The Open Group Guide

Integrator's Guide for FACE™ Technical Standard, Edition 3.x

THE Open GROUP

Distribution Statement A – Approved for Public Release

Distribution Unlimited – (Control Number PR20200298)

Copyright © 2020, The Open Group L.L.C. for the benefit of the FACE Consortium Members. All rights reserved.

The Open Group hereby authorizes you to use this document for any purpose, PROVIDED THAT any copy of this document, or any part thereof, which you make shall retain all copyright and other proprietary notices contained herein.

This document may contain other proprietary notices and copyright information.

Nothing contained herein shall be construed as conferring by implication, estoppel, or otherwise any license or right under any patent or trademark of The Open Group or any third party. Except as expressly provided above, nothing contained herein shall be construed as conferring any license or right under any copyright of The Open Group.

Note that any product, process, or technology in this document may be the subject of other intellectual property rights reserved by The Open Group, and may not be licensed hereunder.

This document is provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Any publication of The Open Group may include technical inaccuracies or typographical errors. Changes may be periodically made to these publications; these changes will be incorporated in new editions of these publications. The Open Group may make improvements and/or changes in the products and/or the programs described in these publications at any time without notice.

Should any viewer of this document respond with information including feedback data, such as questions, comments, suggestions, or the like regarding the content of this document, such information shall be deemed to be non-confidential and The Open Group shall have no obligation of any kind with respect to such information and shall be free to reproduce, use, disclose, and distribute the information to others without limitation. Further, The Open Group shall be free to use any ideas, concepts, know-how, or techniques contained in such information for any purpose whatsoever including but not limited to developing, manufacturing, and marketing products incorporating such information.

If you did not obtain this copy through The Open Group, it may not be the latest version. For your convenience, the latest version of this publication may be downloaded at www.opengroup.org/library.

The Open Group Guide

Integrator's Guide for FACE™ Technical Standard, Edition 3.x

ISBN: 1-947754-63-8

Document Number: G20D

Published by The Open Group, September 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

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 750 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 the Integrator's Guide for the FACE™ Technical Standard, Edition 3.x It is designed to provide integrators of FACE Units of Conformance (UoC) with guidance and considerations regarding FACE UoC integration.

Trademarks

ArchiMate, DirecNet, Making Standards Work, Open O logo, Open O and Check Certification logo, Platform 3.0, The Open Group, TOGAF, UNIX, UNIXWARE, and the Open Brand X logo are registered trademarks and Boundaryless Information Flow, Build with Integrity Buy with Confidence, Commercial Aviation Reference Architecture, Dependability Through Assuredness, Digital Practitioner Body of Knowledge, DPBoK, EMMM, FACE, the FACE logo, FHIM Profile Builder, the FHIM logo, FPB, Future Airborne Capability Environment, IT4IT, the IT4IT logo, O-AA, O-DEF, O-HERA, O-PAS, Open Agile Architecture, Open FAIR, Open Footprint, Open Process Automation, Open Subsurface Data Universe, Open Trusted Technology Provider, OSDU, Sensor Integration Simplified, SOSA, and the SOSA logo are trademarks of The Open Group.

DDS is a registered trademark of Object Management Group, Inc. in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds.

POSIX is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

All other brands, company, and product names are used for identification purposes only and may be trademarks that are the sole property of their respective owners.

Acknowledgements

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

Principal Authors

- Christopher Crook, U.S. Army PEO Aviation
- John Gray, U.S. Army PEO Aviation
- Andre Odermatt, Real-Time Innovations (RTI)
- Stephen Simi, TES-SAVi

Additional Contributors

- Matt Drew, NAVAIR
- Patrick Huyck, Green Hills Software
- Bill Kinahan, SKAYL
- Joel Sherrill, CCDC Aviation & Missile Center
- Alicia Taylor, U.S. Army PEO Aviation

Referenced Documents

The following documents are referenced in this Guide.

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

- ARINC Specification 653: Avionics Application Software Standard Interface
- AV-2: FACE™ Glossary of Terms and Definitions, Edition 3.0.1, The Open Group Guide (G194), August 2019, published by The Open Group; refer to: www.opengroup.org/library/g194
- FACE™ Software Supplier Getting Started Guide, Version 1.0, The Open Group Guide (G173), August 2017, published by The Open Group; refer to: www.opengroup.org/library/g173
- FACE™ Technical Standard, Edition 3.0, The Open Group Standard (C17C), November 2017, published by The Open Group; refer to: www.opengroup.org/library/c17c
- Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 1: General Guidance, The Open Group Guide (G209), May 2020, published by The Open Group; refer to: www.opengroup.org/library/g209
- Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 2: Computing Environment, The Open Group Guide (G209), May 2020, published by The Open Group; refer to: www.opengroup.org/library/g209
- Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 3: Data Architecture, The Open Group Guide (G209), May 2020, published by The Open Group; refer to: www.opengroup.org/library/g209

1 Introduction

1.1 Objective

A primary objective of the FACE Technical Standard is software portability, such that one reusable FACE UoC can be instantiated, with compile-time modifications, onto multiple aviation aircraft and executed to provide reusable capabilities to the aircraft.

This Integrator's Guide for FACE Technical Standard, Edition 3.x identifies the processes for the integration of a FACE conformant UoC. It describes the common Stakeholder roles and processes required to integrate FACE UoCs onto target aviation platforms. While this document is unable to describe all integration use cases, subsections within Chapter 3 should suffice for most integration cases. The FACE Integration Model, a descriptive tool to aid integration activities, is described in Chapter 4.

In a similar fashion to the FACE Software Supplier Getting Started Guide, this document introduces integration steps and, where available, provides links to reference materials for additional integration support. Appendix A provides an integration example.

Integration of a FACE UoC does not supplant software integration procedures for a particular platform or host system. It does, however, involve certain prerequisites that are particular to the FACE Technical Standard, which are not covered by normal software integration practices.

Note: Readers should have an understanding of the FACE Technical Standard, and experience integrating software components.

1.2 FACE Terms

The terms detailed in this section are used throughout this document. They represent important definitions with respect to integration of FACE UoCs and are defined in the FACE AV-2 (see Referenced Documents).

1.2.1 Unit of Conformance (UoC)

A software component or domain-specific data model designed to meet the applicable requirements defined in the FACE Technical Standard. It is referenced as a UoC at any point in its development and becomes a FACE Certified UoC upon completion of the FACE Conformance process.

1.2.2 FACE Computing Environment

A generic concept instantiated for a particular system under development. It includes all elements of the FACE Reference Architecture necessary to deploy FACE conformant components. The FACE Computing Environment is composed of the software infrastructure (Transport Services, Operating System, and I/O Services Segments) and the Platform-Specific Services required by the FACE components.

1.3 Integration

The FACE Technical Standard does not provide a definition for FACE integration; nor does the FACE AV-2. However, for purposes of this document, Integration is considered to be the process for adding a FACE UoC to an existing FACE Computing Environment and resolving all dependencies required for its execution. Chapter 2 illustrates the implied Stakeholder relationships with explicit process flow through the integration process. The integration process described in this document is limited to that shown within the scope of the dotted lines. Other external Stakeholder relationships are identified for pictorial completeness.

1.4 Primary FACE Documents

FACE publications are referenced throughout this document. Their purpose is to provide readers with direct references to either guidance, or to areas in the FACE Technical Standard that require further attention.

1.4.1 FACE Technical Standard

The FACE Technical Standard is the "keystone" document of the FACE Consortium. It embodies a set of requirements and descriptions and uses industry standards for distributed communications, programming languages, graphics, operating systems, and other areas as appropriate.

The FACE Technical Standard defines the FACE Reference Architecture, which consists of five logical segments where variance occurs. Segments are abstracted from one another using defined interfaces. The FACE Technical Standard defines requirements for implementing these segments, and for those documenting the communication between UoCs as specified by the FACE Data Architecture. The FACE Data Architecture consists of a set of Data Model Language bindings and rules for creating Unit of Portability (UoP) Supplied Models (USM).

1.4.2 Reference Implementation Guide (RIG) for the FACE Technical Standard

The RIG provides guidance and best practices for implementing the FACE Reference Architecture. It features implementation scenarios and example implementations of FACE Interfaces and UoCs. It is written to support the FACE Technical Standard.

2 Integrator Roles

The process of integrating a UoC into a FACE Computing Environment involves several Stakeholders, each of which has a specific role:

- Program Manager (PM)
- Systems Engineer
- Software Engineer
- Test Engineer

Figure 1 shows the high-level UoC integration process with Stakeholder roles shown as green boxes, activities shown as blue ovals, and information flowing (primarily the UoC itself) as blue arrows. Specific roles are expanded upon within the subsequent sections.

