ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS receive configuration data from Centralized Configuration Services through the TS Interface. Figure 14 does not show the communication lines between Centralized Configuration Services and all other components. This was intentional to minimize the complexity of the diagram.

DTD Manager ↔ UHF/VHF LOS

The UHF/VHF LOS component communicates with the DTD Manager to send/receive preset data, frequency data, and radio data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD using the OS Interface (Ethernet).

DTD Manager ↔ UHF/VHF BLOS

The UHF/VHF BLOS component communicates with the DTD Manager to send/receive preset data, frequency data, and radio data to be written to and/or read from to the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD using the OSS Interface (Ethernet).

UHF/VHF Manager ↔ UHF/VHF LOS

The UHF/VHF LOS component sends/receives frequency data, channel information, and radio status to/from the UHF/VHF Manager through the TS Interface.

UHF/VHF Manager ↔ UHF/VHF BLOS

The VHF/UHF BLOS component sends/receives frequency data, channel information, and radio status to/from the UHF/VHF Manager through the TS Interface.

Pilot Controls Manager → UIS

The UIS component receives information about Pilot Controls inputs from the Pilot Controls Manager through the TS Interface. These Pilot Controls inputs are used to perform actions which are available through the CDU user interface, such as selecting a radio preset.

ARINC 739 Manager ↔ UIS

The UIS component communicates with the ARINC 739 using an ARINC 739 command stream over the TS Interface.

8.1.4.7.2 PCS ↔ PCS

UIS ↔ UHF/VHF BLOS

The UIS common service communicates with the UHF/VHF BLOS portable application to send/receive radio configuration data, frequency data, and preset data through the TS Interface. This data is sent to the UHF/VHF BLOS portable application from the UIS common service when a user input is made on the CDU or Pilot Controls. This data is sent from the UHF/VHF

BLOS portable application to the UIS common service in order to populate the CDU with accurate UHF/VHF BLOS information.

UIS ↔ UHF/VHF LOS

The UIS common service communicates with the UHF/VHF LOS portable application to send/receive radio configuration data, frequency data, and preset data through the TS Interface. This data is sent to the UHF/VHF LOS portable application from the UIS common service when a user input is made on the CDU or Pilot Controls. This data is sent from the UHF/VHF LOS portable application to the UIS common service in order to populate the CDU with accurate UHF/VHF LOS information.

8.2 Digital Map Manager Scenario

8.2.1 Introduction

The Digital Map (DigMap) is an application that provides a digital image of a map or terrain at a given position. It is used by the pilot for situational awareness. For the purposes of this implementation, the DigMap application receives Digital Terrain Elevation Data (DTED) data and Navigation Data (i.e., Altitude, Heading, Airspeed, Lat/Long) and provides a digital map image on the CDS. This scenario begins after the DTD has already been loaded onto the aircraft. This excludes the pre-flight and post-flight actions.

8.2.2 Scope

8.2.2.1 Operational View

Figure 16 shows the use cases defined in this section, as well as the actors involved in the use cases.

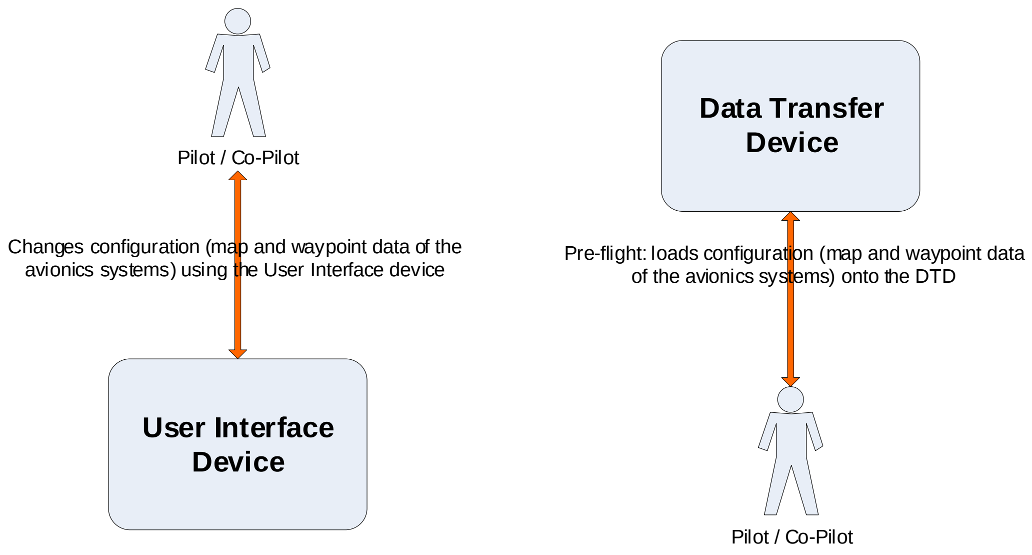

Pilot / Co-Pilot
Changes configuration (map and waypoint data of the avionics systems) using the User Interface device
User Interface Device
Data Transfer Device
Pre-flight: loads configuration (map and waypoint data of the avionics systems) onto the DTD
Pilot / Co-Pilot

Figure 16: Digital Map Operational View

The user (Pilot/Co-Pilot) interfaces with the EGI by interactions with the User Interface Device. The user is capable of changing EGI operational configuration and source selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the Air Data Computer by interactions with the User Interface Device. The user is capable of changing Air Data Computer operational configuration and source selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the VHF Omni-directional Radio-Range (VOR)/Distance Measuring Equipment (DME)/Instrument Landing System (ILS) by interactions with the User Interface Device. The user is capable of changing VOR/DME/ILS operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the Tactical Air Navigation (TACAN) by interactions with the User Interface Device. The user is capable of changing TACAN operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the Radar Altimeter by interactions with the User Interface Device. The user is capable of changing Radar Altimeter operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces through the User Interface Device with the DTD by loading presets and configuration files. The user interfaces through the User Interface Device with the Data Transfer Device to save presets and operational configuration files.

The user (Pilot/Co-Pilot) controls overlays, digital map source, map type, and configuration through the bezels on the display.

8.2.2.2 *Functional View*

The view shown in Figure 17 begins after the DTD has already been loaded onto the aircraft. This excludes the pre-flight and post-flight actions.

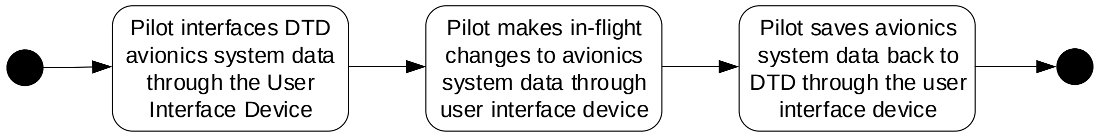

Pilot interfaces DTD avionics system data through the User Interface Device
Pilot makes in-flight changes to avionics system data through user interface device
Pilot saves avionics system data back to DTD through the user interface device

Figure 17: Digital Map Functional View

8.2.2.3 *Physical View*

Figure 18 shows the example FACE Computing Environment and associated UoCs separated by their respective FACE Segments.

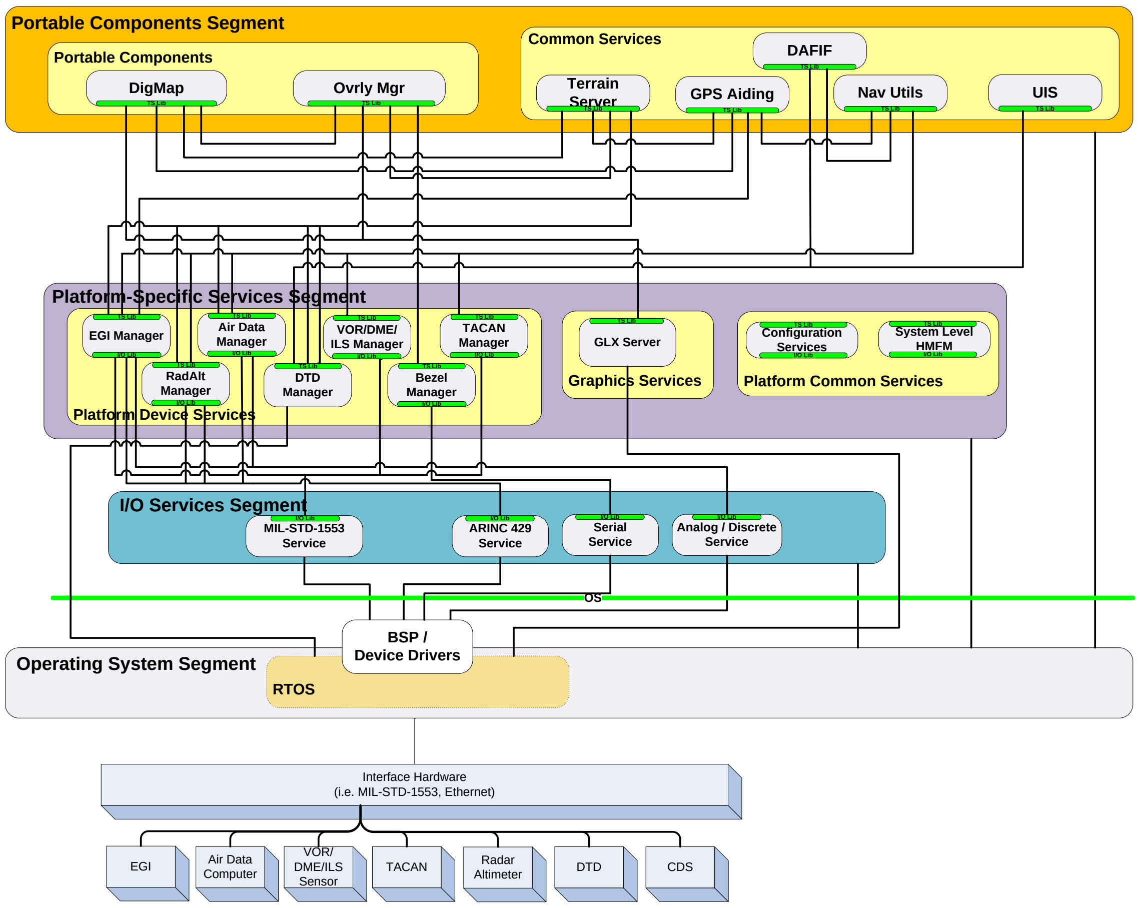


Figure 18: Digital Map Physical View

8.2.2.4 *Partitioned View*

Figure 19 features a partitioned view of the FACE Computing Environment and associated UoCs defined in this example.