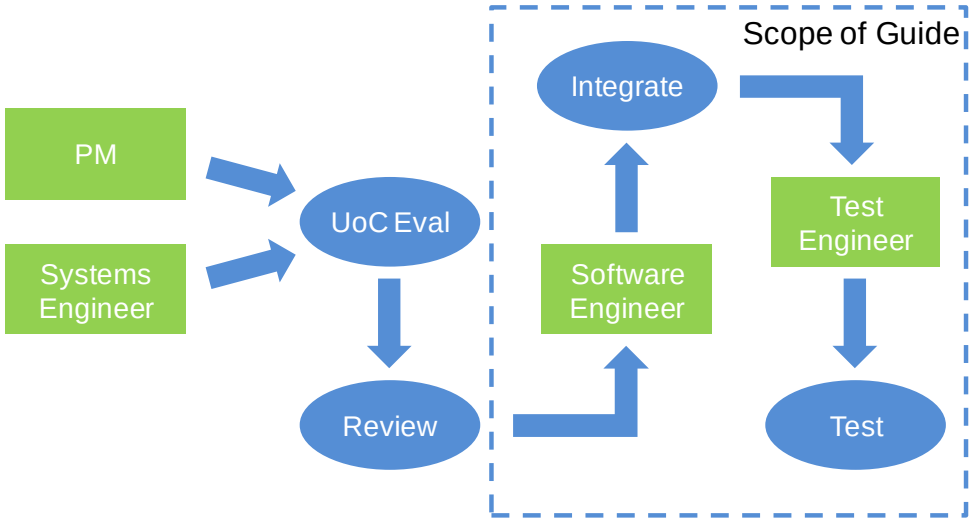


Figure 1: Workflow of FACE Integration

2.1 Program Manager (PM)

The PM is the ultimate decision-maker as to whether a UoC is to be considered for integration. The PM is generally not involved with the integration process as it pertains to the FACE Computing Environment. However, they are responsible for working with the appointed Systems Engineer to review the contract or FACE Registry entry regarding the UoC being considered for adoption onto an aviation platform. In addition, the PM appoints and/or assists the Systems Engineer with review of the UoC requirements documentation. See Section 3.1 for further information on requirements review.

2.2 Systems Engineer

The Systems Engineer involved with the integration of the FACE UoC will assist the PM, or applicable decision-making authority. The Systems Engineer will assist the PM with requirements review, as well as perform system review to determine whether adoption of the UoC meets the platform needs. The Systems Engineer will also verify that the platform provides the appropriate dependencies needed for UoC execution. For further information on systems and requirements review, see Section 3.1.

2.3 Software Engineer

The role of physically integrating the FACE UoC into the FACE Computing Environment will be performed by a Software Engineer. Depending on the titles used by the specific program or project office, the title of Software Engineer may be referred to by a similar name. For the purposes of this document and document scope, the definition used for this title is described as a person who applies the software engineering, a branch of computer science, to the design, development, maintenance, integration, testing, and evaluation of computer software.

The Software Engineer is responsible for transferring the object code to the computing platform that hosts the FACE Computing Environment. The Software Engineer writes or supplies the integration code that transitions the UoC between states and provides dependencies for execution. Upon completion, the Software Engineer compiles the object (that interfaces with the UoC) and links with the UoC and appropriate libraries that provide FACE Interfaces.

Note: The Systems (or System) Integrator, or person performing the integration of the UoC, is likely to take the role of the Software Engineer described in this section.

2.4 Test Engineer

The role of a Test Engineer comes into play after the physical integration of a FACE UoC. The Test Engineer is responsible for creating test cases based on the requirements specification and the acceptance criteria that was produced by the PM and the Systems Engineer. The test cases that are written can be satisfied using the provided unit tests, integration cases, and validation tests. Upon completion, the Test Engineer can confirm that the FACE UoC is working properly, integrated correctly, and the documentation is accurate.

3 FACE Integration

The FACE integration process details the end-to-end process of evaluating a potential UoC through its placement into a target FACE Computing Environment and commencement of the testing process. While this chapter does not address every single use case, it does provide guidance for every step of the FACE integration process and integration considerations for every UoC type based on its FACE Segment.

3.1 Systems and UoC Requirements Review

Requisition of a FACE UoC will generally stem from a system requirement that references the FACE Technical Standard for the software solution. The requirements review process for a FACE UoC should be a detailed effort to examine whether the UoC's functional requirements satisfy the system-level requirement. In other words, the desire is to determine whether the UoC performs the functions that the system requires to accomplish a task.

While individual project and product offices will have their own methods for reviewing products against system requirements, this section outlines the considerations that must be taken into account with regard to FACE UoCs being considered for integration into a FACE Computing Environment. Before proceeding with documentation review, it should be noted that certain FACE UoCs, particularly those in the Portable Components Segment (PCS) and Platform-Specific Services Segment (PSSS), require the presence of a FACE Computing Environment to operate. Because of this, a PM must be aware of their current capabilities with respect to supporting integration of FACE UoCs when considering a UoC for adoption.

3.1.1 Documentation

It is important to review available documentation for the UoC in order to ascertain whether integration is possible into a given FACE Computing Environment. Documentation for FACE UoCs may include things such as publicly released information from the Software Supplier, conformance artifacts, design documents, etc. It is important that the Systems Engineer reviewing the UoC for potential integration works with the Software Supplier Point of Contact (POC) in order to acquire the available documentation to determine whether not only the UoC provides the capability needed, but that the target FACE Computing Environment can facilitate integration (see Section 3.1.2).

3.1.1.1 FACE Registry

For FACE conformant UoCs, the FACE Registry is a good source of descriptive information regarding the UoC's dependencies and target environment. The FACE Registry does not house the actual software; however, there are fields available for Software Suppliers to put information about their product so that potential consumers can gain a general picture of what may be expected without contacting the supplied POC. It should be noted, however, that many of these fields are optional and are at the discretion of the Software Supplier whether or not to populate them. There is also no independent review of the descriptions provided by the Software Supplier against the UoC's actual conformance. For example, the UoC description could be an accurate

description based on requirements that are part of the FACE Technical Standard. The UoC description could also be mainly marketing information that includes capabilities that are part of the FACE Technical Standard as well as other capabilities that are outside of the FACE Technical Standard for which no conformance claims can be made.

3.1.2 UoC Dependencies

The foundational prerequisite for a UoC integration onto a destination platform is the FACE Computing Environment. The FACE Technical Standard defines a FACE Computing Environment as being composed of the software infrastructure (Transport Services, Operating System, and I/O Services Segments), and the PSSS UoCs required by the FACE UoCs.

For all potential UoC integrations, it is important to evaluate the UoC based on interface dependencies. All UoCs require the Operating System Segment (OSS). However, not all OSS implementations provide support for the spectrum of OS Profiles defined by the FACE Technical Standard. It is therefore pertinent that the Profile(s) provided by the target OSS be considered as to whether the UoC under consideration aligns with the environment.

Table 1 details the specific interface dependencies associated with UoCs of a particular segment. This does not include common services or device services particular to the PSSS.

Table 1: UoC Segment Interface Dependencies

UoC Segment	Interface Dependency
Portable Components Segment (PCS)	TypedTS (Transport Services Segment)
	Base (Transport Services Segment)
	Component State Persistence (Transport Services Segment)
	Configuration Services (Operating System Segment)
	Graphics Services
	Health Monitoring and Fault Management (Operating System Segment)
	Life Cycle Management
	Component Frameworks
	Programming Language Run-Time
Transport Services Segment (TSS)	Configuration Services (Operating System Segment)
	Component State Persistence (Transport Services Segment)
	Type Abstraction (Transport Services Segment)
	Transport Protocol Module (Transport Services Segment)
	Device Drivers (TPM only)

UoC Segment	Interface Dependency
	Health Monitoring and Fault Management (Operating System Segment)
	Life Cycle Management
	Component Frameworks
	Programming Language Run-Time
Platform-Specific Services Segment (PSSS)	TypedTS (Transport Services Segment)
	Base (Transport Services Segment)
	Component State Persistence (Transport Services Segment)
	Configuration Services (Operating System Segment)
	Graphics Services
	Health Monitoring and Fault Management (Operating System Segment)
	Life Cycle Management
	Component Frameworks
	Programming Language Run-Time
	I/O Service (I/O Services Segment)
I/O Services Segment (IOSS)	Component State Persistence (Transport Services Segment)
	Configuration Services (Operating System Segment)
	Health Monitoring and Fault Management (Operating System Segment)
	Life Cycle Management
	Component Frameworks
	Programming Language Run-Time
	Device Drivers

Interface dependencies are a vital part of the UoC considerations when reviewing a potential UoC for adoption. However, there may also be application-specific dependencies. A FACE UoC may require another UoC in order to operate. An example of this would be the use of a centralized service, or reliance on a message type that is fairly uncommon but is supplied by a UoC from another vendor. It is important that application-level or third-party dependencies be taken into account when reviewing a UoC for integration, as this may be a potential barrier.

Common services are another type of UoC dependency that involve services that reside in the Platform-Specific Common Services (PSCS) Sub-Segment. Access to these services is accomplished via the TSS. Therefore, a UoC that uses a common service will require a TypedTS interface, in accordance with Table 1, in order to achieve integration. An example of a common service is the Centralized Configuration, which is detailed in Section 3.2.1.1.2.

A PCS or PSSS UoC will have additional dependencies that are defined in the USM. For more information on these specific areas, refer to Section 3.2.2.1.1 for PCS UoCs, and Section 3.2.3.1.1 for PSSS UoCs. PSSS UoCs will also have dependencies on specific I/O Services, which are explained in Section 3.2.3.2.

3.1.3 Performance Considerations

Performance aspects are typically beyond the scope of the FACE Technical Standard as it is an implementation consideration. However, there are a few places within the FACE Technical Standard that reference it as an attribute. Component Frameworks (FACE Technical Standard, Edition 3.0, §2.6) and TSS UoCs are UoC types that are usually expected to have performance considerations. For performance references in the TSS, see the FACE Technical Standard, Edition 3.0, §3.8.1.1.

The Security OS Profile is provided by the FACE Technical Standard as a means of accommodating the more common attributes that can be expected with respect to performance, as it is the most restrictive with regard to POSIX™ and ARINC 653 function calls. However, the integrator may have to take performance into account when integrating a FACE UoC, regardless of the OS Profile to which it is aligned.

If there are memory or throughput requirements with respect to software applications on a target platform, then it is the responsibility of the person reviewing the UoC to consult the UoC documentation in order to resolve these dependencies. Specific information related to performance may exist in the UoC Software User's Manual.

It is important to note that many aspects of the FACE Technical Standard are optional for a specific implementation. This should be taken into account by individuals who review FACE UoCs for adoption. The addition of optional services within UoCs may impact performance, although that may not always be the case.

3.1.4 Toolchain Considerations

Certain considerations for the target platform that must be taken into account with respect to integration of a FACE UoC. Specific examples include cases where a C/C++ UoC is delivered as a static library, or when a Java® UoC is distributed as a .jar file, which calls native shared libraries. In either of these cases, the libraries themselves have to be compatible with the platform used. A platform is usually defined by the following characteristics:

- Operating System (e.g., Linux®, Embedded OS)
- Processor (e.g., x86_64, ARM, PPC)
- Compiler (e.g., GCC, LLVM)
- C/C++ library (e.g., glibc)

  — For Ada, appropriate run-time libraries and environment

— For Java UoCs, the specific version of the Java Development Kit (JDK) used to build the .jar files must be present

In general, binaries are not compatible between platforms and can only be used on the platform for which they have been built. Prior to integrating a binary UoC, it must be verified that the binary will work on the target platform. Therefore, when considering a UoC for integration, the target platform toolchains must be reviewed. If the target platform cannot satisfy the tool requirements for the UoC, then integration may not be successful.

The supported toolchains for compilation and integration are usually dependent on the selected OSS UoC and processor architecture. UoCs that are provided in source form should be both toolchain and target processor architecture-agnostic. UoCs that are provided in object form may require recompilation prior to delivery using the toolchain and target processor architecture associated with the targeted integration environment.

3.1.5 OSS Considerations

This section describes considerations and characteristics about the targeted integration platform's OSS component that should be reviewed before integration of other UoCs can occur. Considerations and characteristics discussed in this section are based on integration of a FACE conformant OSS component as the basis of the targeted integration platform.

The OSS may be a collection of integrated UoCs. Conformance of the UoCs may be individual or for the collection. The Software Engineers responsible for integrating the OSS capabilities may provide some capabilities such as configuration services. OSS characteristics can vary between OSS conformances. Key characteristics that may impact integration of other UoCs include:

- FACE Technical Standard Edition
- Support for target Profile(s)
- Support for targeted Profile options (e.g., multi-core, multi-process, frameworks, networking, configuration services)
- Support for targeted run-time language/version directly or available by integrating OSS UoC sub-components
- For UoCs with processor family dependencies, support for targeted processor family
- Possible support for targeted integration platform (e.g., board support package)

3.2 Integration

Integration of a FACE UoC can vary depending on the FACE Segment in which the UoC resides. This section discusses the common aspects that apply to most FACE UoCs, as well as the segment-specific considerations that an integrator should take into account with respect to UoCs that reside within the particular FACE Segment.

3.2.1 General Information

This section highlights areas of integration concern that are considered universal to all FACE UoCs that reside in the PCS, TSS, PSSS, or IOSS. OSS UoCs, with the exception of a

Configuration Services UoC, are considered outside the realm of this section as these aspects of FACE integration do not apply to them.

3.2.1.1 Configuration Information

Configuration Information is an important aspect of managing a UoC. This subsection provides additional guidance for the integrator, depending on the method configuration used by the UoC.

3.2.1.1.1 Configuration Services API

The Configuration Services API is available as a means of retrieving configuration information. If the Configuration Services interface is used, then the integrator should consult the usage information artifacts for the UoC to determine if the file needs to be generated by the integrator, or whether it is provided. In the event that a file is not provided, or the file provided is not compatible with the Configuration Services UoC contained within the FACE Computing Environment, then the integrator must generate the appropriate file type containing the configuration information specified in the software artifacts.

Once the file has been populated, or imported into the Configuration Services UoC, then the integrator must provide the Configuration Services concrete instance for the UoC during its startup. This must be accomplished by instantiating the Injectable Interface, provided by the UoC, for the Configuration Services Interface. Information on how to instantiate the Injectable Interface can be retrieved from either the Software User's Manual, or the Conformance Test Suite (CTS) Factory Functions, if available.

3.2.1.1.2 Centralized Configuration

A Centralized Configuration may be used to retrieve configuration information using the TS Interface. If this is used, the integrator must ensure that the TypedTS interface for the TSS connection that will be used for configuration retrieval has been generated using the USM or data structures provided. The defined data structure(s) used by the UoC serve(s) as the format of the required configuration information. For detailed information on values and their possible ranges, consult the usage information and/or the Software User's Manual of the UoC artifacts.

In order to integrate the UoC's TypedTS interface with the Centralized Configuration hosted within the FACE Computing Environment, a TSS transformation may be required along with the possible addition of a new TypedTS for the outgoing connection for the Centralized Configuration UoC.

Once the TypedTS concrete instance for the Centralized Configuration connection has been generated, it is up to the integrator, or external agent, to instantiate the Injectable Interface for this interface to resolve the dependency. Information on how to instantiate the Injectable Interface can be retrieved from either the Software User's Manual, or the CTS Factory Functions, if available.

3.2.1.1.3 Local Configuration

If the UoC Software Supplier provides their own means of retrieving configuration information, it is generally expected that the file type with the information required by the UoC will be provided with the software artifacts. Consult the usage information and Software User's Manual of the software artifacts for information on integrating the configuration file type into the software directory structure.

3.2.1.2 *Dependency Injection*

A key aspect of integrating a FACE UoC into a FACE Computing Environment is resolving dependencies via the Injectable Interface. It is expected that a UoC, in its object form, has been compiled against the abstract declarations of the FACE Interfaces. The responsibility of the integrator is to provide the UoC with concrete implementations of the FACE Interfaces in order for the UoC to execute successfully. Information on what interfaces the UoC uses can be found in the UoC design artifacts.

When integrating the object file(s) into the target software environment, the integrator or external agent object(s) must link with the UoC object(s) along with the libraries or objects providing the concrete implementations of the required FACE Interfaces.

The integrator object, or external agent, must resolve the dependency by instantiating the UoC-provided Injectable Interface for each FACE Interface it must use. Information on how to instantiate the Injectable Interface can be retrieved from either the Software User's Manual, or the CTS Factory Functions, if available. The CTS Factory Functions are provided to the FACE Verification Authority as a testable artifact with the FACE CTS.

Once the Injectable Interface has been instantiated, the function(s) responsible for managing the integrated UoC must then use the *Set_Reference*() method for each instantiated Injectable Interface and pass the concrete FACE Interface instance as the reference argument.

3.2.1.3 *UoC Life-Cycle*

An important aspect of UoC integration is managing its life-cycle. This may be accomplished via the FACE Life Cycle Management (LCM) Interfaces, methods provided by the FACE Interface, or, in the case of a PCS or PSSS UoC, by using a set of functions or methods defined by the Software Supplier. The integrator should reference the Software User's Manual for more information. Figure 2 shows the steps involved in the UoC life-cycle.

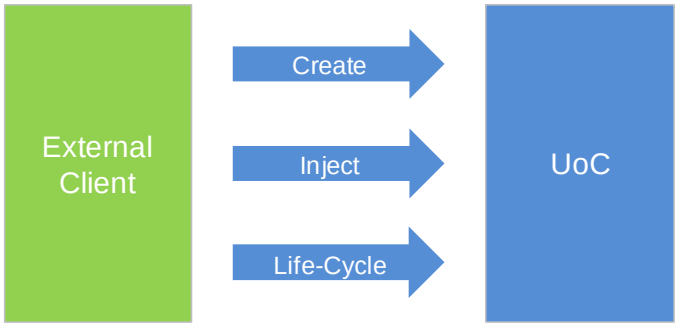


Figure 2: UoC Life-Cycle Phases

3.2.1.3.1 LCM Interfaces

The FACE Technical Standard provides a set of optional interfaces that are intended to standardize how a UoC is managed by an external source. These interfaces are a part of the FACE LCM APIs. In addition to providing a method of UoC state transition, the interfaces provide a standardized means of managing the life-cycle of other FACE UoCs that it may use.

For in-depth examples of UoCs using and providing LCM Services that are beyond the scope of this document, consult the Reference Implementation Guide for FACE Technical Standard, Edition 3.0., Volume 2, §9.

Providing LCM Interfaces

FACE LCM interfaces may be provided by the UoC for initialization, configuration, or state transition. If they are provided, then the integrator must instantiate instances of the provided interfaces in order to call the methods provided by each API.

For information on instantiating provided FACE LCM interfaces, consult the UoC Software User's Manual on how to accomplish this in the source code that will manage the UoC. If available to the integrator, the CTS Factory Functions that are provided for conformance testing can be used as a reference for retrieving instances of the concrete interfaces.

In an additional step, the LCM Stateful interface will require the integrator, or external agent, to transition the UoC to different states using data types that are modeled as a part of the UoC's USM. The integrator will need to retrieve this information from the USM, as well as consult the usage information and Software User's Manual provided by the Software Supplier.

Using LCM Interfaces

If a UoC needs to transition other UoCs using the LCM interfaces, the integrator will be required to provide the UoC with concrete instances of the required interfaces. The integrator should consult the Software User's Manual and UoC design artifact to ascertain which LCM interfaces are required by the UoC.

The integrator is required to instantiate the Injectable Interfaces for the LCM interfaces in order to call the associated *Set_Reference*() method to provide the concrete implementation. The Software User's Manual should contain information on how to instantiate the provided Injectable Interfaces. If available, the CTS Factory Functions that are provided for conformance testing can be used as a reference for retrieving instances of the concrete Injectable Interfaces.