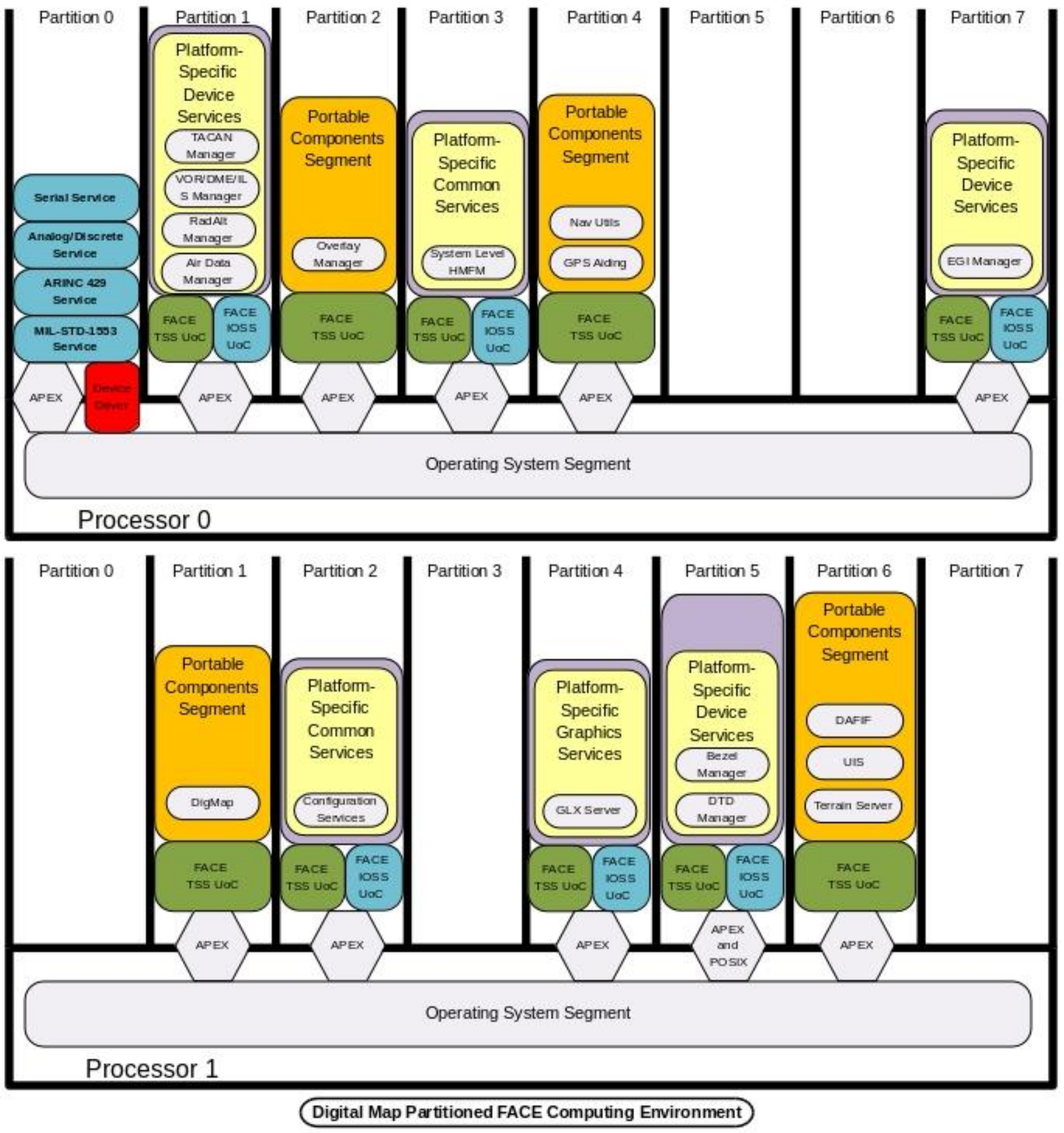


Figure 19: Digital Map Partition View

8.2.3 Assumptions

The devices that are depicted in this scenario are a CDS, an EGI, an Air Data Computer, a VOR/DME/ILS sensor, a TACAN sensor, and a Radar Altimeter. The CDS uses the OpenGL protocol and is connected over an Ethernet bus. Inputs from the bezels on the display are provided over a serial service. An EGI is connected over a MIL-STD-1553 and ARINC 429 bus. An Air Data Manager is connected over an ARINC 429 bus. Both the EGI and Air Data Manager receive discrete inputs provided over analog connections. VOR/DME/ILS and TACAN sensors are connected over a MIL-STD-1553 bus. A Radar Altimeter is connected over an ARINC 429 bus.

8.2.3.1 *Profile*

The software is run on two processors in 11 different partitions. The empty partitions could be used for expansion and future development.

This scenario could be implemented with the Safety Profile or the General Purpose Profile. This implementation assumes the General Purpose Profile is being used with both ARINC 653 (APEX) partitions and POSIX partitions.

8.2.3.2 *Performance*

EGI and ILS have real-time requirements. UIS must provide suitable human response times.

8.2.4 Computing Environment

8.2.4.1 *Operating System Segment*

The OSS provides and controls access to the computing platform for the other FACE Segments.

8.2.4.1.1 *The POSIX Standard*

The OSS implements the FACE General Purpose Profile for the POSIX standard as defined in the FACE Technical Standard §3.2.1.5.

8.2.4.1.2 *ARINC 653*

The OSS implements the FACE General Purpose Profile for ARINC 653 (APEX) as defined in the FACE Technical Standard §3.2.1.5.

8.2.4.1.3 *Board Support Package/Device Drivers*

The OSS provides the appropriate BSP and/or device drivers for the MIL-STD-1553, ARINC 429, Ethernet, and discrete I/O interfaces.

8.2.4.2 *I/O Services Segment*

The IOSS provides a common interface for I/O devices using the standardized API.

8.2.4.2.1 *MIL-STD-1553*

The MIL-STD-1553 service normalizes the interface between the BSP and software component(s) communicating with the platform device over the MIL-STD-1553 bus.

8.2.4.2.2 *ARINC 429*

The ARINC 429 service normalizes the interface between the BSP and software component(s) communicating with the platform devices over ARINC 429.

8.2.4.2.3 *Serial*

The Serial service normalizes the interface between the BSP and software component(s) communicating with the platform device using the serial bus.

8.2.4.2.4 Analog/Discrete

The Analog/Discrete service normalizes the interface between the BSP and software component(s) requiring the discrete inputs over the corresponding analog/discrete connections.

8.2.4.3 Platform-Specific Services Segment

The PSSS allows for platform-unique software component combinations corresponding to the specific platform requirements.

8.2.4.3.1 Platform-Specific Device Services

The Platform-Specific Device Services include components that provide control for, receive data from, and send data to platform devices or external systems.

EGI Manager

- Communicates with the EGI device over the FACE IOSS Interface according to the EGI device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the EGI device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the EGI device (PBIT, CBIT, IBIT)

  — Provides command and control of the EGI device

Air Data Manager

- Communicates with the Air Data Computer device over the FACE IOSS Interface according to the Air Data Computer device's ICD
- Receives data from the other FACE UoCs and translates the data into data formats and messages the Air Data Computer can understand
- Translates the ICD-defined data into data formats and messages that the other FACE UoCs can understand:

  — Provides health and status of the Air Data Computer (PBIT, CBIT, IBIT)

  — Can provide command and control of the Air Data Computer and managed device states

Radar Altimeter Manager

- Communicates with the Radar Altimeter device over the FACE IOSS Interface according to the Radar Altimeter device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the Radar Altimeter device can understand

- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the Radar Altimeter device (PBIT, CBIT, IBIT)

  — Provides command and control of the Radar Altimeter device

TACAN Manager

- Communicates with the TACAN device over the FACE IOSS Interface according to the TACAN device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the TACAN device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the TACAN device (PBIT, CBIT, IBIT)

  — Provides command and control of the TACAN device

VOR/ILS/DME Manager

- Communicates with the VOR/ILS/DME device over the FACE IOSS Interface according to the VOR/ILS/DME device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the VOR/ILS/DME device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the VOR/ILS/DME device (PBIT, CBIT, IBIT)

  — Provides command and control of the VOR/ILS/DME device

DTD Manager

- Communicates with the DTD using Ethernet
- Receives data from the other FACE UoCs and translates the data from the USM into messages the DTD can understand
- Translates inputs from the DTD into the USM for communication to other software components in the FACE Computing Environment:

  — Provides health and status of the DTD (PBIT, CBIT, IBIT)

  — Provides command and control of the DTD and managed device states

Bezel Manager

- Is the gateway to the CDS for software components
- Understands the data specifications for the platform CDS

- Provides a common interpretation of the data to software components
- Handles health and status for the CDS (PBIT, CBIT, IBIT)
- Communicates to other software components through the TS Interface

8.2.4.3.2 Platform-Specific Graphics Services

The OpenGL Extension to the X Window System (GLX) Server provides a standardized interface for client software components to display OpenGL-based information on a CDS. The GLX Server allows for separation of the CDS display from the PCS components. OpenGL commands are serialized and transmitted as packets to the remote display network location. When received by the target display manager, those commands are de-serialized and passed to the local display drivers.

8.2.4.3.3 Platform-Specific Common Services

The Platform-Specific Common Services provide common services to all other FACE Computing Environment segments per system requirements. The Platform-Specific Common Services support Centralized Configuration Services and System-level Health Monitoring for this implementation as defined in the FACE Technical Standard §3.6.3.1.4 and §3.6.3.1.5, respectively.

8.2.4.4 Transport Services Segment

8.2.4.4.1 Components

In this implementation, there are no components needed to handle the TSS. The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. Refer to Section 8.2.4.7 for more implementation-specific details.

8.2.4.5 Portable Components Segment

The PCS is a portion of a FACE solution whose contents are entirely independent from other FACE Segments. PCS components exclusively use the TS Interface for data exchanges.

8.2.4.5.1 Portable Applications

Digital Map

The Digital Map (DigMap) is a portable application that renders a map to send to the GLX Server given terrain and GPS data as inputs.

Overlay Manager

The Overlay Manager is a portable application that provides navigation and sensor data given terrain and GPS data as inputs.

8.2.4.5.2 Common Services

Digital Aeronautical Flight Information File Service

The Digital Aeronautical Flight Information File (DAFIF) service provides airport and navaid data given ownship position and inputs from the DTD Manager.

Terrain Server

The Terrain Server is a service that provides conversion utilities and processed DTED data needed by the portable applications.