3.2.1.4 Health Monitoring and Fault Management (HMFM)

See Section 3.2.6.2 for details on the UoC's usage of the HMFM Interface.

3.2.2 PCS UoC

UoCs that will reside in the PCS are limited to the interfaces provided by the FACE Technical Standard. These interfaces include:

- TSS:
  - — Base
  - — Transport Services (TS) Interfaces
    - TypedTS
    - TypedTS Extended
  - — Component State Persistence
- LCM Services:
  - — Initializable
  - — Configurable

— Connectable

— Stateful

- Configuration Services
- OSS Interface

Of the above listed interfaces, the only APIs that are reasonably expected to be used by every PCS UoC are the TS Interface of the TSS, and the OSS Interface for the specific OS Profile to which the UoC aligns. Therefore, it is considered a requirement that a TSS and OSS (one that provides the UoC-required APIs for the appropriate OS Profile) be present for the PCS integration effort. All other interfaces that a PCS UoC may use are optional, yet must be offered by the destination FACE Computing Environment in order for successful integration to occur.

3.2.2.1 TSS

For FACE UoCs that require the use of the interfaces provided by the TSS, the integrator should consult the UoC design software artifacts to determine which specific interfaces are needed.

3.2.2.1.1 TS Interface

The TS Interface is a requirement for all FACE UoCs that transmit data across the TSS. The TS Interface consists of the following APIs:

- TypedTS
- TypedTS Extended

The TypedTS interface is the API that the UoC will utilize to send and receive messages across the TSS. Each TypedTS declaration and implementation will be unique to the USM and defined data type that will be transmitted. It is highly likely that the concrete implementation for the TypedTS interfaces used by the FACE UoC may be generated using tools to integrate with the existing TS, or TS-TA (Transport Services-Type Abstraction) UoC in the FACE Computing Environment. The integrator should consult the Software User's Manual for the TSS UoC that provides the TS Interface for information on generation of new TypedTS interfaces.

Once the TypedTS has been generated or exists for integration with the FACE UoC and TS UoC, the connection information must then be configured. Consult Section 3.2.1.1 for further information on determining the configuration needs of a FACE UoC.

In addition to the TS Interface, the TSS Base Interface provides initialization and connection management for the FACE UoC. It is expected that the FACE UoC being integrated will provide an Injectable Interface for the dependency resolution of the TSS Base. Consult Section 3.2.1.2 for further information.

For the FACE UoC being integrated, the Connection name element will be present in the corresponding USM which defines the UoP Model for the UoC. This same Connection name must be used as the Connection name for the TSS connection that will be configured by the TSS UoC. It is a requirement for all PCS UoCs, per the FACE Technical Standard.

The connection information for a UoC will be located in the USM for the UoC being integrated. The example below shows a portable component UoP Model defined in a USM. The "connection" element contains information relating to a TSS connection. In the example, the connection is a Single Instance Message Connection that is Non-Blocking. It has a "name" value

of ATC_FROM_AIRCFG, which is the name that is used as the Connection name used by the TSS UoC to create, manage, and destroy the unique connection.

```
<element xmi:type="uop:PortableComponent"
xmi:id="EAID_1E50677C_F340_4d24_8EBF_123047097756" name="ATC_Manager"
description="ATC Manager PCS UoP" transportAPILanguage="CPP"
faceProfile="SafetyBase">
    <thread xmi:type="uop:Thread"
xmi:id="EAID_A8AD7164_864A_457a_AF42_86F13D049058"/>
    <memoryRequirements xmi:type="uop:RAMMemoryRequirements"
xmi:id="EAID_A0DB50DA_362D_4293_8D16_399F531FD7F1"/>
    <connection xmi:type="uop:SingleInstanceMessageConnection"
xmi:id="EAID_6F306931_6576_4ee0_B9FB_A2AEA77B1593"
name="ATC_FROM_AIRCFG" description="TSS Inbound message to ATC_Manager
from Airconfig." period="1.0" synchronizationStyle="NonBlocking"
messageType="EAID_DAD2716C_E707_4f64_934D_CC3416292C15"/>
</element>
```

The connection information from the USM must be added to the configuration information used by the TSS in order to create this new connection. The individual connection details may require formatting if the file structure used by the TSS UoC differs.

3.2.2.1.2 Component State Persistence

The Component State Persistence (CSP) Interface is an optional API provided by the TSS. It allows a UoC to store and retrieve internal state information or private data without defining the data according to the FACE Data Architecture. If a FACE UoC requires the CSP Interface, then the integrator must provide the UoC with a concrete instance of a CSP UoC that is contained within the target FACE Computing Environment. See Section 3.2.1.2 for details on providing the UoC with a concrete instance of a FACE Interface.

3.2.3 PSSS UoC

The PSSS provides UoCs that create an infrastructure unique to the platform. This includes common services, graphics, as well as components that provide device data to UoCs located in the PCS.

The interfaces that may be used by a PSSS UoC include all of those defined in Section 3.2.2 for PCS UoCs, as well as the various I/O Services interfaces provided by the IOSS. It should be noted, however, that there may be cases where a PSSS may not need either a TSS or an IOSS to serve its function. Therefore, the only set of interfaces that are considered a constant requirement for a PSSS UoC are those provided by the OSS.

3.2.3.1 Transport Services

For FACE UoCs that require the use of the interfaces provided by the TSS, the integrator should consult the UoC design software artifacts to determine which specific interfaces the FACE UoC needs to use.

3.2.3.1.1 TS Interface

The TS Interface is a requirement for all FACE UoCs that transmit data across the TSS. The TS Interface consists of the following APIs:

- TypedTS

- TypedTS Extended

The TypedTS interface is the API that the UoC will utilize to send and receive messages across the TSS. Each TypedTS declaration and implementation will be unique to the USM and defined data type that will be transmitted. It is highly likely that the concrete implementation for the TypedTS interfaces used by the FACE UoC may be generated using tools to integrate with the existing TS, or TS-TA UoC in the FACE Computing Environment. The integrator should consult the Software User's Manual for the TSS UoC that provides the TS Interface for information on generation of new TypedTS interfaces.

Once the TypedTS has been generated, or exists for integration with the FACE UoC and TS UoC, the connection information must then be configured. Consult Section 3.2.1.1 for further information on determining the configuration needs of a FACE UoC.

In addition to the TS Interface, the TSS Base Interface provides initialization and connection management for the FACE UoC. It is expected that the FACE UoC being integrated will provide an Injectable Interface for the dependency resolution of the TSS Base. Consult Section 3.2.1.2 for further information.

For the FACE UoC being integrated, the Connection name element will be present in the corresponding USM which defines the UoP Model for the UoC. This same Connection name must be used as the Connection name for the TSS connection that will be configured by the TSS UoC. It is a requirement for all PCS UoCs, per the FACE Technical Standard.

The connection information for a UoC will be located in the USM for the UoC being integrated. The example below shows a portable component UoP Model defined in a USM. The "connection" element contains information relating to a TSS connection. In the example, the connection is a Single Instance Message Connection that is Non-Blocking. It has a "name" value of ATC_FROM_AIRCFG, which is the name that is used as the Connection name used by the TSS UoC to create, manage, and destroy the unique connection.

```
<element xmi:type="uop:PortableComponent"
xmi:id="EAID_1E50677C_F340_4d24_8EBF_123047097756" name="ATC_Manager"
description="ATC Manager PCS UoP" transportAPILanguage="CPP"
faceProfile="SafetyBase">
    <thread xmi:type="uop:Thread"
xmi:id="EAID_A8AD7164_864A_457a_AF42_86F13D049058"/>
    <memoryRequirements xmi:type="uop:RAMMemoryRequirements"
xmi:id="EAID_A0DB50DA_362D_4293_8D16_399F531FD7F1"/>
    <connection xmi:type="uop:SingleInstanceMessageConnection"
xmi:id="EAID_6F306931_6576_4ee0_B9FB_A2AEA77B1593"
name="ATC_FROM_AIRCFG" description="TSS Inbound message to ATC_Manager
from Airconfig." period="1.0" synchronizationStyle="NonBlocking"
messageType="EAID_DAD2716C_E707_4f64_934D_CC3416292C15"/>
</element>
```

The connection information from the USM must be added to the configuration information used by the TSS in order to account for this new connection. The individual connection details will be subject to the format required by the TSS UoC present in the FACE Computing Environment. The integrator must add these details associated with the connection for the integrated FACE UoC to create its needed connections.

3.2.3.1.2 Component State Persistence

The CSP Interface is an optional API provided by the TSS. It allows a UoC to store and retrieve internal state information or private data without defining the data according to the FACE Data Architecture. If a FACE UoC requires the CSP Interface, then the integrator must provide the UoC with a concrete instance of a CSP UoC that is contained within the target FACE Computing Environment. See Section 3.2.1.2 for details.

3.2.3.2 IOSS

A PSSS UoC is expected to interface with devices using the IOS Interface, in most cases. The IOSS provides multiple bus-specific I/O Services that can implement the IOS Interface. The PSSS UoC will have been compiled to the abstract version of a specific I/O Service. It should be noted that a PSSS may use more than one type of I/O Service. The integrator must resolve the dependency, or dependencies, for the PSSS UoC by providing concrete implementations of the I/O Service(s) using the Injectable Interface, as described in Section 3.2.1.2. It is also the responsibility of the integrator to ensure that the target FACE Computing Environment provides the appropriate I/O Service(s) required by the PSSS UoC. For information on what I/O Service(s) the PSSS UoC uses, the integrator should consult the design documents provided with the UoC.

Configuration of the I/O Service is performed by the IOSS UoC. However, the PSSS UoC must provide the Connection name to the IOSS UoC when opening connections using the IOS Interface. In some cases, the PSSS UoC may also provide I/O connection parameters to the IOSS using the IOS Interface. The Software User's Manual for the PSSS UoC should contain any and all configuration information the UoC needs to provide when using the IOS Interface. The integrator must ensure that the configuration information for specific I/O connections matches the configuration information for the IOSS UoC. For additional information on IOSS configuration, see Section 3.2.5.3. Figure 3 displays the relationship a PSSS UoC has to an I/O Service provided by an IOSS UoC. The configuration source for each UoC contains a Connection name, while the configuration source for the I/O Service contains the associated connection details associated with the Connection name.

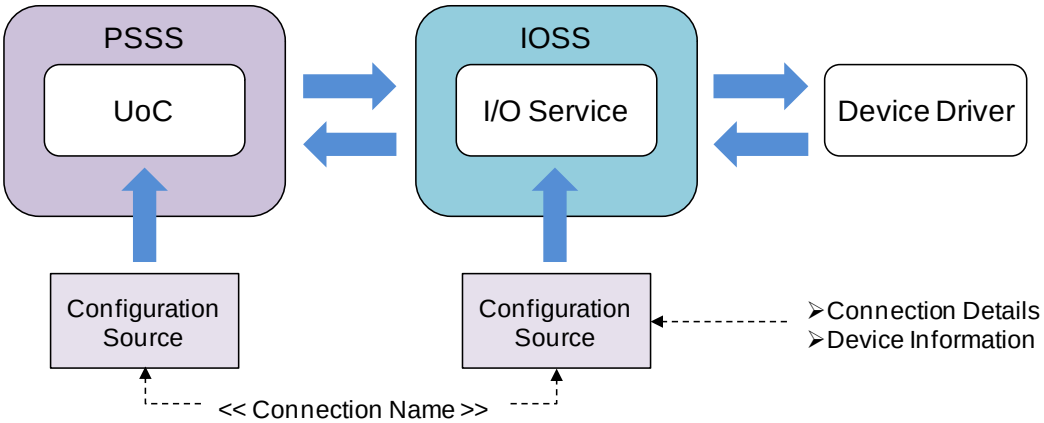


Figure 3: PSSS UoC and I/O Service Relationship

Device drivers used by I/O Services for specific I/O connections are normally outside the realm of the PSSS scope, as it is usually decided by the integrator based on unique platform or system needs. However, in the event that a PSSS UoC relies on a specific device driver for its input and/or output data, it is the responsibility of the integrator to ensure that the device driver and its

corresponding I/O Service are present within the target FACE Computing Environment. In some cases, a PSSS UoC will be accompanied by a specific I/O Service implementation and device driver when the UoC has certain device dependencies. It is the responsibility of the integrator to ensure these dependencies are indeed resolved prior to integration in order for the PSSS UoC to behave as expected. Any and all device dependencies should be reviewed during the systems and UoC requirements review period, as described in Section 3.1.

3.2.4 TSS UoC

Integrating TSS UoCs provides movement of data within a FACE Computing Environment. When it comes to integrating or replacing a TSS with a set of UoCs, four factors must be considered:

- Configuration
- Dependency injection
- TypedTS generation
- Dependencies on other TSS UoC types

For information on configuration of TSS UoCs, see Section 3.2.1.1. For dependency injection, PCS and PSSS UoCs that use interfaces provided by a TSS UoC must be provided with concrete instances. Integrators should reference the UoC Software User's Manual for information on creating instances of the services provided by the TSS UoC.

As it pertains to TypedTS generation, each TSS must provide, at a minimum, the TS Capability, Configuration Capability, and the Distribution Capability. The TS Capability must provide the TS Interface, which provides the TypedTS interface(s). The TypedTS interface(s) is specific to data types defined as templates in a FACE USM. When integrating a TSS UoC that provides the TS Interface, the TypedTS layer must be generated either using a generator tool provided with the UoC, or by the integrator. This may include generating the abstract TypedTS using the FACE CTS and USM for the PSSS/PCS UoC, followed by hand-coding the appropriate calls in the concrete definitions that should be defined in the Software User's Manual for the TSS UoC. It is unlikely that a hand-coding method of generating the TypedTS will occur. However, it is a possible scenario.

The Configuration Capability and Distribution Capability are expected to be satisfied by the same UoC, as they go hand-in-hand. The Distribution Capability is expected to provide the Base Interface for the creation and management of distribution connections, while the Configuration Capability provides the means for doing so, given specific connection information.

3.2.4.1 Types of TSS UoC

FACE Technical Standard, Edition 3.0 defines six potential UoCs that may exist in a TSS. These UoCs are identified in Table 7 of the FACE Technical Standard. The table lists each UoC type, along with the required capabilities that each must provide, as well as optional capabilities that may be implemented at the Software Supplier's discretion.

3.2.4.1.1 TS UoC

A TS UoC is a UoC that must, at a minimum, provide the TS Capability, Configuration Capability, and Distribution Capability. The TS Capability provides the TS Interface, while the Distribution Capability provides the Base Interface.

The TS UoC will likely provide a method of generating the TypedTS layer using either a supplied USM, or a data type declaration. It is advised that the individuals reviewing the candidate TS UoC consider this situation during the review of software artifacts for adoption consideration. For information on how to properly generate and populate the TypedTS layer for the TS UoC, consult the Software User's Manual.

If the candidate TS UoC is a replacement for another TS UoC or a TS-TA Adapter UoC, the following steps are required:

- All integrator objects that provide TSS UoC dependencies to PCS/PSSS UoCs will have to be modified to account for new TSS Base and TypedTS objects

  Note: This will involve a recompilation of integrator object code, as well as re-linking with UoCs and libraries.

- All UoCs that use a TS library will have to re-link with the new TS library that features the new TS UoC

- Configuration files may require modification due to differences in format between the outgoing and incoming TS UoC

  Note: The Configuration of a TSS UoC will only be done using either local configuration or via the Configuration Services API. It is unlikely for a TSS UoC to leverage a Centralized Configuration. For information on configuration of a TSS UoC, see Section 3.2.1.1.

A TS UoC is allowed to use either a CSP and/or a Transport Protocol Module (TPM) as part of its implementation. If a TPM is utilized, it is likely that the TPM UoC will accompany the TS UoC, as it is an implementation-specific option that, typically, will be designed for a specific TPM. For any allowed interfaces that are used by the TS UoC, the integrator will have to resolve dependency injection for each concrete instance of the service used by the UoC. This may be accomplished in the objects that provide dependency resolution for PCS/PSSS UoCs, since each instance of a TS UoC will need to have instances of used interfaces provided to it prior to it being used. For more information about dependency injection, see Section 3.2.1.2.

3.2.4.1.2 Type Abstraction UoC

A Type Abstraction UoC provides transport capabilities without relying on the data types expressed in a USM. Since a PSSS UoC or PCS UoC can only communicate with the TSS through the Type-Specific Typed Interface, a Type Abstraction UoC will require a TS-TA Adapter to provide a TS Capability defined in the FACE Technical Standard, Edition 3.0, §3.7.3.

Integrators should consult the Software User's Manual regarding the accompanying Type Abstraction UoC for guidance on configuration information, which can also be found in Section 3.2.1.1.

Note: The configuration of a TSS UoC will only be done using local configuration or via the Configuration Services API. It is unlikely for a TSS UoC to leverage a Centralized Configuration.

3.2.4.1.3 TS-TA Adapter UoC

The TS-TA Adapter UoC is one of two UoC options that a TSS implementation can use to provide the TS Capability defined in the FACE Technical Standard. The difference between the two is that the TS-TA Adapter, specifically, is the one used if a Type Abstraction UoC is used by

the TypedTS layer. Therefore, for purposes of TSS implementations, the TS Capability is considered a "this or that, but not both" choice of implementation between the TS UoC and TS-TA Adapter UoC.

It is possible that more than one TSS implementation may exist in a FACE Computing Environment. As an example, there may be both a TS UoC and a TS-TA Adapter within the same FACE Computing Environment, each providing a different means of accomplishing the Distribution Capability, either inherent in the TS UoC or encapsulated within the Type Abstraction. Therefore, each implementation of Base and TypedTS is specific to each UoC's implementation and should not be considered interchangeable.

The TS-TA Adapter UoC will likely provide a method of generating the TypedTS layer using either a supplied USM, or a data type declaration. It is advised that the individuals reviewing the candidate TS-TA Adapter UoC consider this situation during the review of software artifacts for adoption consideration. For information on how to generate and populate the TypedTS layer for the TS-TA Adapter UoC, consult the Software User's Manual.

If the TS-TA Adapter UoC being integrated is a replacement for either another TS-TA Adapter UoC or a TS UoC, the following steps will have to be accomplished in order to achieve successful integration:

- All integrator objects that provide TSS UoC dependencies to PCS/PSSS UoCs will have to be modified to account for new TSS Base and TypedTS objects

  Note: This will involve a recompilation of integrator object code, as well as re-linking with UoCs and libraries.

- All UoCs that use a TS library will have to re-link with the new TS library that features the new TS-TA Adapter UoC

- Configuration files may require modification due to differences in format between the outgoing and incoming TS-TA Adapter UoC

  Note: The configuration of a TSS UoC will only be done using either local configuration or via the Configuration Services API. It is unlikely for a TSS UoC to leverage a Centralized Configuration because it must create a TSS connection, which therefore must be modeled in a USM. For information on configuration of a TSS UoC, see Section 3.2.1.1.

3.2.4.1.4 Transport Protocol Module UoC

The TPM UoC is a capability that provides a TSS Intra-Segment Interface for the purpose of moving data between different TS domains. It is meant to promote interoperability between systems while also providing a level of abstraction for the data being exchanged.