GPS Aiding Service

The GPS Aiding service provides heading and airspeed smoothing of the background GPS data provided by the EGI. This utility receives background GPS data, heading, airspeed, altitude, groundspeed, etc. from the PSSS and creates a smoothed solution for use by other common services or portable applications.

Navigation Utilities

The Navigation Utilities (Nav Utils) are services that provide conversion utilities, and calculations for ownership position amongst other utilities needed by the portable applications.

User Interface Service

The User Interface Service (UIS) communicates to the DTD Manager what data is needed from the DTD.

8.2.4.6 IOSS Interface

The IOSS Interface is instantiated within an IOSS UoC which contains the business logic for the specific transport mechanism. This is only used between IOSS and PSSS.

8.2.4.6.1 MIL-STD-1553 Service to EGI, VOR/DME/ILS, and TACAN Managers

The MIL-STD-1553 I/O Service sends/receives EGI, VOR/DME/ILS, and TACAN data, per the pertinent MIL-STD-1553 ICD, to/from the MIL-STD-1553 device driver included in the BSP. The MIL-STD-1553 I/O Service then sends/receives the data to the respective managers, which then make the data available to other FACE UoCs over the TS Interface using the FACE Data Architecture. The MIL-STD-1553 Service is configured by FACE Configuration Services to support the devices for the given implementation.

8.2.4.6.2 ARINC 429 Service to EGI, Radar Altimeter, and Air Data Managers

The ARINC 429 I/O service sends/receives EGI, Air Data, and Radar Altimeter data to/from the ARINC 429 device driver included in the BSP. The ARINC 429 I/O service then sends/receives the data to the respective managers, which then make the data available to other FACE UoCs over the TS Interface using the FACE Data Architecture. The ARINC 429 service is configured by FACE Configuration Services to support the devices for the given implementation.

8.2.4.6.3 Serial I/O Service to Bezel Manager

The Serial connection sends the Bezel commands to the Serial I/O Service. The Serial I/O Service then sends the command to the Bezel Manager, which then makes those inputs available to the rest of the FACE Computing Environment via the TS Interface using the FACE Data Architecture. The Serial I/O Service is configured by FACE Configuration Services to support the device for the given implementation.

8.2.4.6.4 *Analog/Discrete I/O Service to EGI and Air Data Managers*

The Analog/Discrete I/O Service requests data from the analog and discrete device drivers included in the BSP. The Analog/Discrete I/O Service then sends the command to the EGI and Air Data Managers, which then makes those inputs available to the rest of the FACE Computing Environment via the TS Interface using the FACE Data Architecture. The Analog/Discrete I/O Service is configured by FACE Configuration Services to support the devices for the given implementation.

8.2.4.6.5 *System-Level HMFM ↔ **

All services in the IOSS communicate their health status to the System-Level HMFM through the I/O Interface. Figure 18 does not show the communication lines between System-Level HMFM and all other components. This was intentional to minimize the complexity of the diagram.

8.2.4.6.6 *Configuration Services ↔ **

I/O Services are configured using the FACE Configuration Services Interface.

8.2.4.7 *TS Interface*

The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. This is only used between PCS and PSSS. All data passed through the TS Interface must be represented using the FACE Data Architecture.

The source and destination for connections are used by the TS library and are defined in the TS library configuration data. Refer to the FACE Technical Standard §3.7.5.

8.2.4.7.1 *PSSS ↔ PCS*

System-Level HMFM ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS pass health status to the System-Level HMFM through the TS Interface. Figure 18 does not show the communication lines between the System-Level HMFM and all other components. This was intentional to minimize the complexity of the diagram.

Centralized Configuration Services ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS receive configuration data from Centralized Configuration Services through the TS Interface. Figure 18 does not show the communication lines between Centralized Configuration Services and all other components. This was intentional to minimize the complexity of the diagram.

GLX Server ↔ Digital Map

The Digital Map component communicates with the GLX Server using an X11 Byte Stream over the TS Interface.

GLX Server ↔ Overlay Manager

The Overlay Manager component communicates with the GLX Server using an X11 Byte Stream over the TS Interface.

TACAN Manager ↔ Navigation Utilities

The Navigation Utilities component communicates information about TACAN inputs to/from the TACAN Manager through the TS Interface.

Bezel Manager → Overlay Manager

The Overlay Manager component receives information about bezel inputs from the Bezel Manager through the TS Interface.

VOR/DME/ILS Manager ↔ Navigation Utilities

The Navigation Utilities component communicates information about VOR/DME/ILS inputs to/from the VOR/DME/ILS Manager through the TS Interface.

DTD Manager ↔ UIS

The UIS component communicates with the DTD Manager to send/receive data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD using the OS Interface (Ethernet).

DTD Manager ↔ DAFIF

The DAFIF component communicates with the DTD Manager to send/receive data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD using the OS Interface (Ethernet).

DTD Manager ↔ Terrain Server

The Terrain Server component communicates with the DTD Manager to send/receive data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD using the OS Interface (Ethernet).

DTD Manager → Digital Map

The Digital Map component receives data from the DTD Manager to render data to send to the GLX Server using the TS Interface.

Air Data Manager → Terrain Server

The Terrain Server component receives airspeed and altitude data from the Air Data Manager using the TS Interface.

Air Data Manager → Navigation Utilities

The Navigation Utilities component receives airspeed and altitude data from the Air Data Manager using the TS Interface.

Radar Altimeter Manager → Terrain Server

The Terrain Server component receives height above terrain data from the Radar Altimeter Manager using the TS Interface.

Radar Altimeter Manager → Navigation Utilities

The Navigation Utilities component receives height above terrain data from the Radar Altimeter Manager using the TS Interface.

EGI Manager → Terrain Server

The Terrain Server component receives navigation data from the EGI Manager using the TS Interface.

EGI Manager → Navigation Utilities

The Navigation Utilities component receives navigation data from the EGI Manager using the TS Interface.

EGI Manager → GPS Aiding

The GPS Aiding component receives navigation data from the EGI Manager using the TS Interface.

8.2.4.7.2 PCS ↔ PCS

Terrain Server → Digital Map

The Digital Map component receives terrain data from the Terrain Server component using the TS Interface.

GPS Aiding → Terrain Server

The Terrain Server component receives navigation data from the GPS Aiding component using the TS Interface.

Terrain Server → Overlay Manager

The Overlay Manager component receives terrain data from the Terrain Server component using the TS Interface.

GPS Aiding → Digital Map

The Digital Map component receives navigation data from the GPS Aiding component using the TS Interface.

GPS Aiding → Navigation Utilities

The Navigation Utilities component receives navigation data from the GPS Aiding component using the TS Interface.

DAFIF → Navigation Utilities

The Navigation Utilities component receives navaid data from the DAFIF component using the TS Interface.

8.3 Required Navigation Performance Manager Scenario

8.3.1 Introduction

Required Navigation Performance (RNP) is performance-based navigation that allows an aircraft to fly a specific path between two three-dimensionally defined points in space. For the purposes of this implementation, the RNP application receives Navigation Data (i.e., Altitude, Heading, Airspeed, Lat/Lon), and provides a flight plan vector and navigation alerts to the pilot through a CDS and CDU. This scenario begins after the DTD has already been loaded onto the aircraft. This excludes the pre-flight and post-flight actions.

8.3.2 Scope

8.3.2.1 Operational View

Figure 20 shows the use cases defined in this section, as well as the actors involved in the use cases.

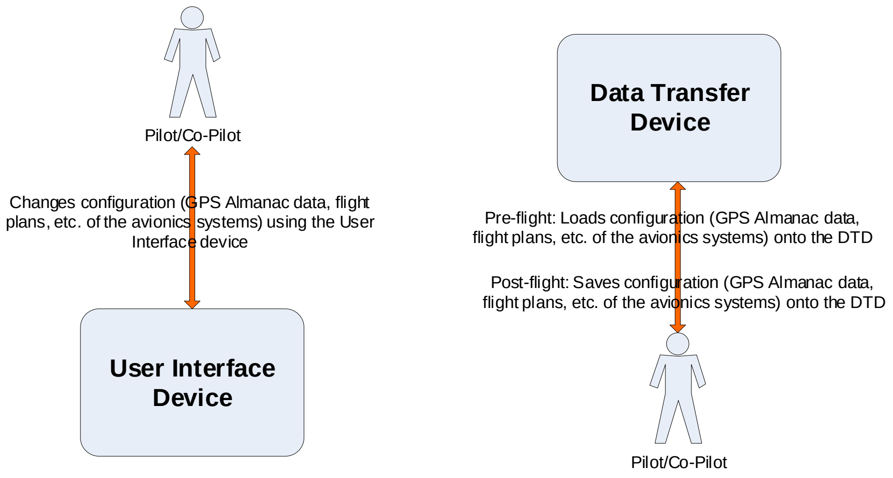

Pilot/Co-Pilot
Changes configuration (GPS Almanac data, flight plans, etc. of the avionics systems) using the User Interface device
User Interface Device
Data Transfer Device
Pre-flight: Loads configuration (GPS Almanac data, flight plans, etc. of the avionics systems) onto the DTD
Post-flight: Saves configuration (GPS Almanac data, flight plans, etc. of the avionics systems) onto the DTD
Pilot/Co-Pilot

Figure 20: Required Navigation Performance Operational View

The user (Pilot/Co-Pilot) interfaces with the EGI by interactions with the User Interface Device. The user is capable of changing EGI configuration and source selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the Air Data Computer by interactions with the User Interface Device. The user is capable of changing Air Data Computer operational configuration and source selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the VOR/DME/ILS by interactions with the User Interface Device. The user is capable of changing VOR/DME/ILS operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the TACAN by interactions with the User Interface Device. The user is capable of changing TACAN operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces with the Radar Altimeter by interactions with the User Interface Device. The user is capable of changing Radar Altimeter operational configuration data and source, channel, and frequency selection using this user input device.

The user (Pilot/Co-Pilot) interfaces through the User Interface Device with the DTD by loading presets and operational configuration files. The user interfaces through the User Interface Device with the Data Transfer Device to save presets and configuration files.

The user (Pilot/Co-Pilot) controls overlays, digital map source, map type, and operational configuration through the bezels on the display.

8.3.2.2 *Functional View*

The following view shown in Figure 21 begins after the DTD has already been loaded onto the aircraft. This excludes the pre-flight and post-flight actions.