The TPM Interface is used by either a TS UoC or Type Abstraction UoC to provide the capability to transmit data to another TSS residing outside of the present FACE Computing Environment. Even though portability is a common feature among FACE UoCs, a TPM UoC is not portable to TS or Type Abstraction UoCs that are not designed to use a TPM Interface. Considering this, it is expected that a TPM will be provided along with a compatible TS or Type Abstraction UoC. Integration of a TPM UoC alone is an unlikely use case unless a TPM Capability already exists in the FACE Computing Environment. Therefore, it is assumed that a TPM implementation will be tied to a TSS UoC that uses it as a means of transmitting messages.

For transmission of messages using the TPM Capability, the TPM must rely on a method of serialization and deserialization for each message. In order for the TPM to remain agnostic of message details, it must be supplied with appropriate serialization and deserialization methods that will call specific marshalling methods based on the native types contained within the message. The TPM may provide this means as part of the UoC capabilities, or it will rely on methods provided to it by the Serialization interface. The integrator is expected to consult the Software User's Manual to ascertain the method of serialization dependency used by the TPM UoC.

3.2.4.1.5 Serialization Interface

The Serialization interface is an optional interface used by a TPM, which must be supplied by the integrator in cases where the TPM UoC does not provide a means of generating serialization methods based on a USM-defined message type. The interface supplies the TPM UoC with a *Get_Serialization*() method to be used for retrieving unique instances of the Message_Serialization interface. Each instance of the Message_Serialization interface is referenced by a Globally Unique Identifier (GUID), and provides the *Serialize*() and *Deserialize*() methods that are particular to a specific message structure. The Message_Serialization interface uses a passed instance of the Primitive_Marshalling interface, provided by the TPM, in order to have a means of marshalling and unmarshalling a message structure based on the Interactive Data Language (IDL) types of each data element.

It is expected that the integrator will supply the concrete implementations of the Serialization and Message_Serialization interfaces. A capability may be provided by the TPM UoC to generate these dependencies. However, if it is not, the dependencies must be hand-generated. Regardless of the origin, the integrator must resolve the Serialization dependency for the TPM using the Injectable Interface provided by the TPM. See Section 3.2.1.2 for information on dependency injection.

In Figure 4, a use case for the relationship a TPM has with the external client and Serialization interface is depicted. The dotted line is meant to represent the notion that the Serialization and Message_Serialization interfaces are expected to reside outside of the TPM UoC boundary.

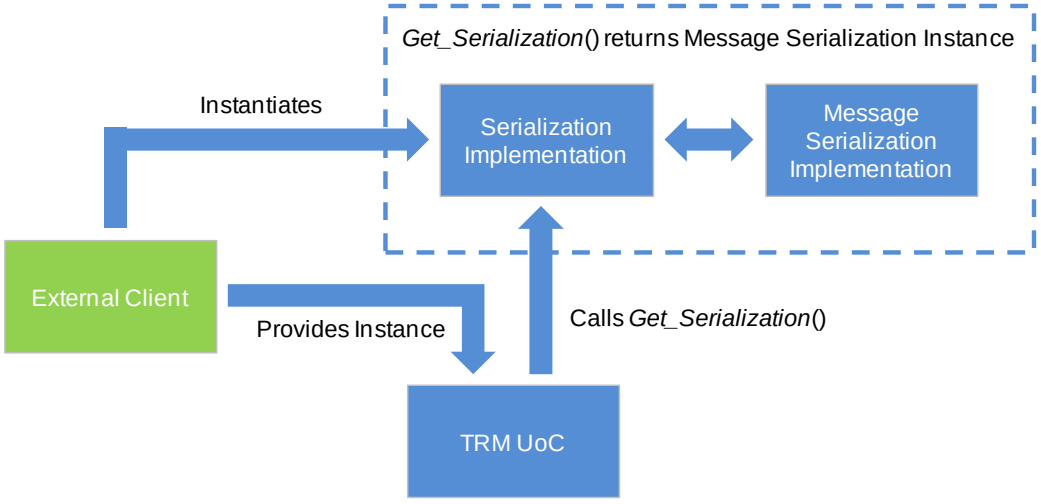


Figure 4: TPM UoC and Serialization Interface Use Case

3.2.4.1.6 Component State Persistence UoC

The CSP UoC is considered a standalone UoC in the TSS. This means that it does not use any other interface defined as part of the TSS. With this in mind, there are no specific considerations for integration of a CSP other than configuration and dependency injection for allowed FACE APIs, per Table 8 of the FACE Technical Standard.

For configuration of the CSP, see Section 3.2.1.1. Information on dependency injection can be found in Section 3.2.1.2.

3.2.5 IOSS UoC

The IOSS provides a means of abstracting communication between PSSS UoCs and the device drivers that interface with external hardware. UoCs in the IOSS provide I/O "Services" which handle communication based on specific protocols or buses. PSSS UoCs achieve communication with these I/O Services using a standardized interface which is specific to an I/O bus architecture.

3.2.5.1 I/O Services

An IOSS UoC will provide at least one I/O Service in its implementation. Table 6 of the FACE Technical Standard defines multiple I/O Service types, each with its own tailored implementation of the IOS Interface.

The common functions of the IOS Interface are detailed in Appendix C of the FACE Technical Standard. Each of the bus-specific declarations are defined in Appendix C as well.

The documentation accompanying an IOSS UoC should properly identify each of the I/O Services provided by the UoC, as well as their variants. It should be noted that variants can exist for different I/O Services, depending on the different devices with which it supports communication. It is the responsibility of the integrator to ensure that the I/O Services provided by the UoC meet the needs of the PSSS UoCs in the hosted FACE Computing Environment and will satisfy future integration needs. The IOS Interfaces are not meant to be interchangeable and each bus-specific interface used by a PSSS must have this dependency satisfied by an I/O Service provided by an IOSS UoC at run-time.

IOSS UoCs commonly contain device driver calls that are not provided by the OSS Interface. However, the remainder of the UoC method/function calls are still subject to the allowed calls for a specific OS Profile. The integrator should ensure that the OSS in the target FACE Computing Environment provides the OS Profile required by the UoC prior to integration.

3.2.5.2 Device Drivers

I/O Services are typically dependent on specific device drivers for communication with external hardware. These device drivers are not always native to OS platforms and may require further steps on behalf of the integrator. This may include one of the following scenarios:

- Device drivers or bus architectures provided by the external hardware or hardware vendor
- Device drivers provided along with the IOSS UoC
- Third-party device drivers that have to be requisitioned separately

It is the responsibility of the integrator to ensure that the device driver access needed by the IOSS UoC is satisfied given one of the means highlighted in the scenarios above. A proper version of the device driver is also important. Specific version dependencies should be detailed in the UoC documentation. It should be noted that hardware and device needs may be the reason IOSS UoCs are requisitioned for use in the hosted FACE Computing Environment. These dependencies should be identified as early as possible during the requirements review and system design processes.

Device drivers that are not native to OS environments must be carefully selected to ensure that system requirements are met and that safety certifications are not impacted.

3.2.5.3 IOSS Configuration

The IOSS UoC establishes unique I/O connections for communication between a PSSS UoC and instances of an I/O Service. These connections must be configured using a configuration source. The IOSS UoC Software User's Manual should contain information for how the UoC is configured and how an integrator needs to set up the configuration source (e.g., XML file, ini file, etc.).

It is expected that each PSSS requiring an I/O connection will read in a Connection name for use when it makes the call to *Open_Connection*() using the IOS Interface. This Connection name must be referenced by the IOSS UoC using its configuration source to establish a connection via configured connection parameters. The result is an assigned handle returned to the PSSS for use in subsequent IOS Interface method calls in order to reference the specific I/O connection. The integrator must ensure that the Connection name retrieved by the PSSS UoC corresponds to a Connection name in the IOSS UoC configuration source with associated connection parameters in accordance with the IOSS UoC Software User's Manual.

3.2.5.4 I/O Adapters

There may be situations with certain PSSS components where a specific I/O Service is provided as part of either the IOSS or the provided device driver. This phenomenon is seen in cases where the hardware source for a PSSS input via the IOS Interface varies across platforms. These are sometimes known as I/O Adapters, as they are outside the scope of existing IOSS UoCs and are considered integrator "pieces". When this occurs, the integrator must ensure that the present I/O Adapter is compiled with the appropriate external client. For further information on the external client, see Section 3.2.6.

3.2.6 External Client Considerations

Integration of FACE UoCs requires a method of instantiation, dependency resolution, and transition between states. This must be provided by the integrator, based on the design of the target system, and is sometimes referred to as the "external client" in the FACE community. The external client should provide an entry point to one or more UoCs.

Information on how an integrator should construct and tailor the external client for a specific UoC should be found in the UoC Software User's Manual. This will include things such as interface dependencies (see Section 3.2.1.2) and configuration information (see Section 3.2.1.1). It should also contain information about how to instantiate and transition the UoC (see Section 3.2.1.3).

An example external client written in C++ is provided with the integration example in Appendix A.

3.2.6.1 *Interface Initialization and Expected State*

Many of the FACE Interfaces provide initialization and configuration entry points as part of the API. However, the FACE Technical Standard does not place requirements on whether the UoC itself or the external client makes these various API calls. Therefore, it can be assumed that some UoCs will handle initialization and configuration of the FACE APIs they use, whereas others will not.

It is the responsibility of the integrator to consult the UoC Software User's Manual, as well as other available documents, to verify the state of the FACE Interfaces that the candidate UoC uses. When the FACE UoC performs its own initialization and configuration of FACE APIs, the integrator only needs to provide a usable instance of the concrete interface. Otherwise, the integrator must add logic to the external client to perform the necessary actions to transition the concrete interface to the state that the integrated FACE UoC expects, given its own internal state.

3.2.6.2 *Health Monitoring and Fault Management (HMFM)*

If a FACE UoC uses the HMFM capability provided by the OSS, it is important for the integrator to consult the UoC design documents to ascertain whether the UoC uses the ARINC 653 health monitoring API, or the HMFM Interface defined by the FACE Technical Standard. The FACE Technical Standard does not explicitly state that the UoC has to access the health monitoring functions directly. There are use cases stating that the external client is expected to initialize and prepare the health monitor independent of the UoC life-cycle. The integrator should consult the UoC Software User's Manual to ascertain which use case for the health monitor applies.

The fault handler is the piece of the HMFM service that deals with the various faults raised by the UoC. There may be cases where the UoC provides its own fault handler, but it is considered a best practice that it does not.

The integrator should provide the fault handler so that system requirements and restrictions are met. It is encouraged that the presence of a fault handler be established during the requirements review process in Section 3.1. This will avoid instances where the fault handler provided by the UoC does not match the intent of the fault handlers provided by the hosted OSS. For a UoC that transitions the HMFM capability during its own life-cycle processes, the integrator must ascertain, from the UoC documentation, the point at which the external client must assign the fault handler.

3.2.6.3 *Linking with UoC*

Depending on the type of UoC, the object code must either be linked with an external client application (for PCS and PSSS) or be linked with existing UoCs within a FACE Computing Environment (TSS, IOSS, and Configuration Services). For PCS and PSSS UoCs, the integrator is expected to compile and link the external client using the same (compiler and toolset) settings. The integrator should consult the Software User's Manual for details on compiler and linker configuration. For TSS, IOSS, and Configuration Services UoCs, the external client for the UoCs that will use the new capabilities must link to the new library.

It is important for the compiler and linker dependencies to be established during the requirements review process, described in Section 3.1, in order for successful integration with existing UoCs and services. Failure to account for differing compiler and linker settings may result in the inability to integrate the UoC.

3.3 Test

Testing of a UoC will be performed between the Software Supplier and the integrator. Testing a UoC will be mission-specific. The integrator will be required to evaluate the end system use case and identify the amount of testing needed to demonstrate successful integration. Depending on assurance level, there may be a need to consult with the supplier for testing input. However, that may not be true for all levels of assurance, particularly lower levels. The integrator is expected to evaluate the UoC from many aspects as to the level of testing needed.

It is the expectation that the UoC functions according to the design and should pose little to no barriers to integration as far as performance. The integrator should consult software artifacts, to include the Software User's Manual, for information on the construction of integration tests.

4 FACE Integration Model

The FACE Integration Model is a descriptive tool to aid integration activities and provide a mechanism to describe the TSS integration details between two or more UoPs. Although it is a component of the FACE Data Model Language, it is considered an optional component and is not part of the FACE Conformance process. The focus of the Integration Model is on documenting UoP (or UoC) data exchange details for use by an integrator.

The Integration Model addresses integration by providing elements for capturing data exchange information embodied in an instance of a FACE TSS. An Integration Model relies on UoP Model details defined through the data modeling process for expressing interconnectivity. An Integration Model provides UoP connection details including routing, transformation, and transport details to be used during an integration activity and provides a container external to all UoPs concerned.

The value of the Integration Model is that it provides an explicit integration representation of FACE conformant components with other FACE conformant components. The Integration Model can be used to provide transparency into the information exchange. This transparency permits the integrator to create integration test and configuration artifacts, TSS type-specific code for information exchange, and data types used in an integration context.

The IntegrationModel element can be constructed from the following primary model elements:

- IntegrationContext: a description of all TSNodeConnections and TransportNodes that define a transport

  Multiple IntegrationContexts can be used to organize related transport elements (e.g., to contain connections and nodes related to distributing aircraft state data).

  — TSNodeConnection: connects source and destination transport nodes forming one piece of the flow of data

  — TransportNode: a node that performs some processing/transformation; the following are types of TransportNodes:

    - ViewTransporter – a node indicating the movement of data using a TransportChannel
    - ViewAggregation – a node intended to provide a mapping/transformation of multiple input views to a single output (e.g., merging of data from multiple messages to a single one)
    - ViewTransformation – a node intended to transform the contents of one view into that required by another; the kinds of transformation range from adjusting units or frames of measures to producing a subset of the input data
    - ViewFilter – a node intended to pass data through from source to destination unchanged if the contents of the view meet an integrator specified criteria

        - ViewSource – a node which can be used to produce a view without input from a UoC; the planned use of this kind of node is to provide static data to a UoP to satisfy its input requirements in a defined way (i.e., connect a UoP input to something rather than leaving it disconnected)
        - ViewSink – a node which can be used to receive and discard data from a UoP; it is intended to allow integrators to connect UoP outputs to a valid destination rather than leaving the output disconnected

- TransportChannel: identifies where the integrator intends to supply a means of transport

  Because of the near limitless number of possible transports (e.g., UDP, ARINC 653, DDS®, shared memory, etc.), the Integration Model does not provide a means of capturing all of the parameters for each possible transport. Instead, the integration provides the TransportChannel element as a way for integrators to connect their external data to the model. This might happen through use of the channel name, or the GUID assigned to the channel node.

- UoPInstance: instance of a FACE UoP

  Integrators should consider a UoP in a USM as a declaration of a kind of UoP, while the UoPInstance in the Integration Model represents an instance of that kind of UoP. In a complex system, integrators may have to accommodate multiple instances of the same UoP (e.g., redundant copies of device managers, separate copies for pilot and copilot, etc.).

More extensive details of the Integration Model can be found in the FACE Technical Standard, Edition 3.0, §J.2. Diagrams for the FACE Metamodel "face.integration" "Package" and the FACE Metamodel "face.integration" "Package:Transport" are collected and each component is discussed in detail.

A Integration Example

A.1 BALSA ADSB-Out

The example used in this document utilizes the ADSB-Out PSSS UoC provided by the Basic Avionics Lightweight Source Archetype (BALSA) architecture. The artifacts that will be leveraged will be the Software Requirements Specification (SRS), the Software Design Description (SDD), the Software Architecture Document (SAD), and the Software User's Manual.

The destination environment for the notional integration effort will be the FACE Computing Environment provided by BALSA on a CentOS 7 guest operating system. The only OSS features that will be provided by the guest OS will be the POSIX APIs.

A.1.1 Review of UoC Documentation

The first step in the review process is to review documentation available with the UoC in order to ascertain whether integration into the target environment is feasible. In this example, the system requirement leading this integration effort is for the capability to send ADSB callsign and position messages to an IOSS I/O Service within a FACE Computing Environment.

The BALSA ADSB-Out is not listed in the FACE Registry; however, for example purposes, the appropriate documents from the UoC software artifacts are available to perform an effective review.

System Review

The primary need for this example is that an ADSB-Out capability is needed within a FACE Computing Environment for a notional platform. In order to determine whether this PSSS UoC can satisfy the needs of the target environment, a review of the SRS and SDD is required.

For purposes of the example review, the following notional system requirement will be used:

- The FACE Computing Platform shall supply a method of transmitting ADSB position and callsign messages to a radio device

It is assumed that the need to satisfy device driver access follows another integration path. This example covers the need to process position and callsign data and encode corresponding messages for transmission.

In order to do an adequate system review, the SRS document provided by the ADSB-Out UoC will be reviewed first. Within the document, there are two types of requirements defined: interface and gunctional. The primary goal is to ascertain whether the functional requirements of the UoC can satisfy the system requirement that defines the behavior.

The following cited requirements are of interest in reviewing the ADSB-Out:

- The BALSA ADSB-Out Component shall receive a message from the TypedTS interface containing an ICAO ID number
- The BALSA ADSB-Out Component shall receive a message from the TypedTS interface containing aircraft position information
- The BALSA ADSB-Out Component shall transmit an ADS-B callsign message using the I/O Services interface
- The BALSA ADSB-Out Component shall transmit an ADS-B position message using the I/O Services interface
- The BALSA ADSB-Out Component shall encode a callsign message according to the specifications in TSO-C195b
- The BALSA ADSB-Out Component shall encode a position message according to the specifications in TSO-C195b

The UoC in question appears to satisfy the system requirement in that it transmits callsign and position messages encoded to the ADSB format. However, it is to be expected that the TSS message the UoC requires will not match the format of present messages that carry this information in the destination FACE Computing Environment. This will not be an automatic disqualification for consideration as a common capability in TSS implementations is message transformation. However, if there is a data type that cannot be satisfied by a current message, then integration may not be possible.

The BALSA TSS does not perform data transformations, so the expected message by the ADSB-Out must match a currently transmitted message. The ADSB-Out SDD contains an appendix that defines the modeled message, and its semantics, which are leveraged from the USM. From this, the data structure, primitive types, and realizations from the logical and conceptual levels can be viewed. It should be noted that this information may not exist in this state in actual FACE integration cases.

From the information in the ADSB-Out SDD, it is determined that the message requirements of the ADSB-Out can be supplied by a present message transmitted by the ATC Manager, another BALSA UoC.

UoC Dependencies

Within the SRS document provided, there are four sections that are of benefit to the integrator: System Interfaces; Communication Interfaces; Constraints, Assumptions, and Dependencies; and the Specific Requirements sections.