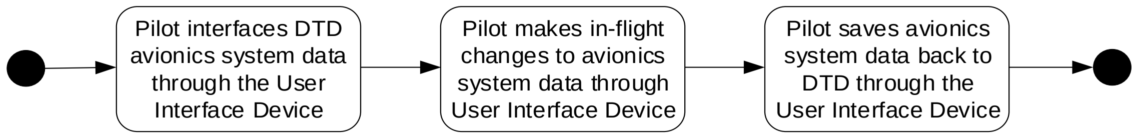

Pilot interfaces DTD avionics system data through the User Interface Device
Pilot makes in-flight changes to avionics system data through User Interface Device
Pilot saves avionics system data back to DTD through the User Interface Device

Figure 21: Required Navigational Performance Functional View

8.3.2.3 *Physical View*

Figure 22 shows the example FACE Computing Environment and associated UoCs separated by their respective FACE Segments.

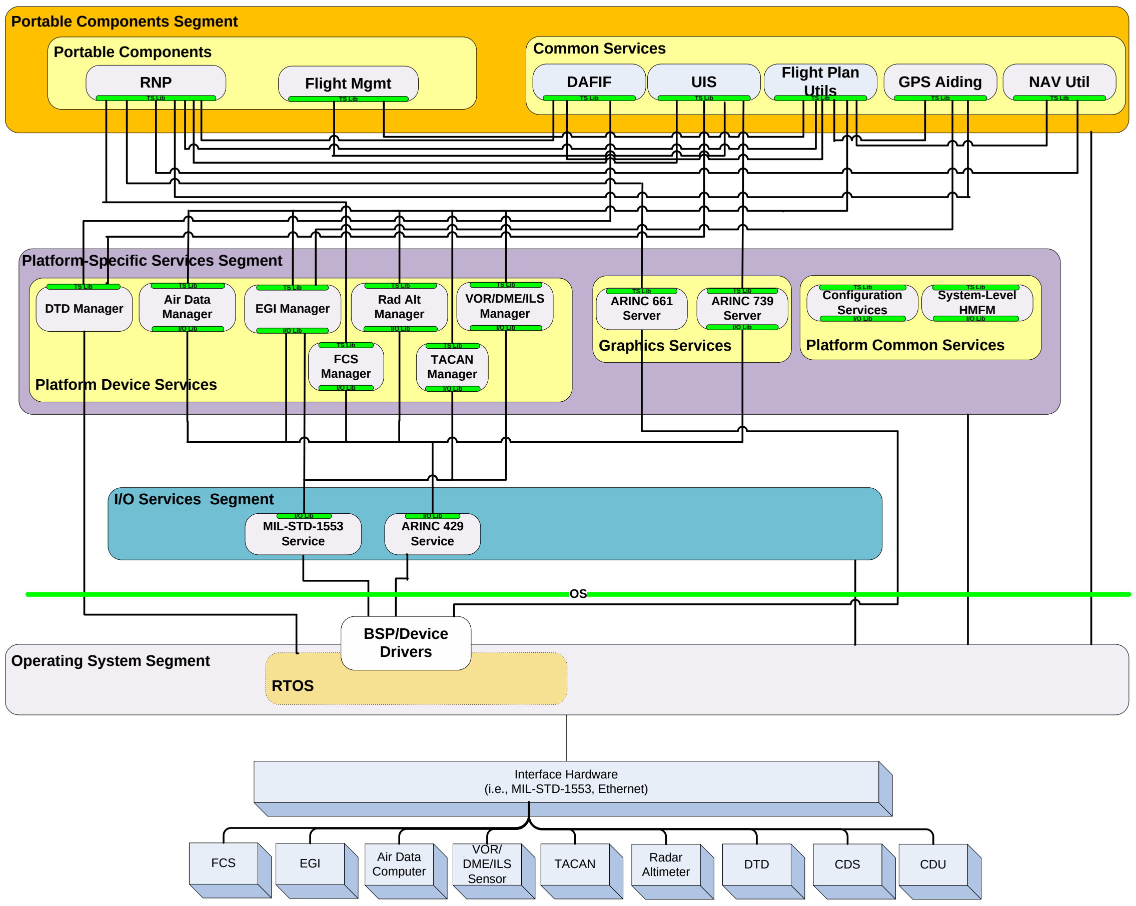


Figure 22: Required Navigational Performance Physical View

8.3.2.4 *Partitioned View*

Figure 23 features a partitioned view of the FACE Computing Environment and associated UoCs defined in this example.

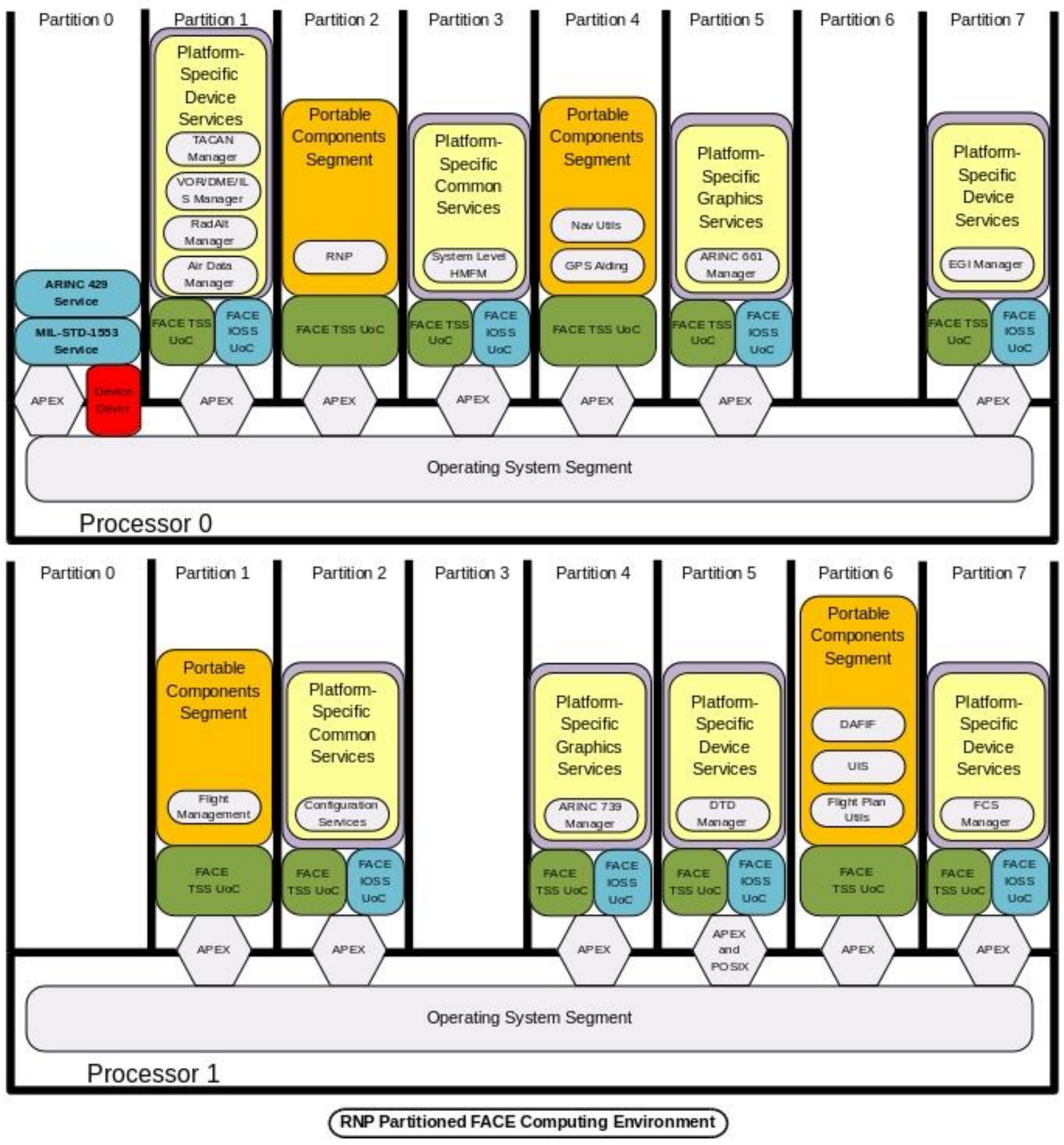


Figure 23: Required Navigational Performance Partition View

8.3.3 Assumptions

The devices that are depicted in this scenario are a CDU, CDS, an EGI, an Air Data Computer, a VOR/DME/ILS sensor, a TACAN sensor, a Radar Altimeter, and a Flight Control System (FCS). The CDU uses the ARINC 739 protocol and is connected over an ARINC 429 bus. The CDS uses the ARINC 661 protocol and is connected over Ethernet. An EGI is connected over a MIL-STD-1553 and ARINC 429 bus. An Air Data Manager is connected over an ARINC 429 bus. A VOR/DME/ILS and a TACAN sensor are connected over a MIL-STD-1553 bus. A Radar Altimeter and FCS System are connected over an ARINC 429 bus.

8.3.3.1 Profile

The software is all run on two processors in 13 different partitions. The FACE Safety Profile is being used with both ARINC 653 (APEX) partitions and POSIX partitions.

8.3.3.2 Performance

The FCS has real-time requirements. UIS must provide suitable human response times.

8.3.4 Computing Environment

8.3.4.1 Operating System Segment

The OSS provides and controls access to the computing platform for the other FACE Segments.

8.3.4.1.1 The POSIX Standard

The OSS implements the FACE Safety Profile for the POSIX standard as defined in the FACE Technical Standard §3.2.1.4.

8.3.4.1.2 ARINC 653

The OSS implements the FACE Safety Profile for ARINC 653 (APEX) as defined in the FACE Technical Standard §3.2.1.4.

8.3.4.1.3 Board Support Package/Device Drivers

The OSS provides the appropriate BSP and/or device drivers for the MIL-STD-1553, ARINC 429 I/O, and Ethernet interfaces.

8.3.4.2 I/O Services Segment

The IOSS provides a common set of interfaces for I/O devices using the FACE I/O Services Interface.

8.3.4.2.1 MIL-STD-1553 Service

The MIL-STD-1553 service normalizes the interface between the BSP and software component(s) communicating with the platform device over the MIL-STD-1553 bus.

8.3.4.2.2 ARINC 429 Service

The ARINC 429 service normalizes the interface between the BSP and software component(s) communicating with the platform device over ARINC 429.

8.3.4.3 Platform-Specific Services Segment

The PSSS allows for platform-unique software component combinations corresponding to the specific platforms requirements.

8.3.4.3.1 Platform-Specific Device Services

The Platform-Specific Device Services include components that provide control for, receive data from, and send data to platform devices or external systems.

EGI Manager

The EGI manager:

- Communicates with the EGI device over the FACE IOSS Interface according to the EGI device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the EGI device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the EGI device (PBIT, CBIT, IBIT)

  — Provides command and control of the EGI device

Air Data Manager

The Air Data manager:

- Communicates with the Air Data Computer device over the FACE IOSS Interface according to the Air Data Computer device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the Air Data Computer device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the Air Data Computer device (PBIT, CBIT, IBIT)

  — Provides command and control of the Air Data Computer device

Radar Altimeter Manager

The Radar Altimeter manager:

- Communicates with the Radar Altimeter device over the FACE IOSS Interface according to the Radar Altimeter device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the Radar Altimeter device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the Radar Altimeter device (PBIT, CBIT, IBIT)

  — Provides command and control of the Radar Altimeter device

TACAN Manager

The TACAN manager:

- Communicates with the TACAN device over the FACE IOSS Interface according to the TACAN device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the TACAN device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the TACAN device (PBIT, CBIT, IBIT)

  — Provides command and control of the TACAN device

VOR/ILS/DME Manager

The VOR/ILS/DME manager:

- Communicates with the VOR/ILS/DME device over the FACE IOSS Interface according to the VOR/ILS/DME device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the VOR/ILS/DME device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the VOR/ILS/DME device (PBIT, CBIT, IBIT)

  — Provides command and control of the VOR/ILS/DME device

Flight Control System Manager

The FCS manager:

- Communicates with the FCS device over the FACE IOSS Interface according to the TACAN device's ICD
- Receives data from the other FACE UoCs and translates the data from the USM into messages the FCS device can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the FCS device (PBIT, CBIT, IBIT)

  — Provides command and control of the FCS device

DTD Manager

The DTD manager:

- Communicates with the DTD using discrete inputs over the FACE IOSS Interface

- Receives data from the other FACE UoCs and translates the data from the USM into messages the DTD can understand
- Translates inputs from the DTD into the USM for communication to other software components in the FACE Computing Environment:

  — Provides health and status of the DTD (PBIT, CBIT, IBIT)

  — Provides command and control of the DTD device

8.3.4.3.2 *Platform-Specific Graphics Services*

ARINC 739 Manager

The ARINC 739 manager:

- Provides a standardized interface for client software components to display text-based information on a CDU
- Manages the currently active subsystems
- Controls the subsystem menu display
- Provides subsystem fault detection
- Provides an abstraction for non-standard ARINC 739 hardware

ARINC 661 Manager

The ARINC 661 manager:

- Provides a standardized interface for client software components to display graphic information on a CDS
- Manages the currently active subsystems
- Controls the subsystem display
- Provides subsystem fault detection
- Provides an abstraction for non-standard ARINC 661 hardware

8.3.4.3.3 *Platform-Specific Common Services*

The Platform-Specific Common Services provides common services to all other FACE Reference Architecture segments per system requirements. The Platform-Specific Common Services support Centralized Configuration Services, Logging, DPM Services, Streaming Media Services, and System-level Health Monitoring as defined in the FACE Technical Standard §3.6.3.1.4, §3.6.3.1.1, §3.6.3.1.2, §3.6.3.1.3, and §3.6.3.1.5, respectively.

8.3.4.4 Transport Services Segment

8.3.4.4.1 Components

In this implementation, there are no stand-alone services needed to handle the TSS. The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. Refer to Section 8.3.4.7 for more implementation-specific details.

8.3.4.5 Portable Components Segment

The PCS is a portion of a FACE solution whose contents are entirely independent from other FACE Segments. PCS components exclusively use the TS Interface for data exchanges.

8.3.4.5.1 Portable Applications

Required Navigation Performance Manager

The Required Navigation Performance (RNP) Manager is a portable application designed to calculate the changes needed for the FCS to fly a specific path given current control system, GPS, flight plan, and navigation data as inputs. The RNP Manager provides data to the FCS Manager to control the aircraft and to the UIS to be displayed.

Flight Management

The Flight Management is a portable application designed to manage the flight plan and provide output to the UIS to be displayed given flight plan data as input.

8.3.4.5.2 Common Services

DAFIF Service

The DAFIF Service provides airport and navaid data given ownership position and inputs from the DTD Manager.

User Interface Service

The UIS is a configurable ARINC 739 client software component which provides a service for capability-based software components to display and format information on MCDU displays. The UIS communicates with a ARINC 739 Graphics Service (in PSSS) using an ARINC 739 command stream over the TS Interface, as described in the FACE Technical Standard §3.12.9. The UIS also communicates to the DTD Manager what data is needed from the DTD. The UIS communicates with other FACE UoCs over the TS Interface using the FACE Data Architecture. The actual information (e.g., menus, text fields, options) which is displayed on the CDU and the associated name/value pairs which are exchanged with other FACE UoCs are fully configurable, and are stored/retrieved using the FACE Centralized Configuration Services.

Flight Planning Utilities

The Flight Planning utilities are services that provide conversion utilities and generated flight plans amongst other utilities needed by the portable applications.

GPS Aiding

The GPS Aiding service provides heading and airspeed smoothing of the background GPS data provided by the EGI. This utility receives background GPS data, heading, airspeed, altitude, groundspeed, etc. from the PSSS and creates a smoothed solution for use by other common services or portable applications.

Navigation Utilities (Nav Utils)

The Navigation Utilities are services that provide conversion utilities, and calculations for ownership position amongst other utilities needed by the portable applications.

8.3.4.6 IOSS Interface

The IOSS Interface is instantiated within an IOSS UoC which contains the business logic for the specific transport mechanism. This is only used between IOSS and PSSS.

8.3.4.6.1 MIL-STD-1553 Service to EGI, VOR/DME/ILS, and TACAN Managers

The MIL-STD-1553 I/O Service sends/receives EGI, VOR/DME/ILS, and TACAN data, per the specific ICDs, to/from the MIL-STD-1553 device driver included in the BSP. The MIL-STD-1553 I/O Service then sends/receives the data to the respective managers, which then make the data available to other FACE UoCs over the TS Interface using the FACE Data Architecture. The MIL-STD-1553 Service is configured by FACE Configuration Services to support the devices for the given implementation.

8.3.4.6.2 ARINC 429 Service to FCS, EGI, Radar Altimeter, Air Data, and ARINC 739 Managers

The ARINC 429 I/O service sends/receives FCS, EGI, Radar Altimeter, Air Data, and ARINC 739 data to/from the ARINC 429 device driver included in the BSP. The ARINC 429 I/O service then sends/receives the data to the respective managers, which then make the data available to other FACE UoCs over the TS Interface using the FACE Data Architecture. The ARINC 429 service is configured by FACE Configuration Services to support the devices for the given implementation.

*8.3.4.6.3 System-Level HMFM ↔ **

All services in the IOSS communicate their health status to the System-Level HMFM through the IOSS Interface. Figure 22 does not show the communication lines between FACE Configuration Services and all other components. This was intentional to minimize the complexity of the diagram.

*8.3.4.6.4 Configuration Services ↔ **

I/O Services are configured using the FACE Configuration Services Interface.

8.3.4.7 TS Interface

The TS Interface is instantiated within a TS library which contains the business logic for the specific transport mechanism. This is only used between PCS and PSSS. All data passed through the TS Interface uses the FACE Data Architecture to represent the data exchanged.

The source and destination for connections are used by the TS library and are defined in the TS library configuration data. See the FACE Technical Standard §3.7.

8.3.4.7.1 PSSS ↔ PCS

System-Level HMFM ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS pass health status to the System-Level HMFM through the TS Interface. Figure 22 does not show the communication lines between the System-Level HMFM and all other components. This was intentional to minimize the complexity of the diagram.

Centralized Configuration Services ↔ *

All Portable Applications, Managers, and Services in the PCS and PSSS receive configuration data from Centralized Configuration Services through the TS Interface. Figure 22 does not show the communication lines between Centralized Configuration Services and all other components. This was intentional to minimize the complexity of the diagram.

ARINC 739 Server↔ UIS

The UIS communicates with the ARINC 739 server using an ARINC 739 command stream over the TS Interface.

ARINC 661 Server ↔ RNP Manager

The RNP Manager communicates with the ARINC 661 server using an ARINC 611-4 command stream over the TS Interface. The ARINC 661 server communicates directly with the GPU using the OSS Interface.

VOR/DME/ILS Manager ↔ Flight Plan Utilities

The Flight Plan Utilities component communicates information about VOR/DME/ILS inputs to/from the VOR/DME/ILS Manager through the TS Interface.

TACAN Manager ↔ Flight Plan Utilities

The Flight Plan Utilities component communicates information about TACAN inputs to/from the TACAN Manager through the TS Interface.

Radar Altimeter Manager → Flight Plan Utilities

The Flight Plan Utilities component receives height above terrain data from the Radar Altimeter Manager using the TS Interface.

EGI Manager → Flight Plan Utilities

The Flight Plan Utilities component receives navigation data from the EGI Manager using the TS Interface.

Air Data Manager → Flight Plan Utilities

The Flight Plan Utilities component receives airspeed and altitude data from the Air Data Manager using the TS Interface.

FCS Manager ↔ RNP Manager

The RNP Manager communicates information about FCS inputs to/from the FCS Manager through the TS Interface.

EGI Manager → GPS Aiding

The GPS Aiding component receives navigation data from the EGI Manager using the TS Interface.

DTD Manager ↔ UIS

The UIS component communicates with the DTD Manager to send/receive data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD.

DTD Manager ↔ DAFIF

The DAFIF component communicates with the DTD Manager to send/receive data to be written to and/or read from the DTD using the TS Interface. The DTD Manager then communicates the data directly to the DTD.

8.3.4.7.2 PCS ↔ PCS

DAFIF → Flight Plan Utilities

The Flight Plan Utilities component receives navaid data from the DAFIF component using the TS Interface.

GPS Aiding → Flight Plan Utilities

The Flight Plan Utilities component receives navigation data from the GPS Aiding component using the TS Interface.

Navigation Utilities → Flight Plan Utilities

The Flight Plan Utilities component receives navigation data from the Navigation Utilities component using the TS Interface.

Flight Plan Utilities → Flight Management

This Flight Management component receives flight plan information from the Flight Plan Utilities component using the TS Interface.

DAFIF → RNP

The RNP component receives navaid data from the DAFIF component using the TS Interface.

Flight Plan Utilities → RNP

The RNP component receives flight plan data from the Flight Plan Utilities component using the TS Interface.

GPS Aiding → RNP

The RNP component receives navigation data from the GPS Aiding component using the TS Interface.

Navigation Utilities → RNP

The RNP component receives navigation data from the Navigation Utilities component using the TS Interface.

RNP → UIS

The UIS component receives display and format information from the RNP component using the TS Interface.

Flight Management → UIS

The UIS component receives display and format information from the Flight Management component using the TS Interface.

8.4 Secure Reporting Status Scenario

8.4.1 Introduction

The following example is based on Scenario B from the NSA SSE-100-1. This document is publicly available and provides useful notional architectures and scenarios that reflect security considerations for real-time systems. While this document contains a wide range of scenarios, the Scenario B Reporting Status example was selected to illustrate the security implications of the FACE Profiles against an identified use case in the public domain. It is recommended that NSA SSE-100-1 be referenced for further details not provided in the sections below.

NSA SSE-100-1 Figure 15 (as replicated below in Figure 24) depicts a Reporting-Status use case where a TS User Application on Processor #2 provides basic status log information to a TS removable memory device connected to Processor #1.

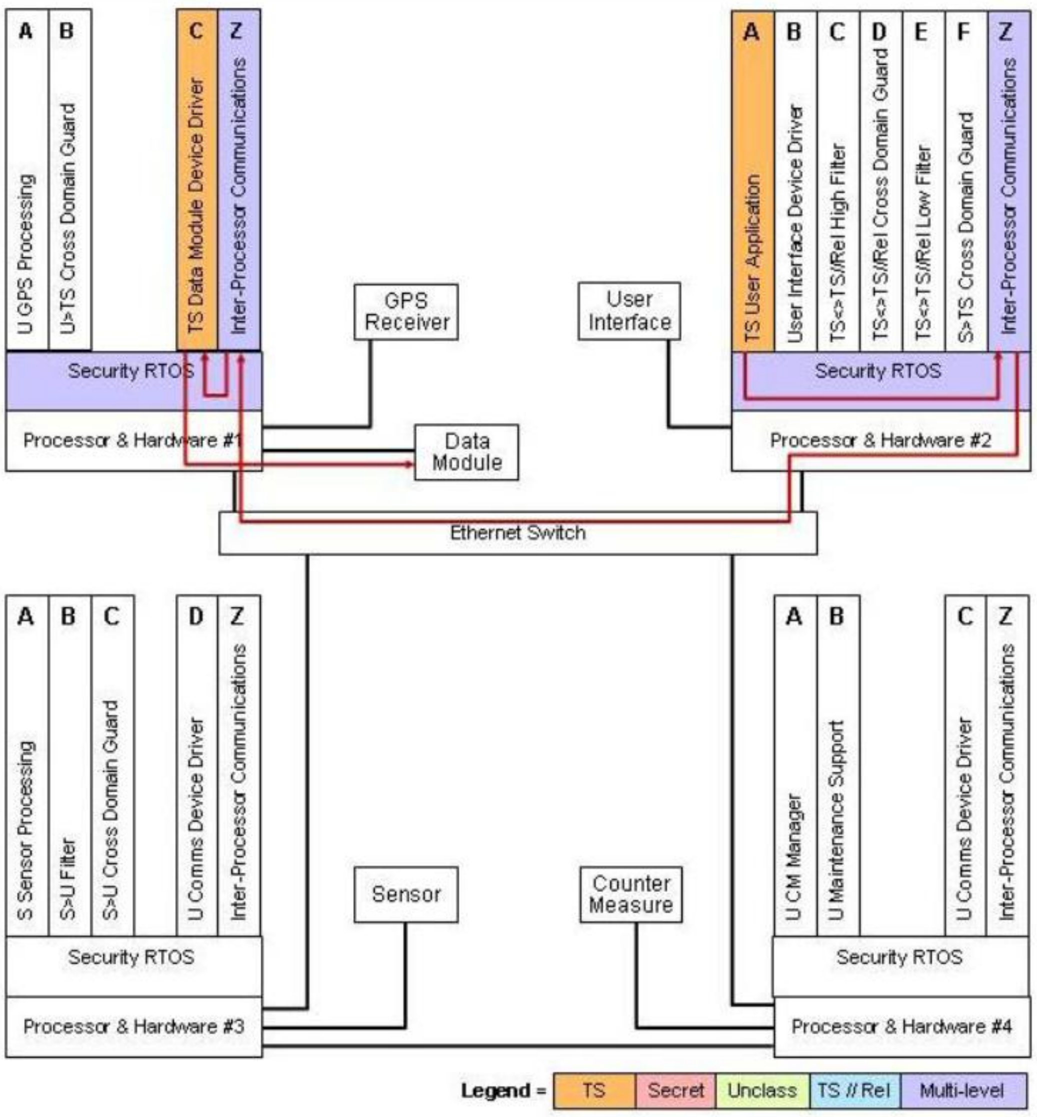


Figure 24: NSA SSE-100-1 Scenario B Reporting Status

The NSA SSE-100-1 Scenario B Reporting Status scenario reflects sending basic status log information from the TS User Application to the Data Module (NSA SSE-100-1 §8.3.3.6). This scenario begins after the TS User Application has created fused/composed a TS situational awareness picture for the user.

8.4.2 Scope

8.4.2.1 *Operational View*

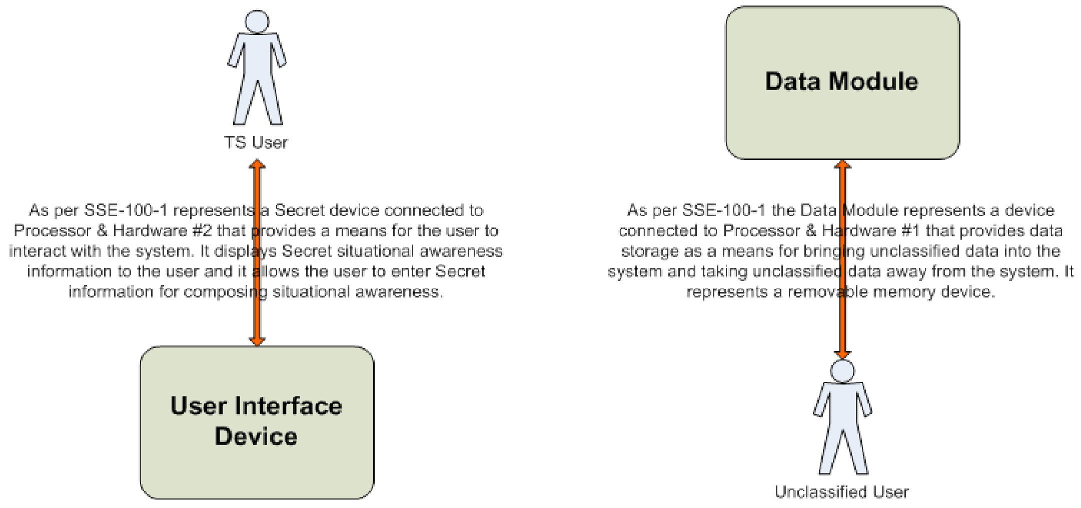

TS User
As per SSE-100-1 represents a Secret device connected to Processor & Hardware #2 that provides a means for the user to interact with the system. It displays Secret situational awareness information to the user and it allows the user to enter Secret information for composing situational awareness.
User Interface Device
Data Module
As per SSE-100-1 the Data Module represents a device connected to Processor & Hardware #1 that provides data storage as a means for bringing unclassified data into the system and taking unclassified data away from the system. It represents a removable memory device.
Unclassified User

Figure 25: Secure Reporting Status Operational View

The TS User Applications provides basic status log information to the Data Module without user inputs. The user is capable of storing a TS report on the Data Module to be extracted and analyzed at a later time.

8.4.2.2 *Functional View*

The following view shown in Figure 26 begins after the TS User Application has fused a situational awareness picture.

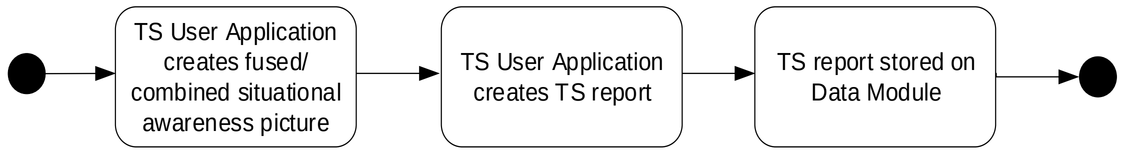

TS User Application creates fused/ combined situational awareness picture
TS User Application creates TS report
TS report stored on Data Module

Figure 26: Secure Reporting Status Functional View

8.4.2.3 *Physical View*

Figure 27 shows the example FACE Computing Environment and associated UoCs separated by their respective FACE Segments.

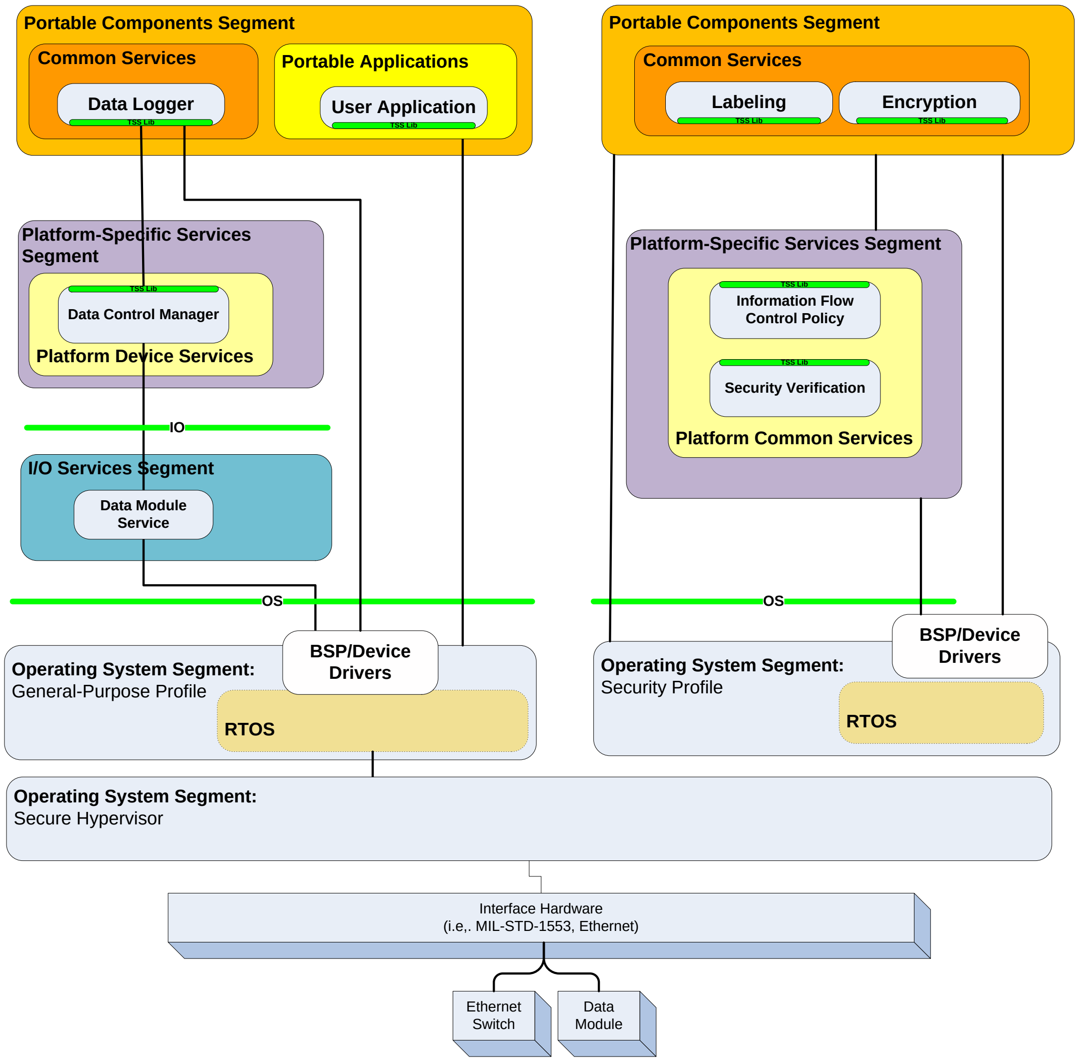


Figure 27: Secure Reporting Status Physical View

8.4.2.4 Partitioned View

Figure 28 features a partitioned view of the FACE Computing Environment and associated UoCs defined in this example.

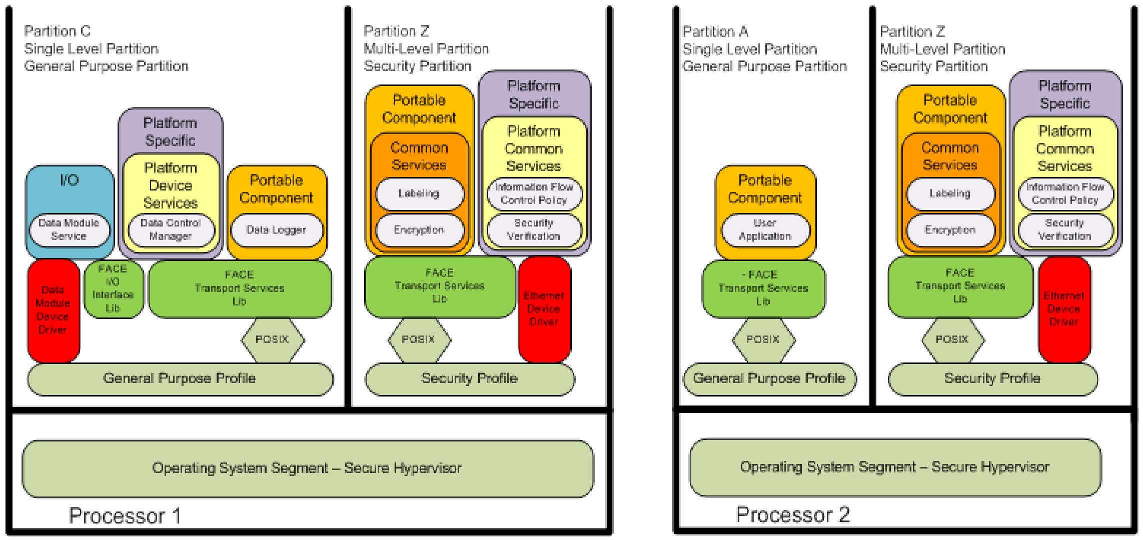


Figure 28: Secure Reporting Status Partition View

8.4.3 Assumptions

The system has reached a trusted state.

8.4.3.1 Profile

The software executes within two partitions per processor. The FACE Security Profile (POSIX standard) is being used on the inter/intra-processor communications and a FACE General Purpose Profile (POSIX standard) partition is being used for the host applications.

8.4.3.2 Performance

No strict real-time requirements.

8.4.4 Computing Environment

8.4.4.1 Operating System Segment

The OSS provides and controls access to the computing platform for the other FACE Segments.

8.4.4.1.1 The POSIX Standard

The OSS implements the FACE General Purpose Profile for the POSIX standard as defined in the FACE Technical Standard §3.2.1.5 for the single security-level partitions used for host applications. Additionally, the OSS implements the FACE Security Profile for the POSIX standard as defined in the FACE Technical Standard §3.2.1.3 for the multi-level secure partitions that perform security enforcing and/or relevant functions.

8.4.4.1.2 Board Support Package/Device Drivers

The OSS provides the appropriate BSP and/or device drivers for the data storage device I/O, and Ethernet interfaces.

8.4.4.2 I/O Services Segment

The IOSS provides a common set of interfaces for I/O devices using the FACE I/O Services Interfaces.

8.4.4.2.1 Data Module Service

The Data Module service normalizes the interface between the BSP and software component(s) communicating with the platform storage device over the data bus. While NSA SSE-100-1 is not explicit, we are assuming the storage device is connected to the platform through a standard serial (e.g., SATA) or other common avionics bus (e.g., MIL-STD-1553).

8.4.4.3 Platform-Specific Services Segment

The PSSS allows for platform-unique software component combinations corresponding to the specific platforms requirements.

8.4.4.3.1 Platform-Specific Device Services

The Platform-Specific Device Services include components that provide control for, receive data from, and send data to platform devices or external systems.

Data Control Manager

The Data Control Manager:

- Communicates with the Data Module Service over the FACE IOSS Interface
- Receives data from the other FACE UoCs and translates the data from the USM into messages the Data Module Service can understand
- Translates the ICD-defined data into the USM for communication with other FACE UoCs:

  — Provides health and status of the Data Module device (PBIT, CBIT, IBIT)

  — Provides command and control of the Data Module device

8.4.4.3.2 Platform-Specific Common Services

The Platform-Specific Common Services provides common services to all other FACE Segments per system requirements.

Information Flow Control Policy

The Information Flow Control Policy component authorizes the message flow between applications on different partitions/devices.

Security Verification

The Security Verification component ensures the message was unaltered through the Ethernet switch and that the data was not compromised.

8.4.4.4 Transport Services Segment

The TS Interface is only instantiated within a TS library which contains the business logic for the specific transport mechanism. This is only used between PCS and PSSS.

The source and destination for connections are used by the TS library and are defined in the TS library configuration data. See the FACE Technical Standard §3.7.

8.4.4.4.1 PSSS ↔ PCS

Information Flow Control Policy ↔ *

The Information Flow Control Policy component communicates with any application passing messages between partitions/devices to receive the source and destination of any messages through the TS Interface.

Security Verification ↔ *

The Security Verification component communicates with any application passing messages between partitions/devices to receive message information through the TS Interface.

Data Control Manager ← *

The Data Logger component receives TS reports and basic status log information to be written to the Data Module using the TS Interface. The Data Control Manager then communicates the data to the Data Module device using the IOSS Interface.

8.4.4.4.2 PCS ↔ PCS

Data Logger ← *

The Data Logger component receives basic status log information from any application through the TS Interface.

Data Logger ← User Application

The Data Logger component receives a TS Report from the User Application through the TS Interface.

Labeling ↔ *

The Labeling component communicates with any application to apply or remove labels to a message using the TS Interface.

Encryption ↔ *

The Encryption component communicates with any application to encrypt or decrypt a message using the TS Interface.

8.4.4.5 *Portable Components Segment*

The PCS is part of a FACE Reference Architecture whose contents are entirely independent from other FACE Segments. PCS components exclusively use the TS Interface for data exchanges.

8.4.4.5.1 *Common Services*

Encryption Service

The Encryption Service (ES) is a Security Verification component which provides a service for User Application data (data in transit) that moves from a General Purpose Profile partition flowing horizontally into a Security Profile partition through a High Robustness/Secure Hypervisor Operating System Segment Partition, and then encrypted based on the required confidential level.

Labeling Service

The Labeling Service (LS) is a Security Verification component which provides a service to ensure data confidentiality and integrity is enforced through Mandatory access controls labeling.

Data Logger Service

The Data Logger Service (DLS) is a PCS component with the responsibility of time-stamping and logging User Application data received via the TS Interface from Common Services components and transferring the Application data at specified time intervals via the TS Interface to the PSSS Data Control Manager.

8.4.4.5.2 *Portable Applications*

User Application

The User Application (UA) manages various types of data that have been designated to be moved from the General Purpose Profile partition into a Security Profile partition via the FACE TS Interface where the ES component will execute the data encryption process.

8.4.4.6 *IOSS Interface*

The IOSS Interface is only instantiated within an IOSS UoC which contains the business logic for the specific transport mechanism. This is only used between IOSS and PSSS.

8.4.4.6.1 *Data Module Service*

The Data Module IOSS UoC sends encrypted data from the Data Control Manager, a Platform Device Services component, to the Data Module device driver included in the BSP. The Data Module device driver at specified intervals will send encrypted data to the Data Module.

8.4.4.7 *TS Interface*

The TS Interface is only instantiated within a TS library which contains the business logic for the specific transport mechanism. This is only used between PCS and PSSS.

The source and destination for connections are used by the TS library and are defined in the TS library configuration data. See the FACE Technical Standard §3.7.

8.4.4.7.1 *PCS ↔ PCS*

User Application ↔ Common Services

User Application data from Partition A (Single Level/General Purpose Partition) is transferred via the TS Interface to Partition Z (Multi-Level/Security Partition) PCS Common Services. Figure 28 does not show the communication lines between Partition Z PCS Common Services components. This was intentional to minimize complexity of the diagram.

8.4.4.7.2 *PCS ↔ PSSS*

Common Services ↔ Platform Common Services

Application data from Partition Z PCS Common Services is transferred via TS Interface to PSSS Platform Common Services. Figure 29 shown in a subsequent notional data flow section below does not show the communication lines between Partition Z PCS Common Services components and PSSS Platform Common Services components. This was intentional to minimize complexity of the diagram.

8.4.4.7.3 *PSSS ↔ Ethernet Device Driver*

Platform Common Services ↔ Ethernet Device

Application data is transferred via the TS Interface to the Ethernet Device Driver BSP. Per specified interval, the Application data will be sent via the TS Interface to Partition C (Single Level/General Purpose Partition) PCS.

8.4.4.7.4 *PCS ↔ PSSS*

Data Logger ↔ Platform Device Services

Application data transferred via the TS Interface to the PCS Data Logger component is time-stamped and logged before being transferred via the TS to Platform Device Services. Figure 29 does not show the communication lines between Partition C PCS Data Logger and Platform Device Services components. This was intentional to minimize complexity of the diagram.

8.4.4.8 *Notional Data Flow View*

To further describe this FACE Secure Reporting Status example scenario, Figure 29 depicts the partitions and data flow within Processor & Hardware #2, flowing between Operating System Segments from partition A to partition Processor & Hardware #1. Partitions are represented by the vertical barriers between Operating System Segments and the classification of the partitions is included within the components in each FACE Segment. The partitions that support the movement of data within a segment are shown at different classification levels at the beginning of their name, as well as symbols to denote the direction in which the data is flowing.

On Processor & Hardware #2, the Operating System Segment for Partition A provides the data flow from a User Application (TS) through the Transport Services Segment to Partition Z. The data moves from a General Purpose Profile partition flowing horizontally into a Security Profile partition through a High Robustness/Secure Hypervisor Operating System (Operating System Segment Partition Z). After proper hardening of data (labeling and/or encryption through the Security Verification component) the TS-level information can then travel through a COTS Ethernet switch to Processor & Hardware #1.

The check between Security Verification components on the separate processor hardware is needed to ensure the data was properly routed and unaltered through the Ethernet switch and that the TS data was not compromised. Processor & Hardware #1 then reverses the process through the Transport Services Segment in the High Robustness Partition Z which subsequently sends the validated data via the TSS and OSS to a Data Logger application in Partition C. The Data Logger application then utilizes a platform-specific Data Control Manager to send the data to the external Data Module using a Data Module Service in the I/O Services Segment of Partition C.

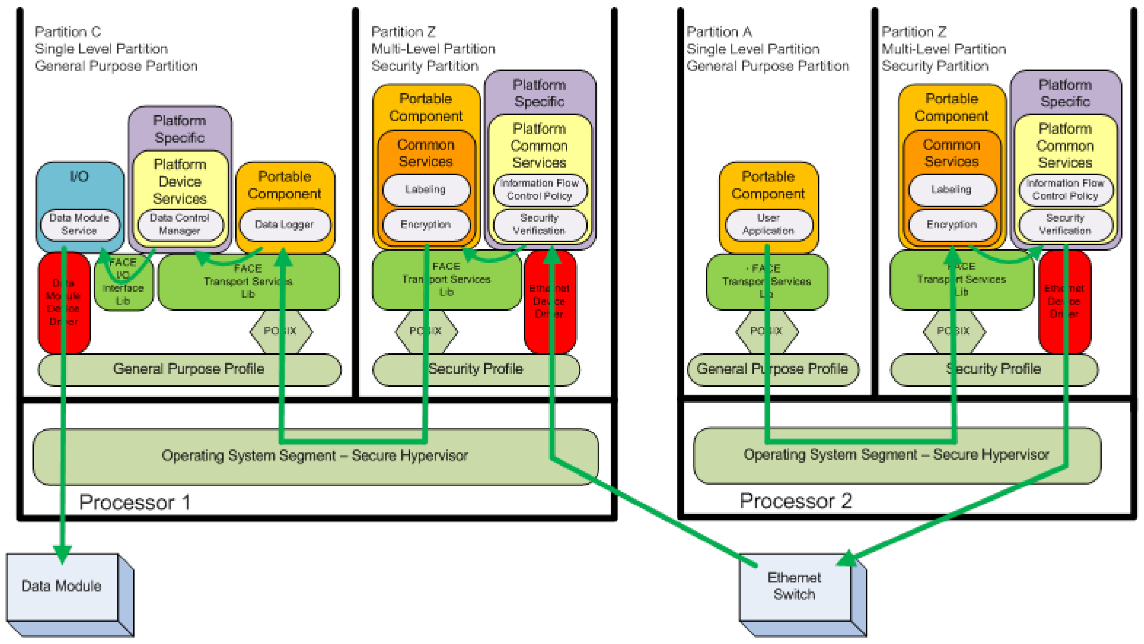


Figure 29: Secure Reporting Status Data Flow View

A Acronyms

The following acronyms are used within this Guide:

Acronym	Definition
3D	3-Dimensional
AC	Aircraft
APEX	Application Executive
API	Application Programming Interface
AQS	Airworthiness Qualification Specification
AQSR	Airworthiness Qualification Substantiation Report
ARINC	Aeronautical Radio Inc.
ARP	Aerospace Recommended Practices
AWR	Airworthiness Release
BIT	Built-In Test
BLOS	Beyond Line Of Sight
BSP	Board Support Package
CBIT	Continuous Built-In Test
CC	Common Criteria
CDRL	Contract Data Requirements Lists
CDS	Cockpit Display System
CDU	Control and Display Unit
CM	Configuration Management
CNSS	Committee of National Security Systems
COE	Common Operating Environment
COMM	Communications Management
CORBA	Common Object Request Broker Architecture

Acronym	**Definition**
COTS	Commercial Off-The-Shelf
CPI	Critical Program Information
CRC	Cyclical Redundancy Checking
CSP	Component State Persistence
CWE	Common Weakness Enumeration
DAFIF	Digital Aeronautical Flight Information File
DAL	Design Assurance Level
DASD (SE)	Deputy Assistant Secretary of Defense for Systems Engineering
DDS	Data Distribution Service
DF	Definition File
DID	Data Item Description
DigMap	Digital Map
DLS	Data Logger Service
DMA	Dynamic Memory Access
DME	Distance Measuring Equipment
DO	Document
DoD	Department of Defense
DoDI	Department of Defense Instruction
DPM	Device Protocol Mediation
DTD	Data Transfer Device
DTED	Digital Terrain Elevation Data
ECC	Error Checking and Correction
EGI	Embedded Global Positioning System/Inertial Navigation System
EIA	Electronic Industries Alliance
ES	Encryption Service
EXT	System Data Model Extensions

Acronym	**Definition**
FAA	Federal Aviation Administration
FACE	Future Airborne Capability Environment
FCS	Flight Control System
FHA	Functional Hazard Assessment
FISMA	Federal Information Security Management Act
FMEA	Failure Modes Effects Analysis
GLX	OpenGL Extension to the X Window System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HMFM	Health Monitoring/Fault Management
HMI	Human Machine Interface
I/O	Input/Output
IA	Information Assurance
IBIT	Initiated Built-In Test
ICAO	International Civilian Aviation Organization
ICD	Interface Control Document
ID	Identification, Identifier
IDD	Interface Design Description
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ILS	Instrument Landing System
IOMM	I/O Message Model
IOS	Input/Output Services
IOSS	Input/Output Services Segment
IPC	Inter-Process Communication
IRS	Interface Requirements Specification

Acronym	**Definition**
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
IT	Information Technology
LCM	Life Cycle Management
LOS	Line Of Sight
LS	Labeling Service
MBE	Model-Based Engineering
MCDU	Multi-purpose Control and Display Unit
MIL-STD	Military Standard
MISRA	Motor Industry Software Reliability Association
MMU	Memory Management Unit
MVC	Model View Controller
NAS	National Airspace
NAVSEA	Naval Sea Systems Command
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NSTISSP	National Security Telecommunications and Information Systems Security Policy
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OSS	Operating System Segment
OWASP	Open Web Application Security Project
PBIT	Periodic Built-In Test
PCS	Portable Components Segment
PDS	Platform-Specific Device Service
PFD	Primary Flight Display
PKI	Public Key Infrastructure

Acronym	**Definition**
POD	Partially Ordered Dependency
POSIX	Portable Operating System Interface
PPP	Program Protection Plan
PSAC	Plan for Software Aspects of Certification
PSSA	Preliminary System Safety Assessment
PSSS	Platform-Specific Services Segment
QA	Quality Assurance
RFC	Request for Comments
RMF	Risk Management Framework
RNP	Required Navigation Performance
RTCA	Radio Technical Commission for Aeronautics
RTOS	Real-Time Operating System
SA	Safety Assessment
SAE	Society of Automotive Engineers
SAS	Software Accomplishment Summary
SEI	Software Engineering Institute
SSA	System Safety Assessment
SSE	Systems Security Engineering
TACAN	Tactical Air Navigation
TLS	Transport Layer Security
TS	Transport Services
TSS	Transport Services Segment
TWG	Technical Working Group
UA	User Application
UHF	Ultra High Frequency
UIS	User Interface Service

Acronym	**Definition**
UoC	Unit of Conformance
UoP	Unit of Portability
USM	UoP Supplied Model
VHF	Very High Frequency
VOR	Very High Frequency (VHF) Omni-directional Radio-Range
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition

Index

Airworthiness Authority 49
AQS 64
AQSR 64
ARINC 661 .... 19, 22
ARINC 739 26
artifacts 63
Audit Log 90
AWR 53
CDRL 61
CDS 19
CDU 26
Centralized Configuration Services 14
Centralized Logging Service 11
COE 1
COMM 98
CPI 92
CRC 72
cryptography 90
CSP 5, 8
CWE 96
DAFIF 114
DAL 51
dead code 69
DF 23
DID 60
DigMap 107
Display Management Services 27
Display Manager 20
DPM 12
ECC 72
EGL Compositor 21
FAA 54
FACE approach 1
FACE Profiles 16
FACE Registry 63
FACE Safety Profile 50
FACE Security Profile 87
FHA 51
FMEA 52
GLX 114
GPU 21
graphics requirements 37
Graphics Services 18
HMI 18
IDD 60
IETF RFC 5424 11
IOMM 7
IRS 60
ISO/IEC 27034 96
LCM 5, 8
MBE 76
MISRA 56
MMU 83
MVC 18, 22
NAVSEA 49
NIST SP 800-160 .... 87, 97
NIST SP 800-37 96
NIST SP 800-53 88
NSA SSE-100-1 94
OpenGL 24
OSGi 5
OWASP 96
partitioning 65
PCS 2
PCS UoC 3
PCS UoC configurability 4
PKI 90
portability 3
PPP 97
privileged process 91
PSAC 64
PSCS UoC 10
PSDS UoC 9
PSGS UoC 15
PSSA 51
PSSS 2
PSSS UoC 7
qualification 64, 68
Ravenscar 56
regression testing 65
reusability 3
risk 87
RMF 96
RNP 119
run-times 77
safety guidance 49
SAS 62
security 87
security controls 88
security guidance 80
Security Profile 80
Security Transformation 92
software reuse 57
source code reuse 58
SSA 51
Streaming Media 13
System-level Health Monitoring 15
test programs 69

test reuse 62
traceability 60
UA 20
UoC 2
XSD 23