Beginning with the System Interfaces section, the content within the SRS document states the following:

- FACE OSS Safety Base Profile API (see the FACE Technical Standard, Edition 3.0, §3.6.1, Requirement 1)

While the target OS for the integration does not provide all of the aspects of the FACE OSS Safety Base Profile, requirement SRS_003 in the Interface Requirements section does provide clarification as to specific UoC needs. It reads as follows:

- SRS_003: The BALSA ADSB-Out Component shall use the FACE Safety Base OS Profile subset of the POSIX API for any and all function calls to the OSS Interface

Because the POSIX APIs are required for this UoC to function, and the target environment provides those, the review process can continue.

The Communication Interfaces section identifies two interfaces that are required for UoC function:

- FACE TSS (see the FACE Technical Standard, Edition 3.0, §3.6.1, Requirement)
- FACE IOSS (see the FACE Technical Standard, Edition 3.0, §3.6.1, Requirement 1)

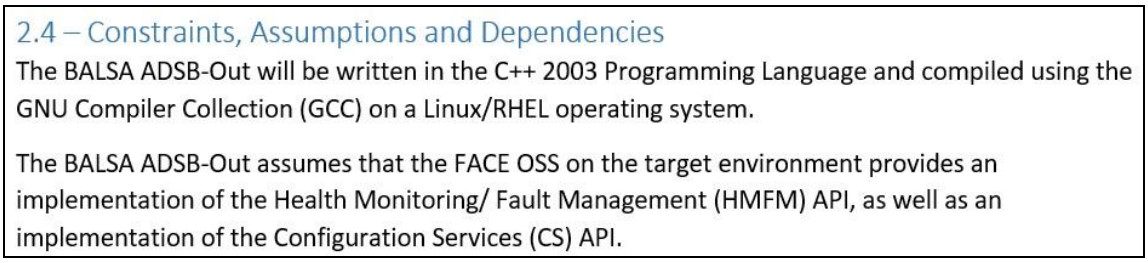
Since the FACE Computing Environment provided by BALSA contains an implementation of the FACE TSS, as well as a method of generating the TS Interface, it is assumed that this dependency is met. However, specific I/O Services are not referenced with regard to the IOSS. Because the BALSA IOSS UoC only provides the Generic I/O Service, the Interface Requirements section must be referenced for more information.

The Interface Requirements section identifies the following requirements:

- The BALSA ADSB-Out Component shall use the Generic I/O Service interface of the IOSS to open an I/O Service connection
- The BALSA ADSB-Out Component shall use the Generic I/O Service interface of the IOSS for transmitting ADSB-encoded messages

Since there are no further requirements that reference the IOSS usage, then it is known, at this point, that the IOSS dependency can be satisfied by the target environment.

Moving on to the Constraints, Assumptions, and Dependencies section of the SRS document, Figure 5 provides the contents as it reads in the document.


2.4 – Constraints, Assumptions and Dependencies

The BALSA ADSB-Out will be written in the C++ 2003 Programming Language and compiled using the GNU Compiler Collection (GCC) on a Linux/RHEL operating system.

The BALSA ADSB-Out assumes that the FACE OSS on the target environment provides an implementation of the Health Monitoring/ Fault Management (HMFM) API, as well as an implementation of the Configuration Services (CS) API.

Figure 5: ADSB-Out Dependencies

The target operating system features the GCC complier, therefore that dependency can be resolved. In addition, the FACE Computing Environment provided by BALSA contains an implementation of the HMFM interface, as well Configuration Services. Therefore, it can be ascertained that the UoC dependencies for the ADSB-Out can be resolved by the target FACE Computing Environment that the platform provides.

Performance Considerations

There are no performance considerations for the BALSA ADSB-Out, therefore the example experiment will move on to the next step in the integration process.

Tool Considerations

There are no tool considerations for the BALSA ADSB-Out, therefore the example experiment will move on to the next step in the integration process.

A.1.2 Integration

Configuration

The first task in the integration process is configuration of the ADSB-Out UoC. The Software User's Manual, or in this case the ADSB-Out User's Manual, is used to ascertain how configuration is to occur.

The ADSB-Out User's Manual contains a section titled FACE Configuration Services. Within this section are two subsections on Configuration Information and Configuration File Example. These sections detail three distinct values for string identifiers that must be retrieved by the ADSB-Out using Configuration Services. The first value is the path to which the integrator wants print statements written. The other two are for the TSS connection and IOSS connection. The TSS connection name is leveraged from the ADSB-Out USM, which is found in the UoP Model details, highlighted in Figure 6.

```
  <um xmi:type="uop:UoPModel" xmi:id="EAID_1D5F24AC_4BFC_445e_8955_DC7391C7B708" name="ADSB_UoP">
    <element xmi:type="uop:PlatformSpecificComponent" xmi:id="EAID_6830F338_2337_4dc0_81E7_A72BF648116F"
name="ADSB_Out" description="ADSB Out UoP." faceProfile="SafetyBase">
      <thread xmi:type="uop:Thread" xmi:id="EAID_F49B8F8A_3B0D_4e6e_A5E5_8D342466D5F4"/>
      <memoryRequirements xmi:type="uop:RAMMemoryRequirements"
xmi:id="EAID_21EE2385_1AE1_499d_B783_07D1EAC0032A"/>
      <connection xmi:type="uop:SingleInstanceMessageConnection"
xmi:id="EAID_8B86ED37_F2D8_4a1d_B472_878E3B4FB701" name="ADSB_FROM_ATC" description="TSS to ADS-B from ATC
Manager." period="1.0" synchronizationStyle="NonBlocking"
messageType="EAID_FA9C377F_FBE0_4e27_9FED_7FD5FA0B64B8"/>
      <lcmPort xmi:type="uop:LifeCycleManagementPort" xmi:id="EAID_EDCBADB5_CC11_4fbf_8266_E9A377E7EDFD"
messageExchangeType="OutboundMessage" lcmMessageType="EAID_AE65BAA8_B4FA_4168_BEDC_9CC712BAED14"/>
      <lcmPort xmi:type="uop:LifeCycleManagementPort" xmi:id="EAID_B6DA48FF_1531_4baf_AC83_D003D78A48E9"
lcmMessageType="EAID_CC8FD0D1_F1ED_4eb7_92D7_836E906637C8"/>
    </element>
  </um>
```

Figure 6: ADSB-Out UoP Model from USM

From the details of Figure 6, we can determine that the name of the TSS connection that needs to be added to the BALSA TSS configuration file is 'ADSB_FROM_ATC'. Therefore, the following details are added to the ini file used by the BALSA TSS, based on the integrator's decision to use the User Datagram Protocol (UDP) Multiplex capability provided by the TSS:

```
[ADSB_FROM_ATC]
TRANSPORT = FACE_Multiplex
PROTOCOL = FACE_Multiplex
CONNECTION_DIRECTION = DESTINATION
```

Since the BALSA Configuration Services has the capability to read ini files, the integrator uses the example provided in the ADSB-Out User's Manual to construct the configuration file for integration. The example configuration file is shown in Figure 7.

```
# ADSB Out Component
#
# This Component can be configured for which PSS connection to use,
# and for which IOS connection to use.
#
[ADSB_Out]
TSS_Connection = ADSB_FROM_ATC
IOS_Connection = ADSB_Out_IOS

# General UoC parameters
[UoC]
Output = /dev/stderr
```

Figure 7: ADSB-Out Configuration File Example

UoC Lifecycle

The ADSB-Out provides the LCM interfaces to manage the UoC's lifecycle. This is described in the ADSB-Out User's Manual. The interfaces provided by the ADSB-Out are the Initializable, Configurable, and Stateful LCM interfaces. The User's Manual provides examples of method calls on an instance of the ADSB-Out, as well as the values used by the ADSB-Out Stateful interface.

The ADSB-Out User's Guide also features an appendix that contains an example External Client, complete with method calls to the LCM interfaces. This helps the integrator with how to populate the transition calls in their respective version of the External Client. See the example below from the appendix of the ADSB-Out User's Manual:

```
BALSA::ADSB_Out            ADSB_out;

FACE::STRING_TYPE ADSB_Out_Config_name("ADSB_Out_Config");

FACE::RETURN_CODE_TYPE::Value    status;

FACE::Configuration::HANDLE_TYPE handle;

FACE::DM::BALSA::ComponentStateRequested Component_State_value;

// Initialize Component
ADSB_out.Initialize(status);

// Configure Connections
ADSB_out.Configure(ADSB_Out_Config_name, status);

// Run Component
Component_State_value.componentStateRequested =
FACE::DM::BALSA::ComponentRequestedOperationalState::ACTIVE;

ADSB_out.Request_State_Transition(
  Component_State_value, status
);
```

Dependency Injection

The ADSB-Out uses the TSS, IOSS, and Configuration Services interfaces for operation. Therefore, it requires the following interfaces, which are confirmed in the ADSB-Out User's Manual:

- TSS TypedTS Injectable
- TSS Base Injectable
- Configuration Services Injectable
- IOSS Generic I/O Service Injectable

The ADSB-Out User's Manual indicates that the ADSB-Out uses multiple inheritance in its C++ class structure to provide the Injectable Interfaces using a single instance of the ADSB-Out. The ADSB-Out User's Manual has example *Set_Reference*() method calls for each of the Injectable Interfaces provided by the ADSB-Out. This gives the integrator knowledge on how to create the external client to resolve the UoC's dependencies.

External Client

The ADSB-Out User's Manual provides the integrator with example function calls regarding instantiating an ADSB-Out instance, as well calling the provided LCM interfaces. The next step in the integration example is to create the external client in order to use the ADSB-Out.

The appendix in the ADSB-Out User's Manual satisfies the integrator's requirements for creating an external client using the FACE Computing Environment provided by BALSA. An example external client may not be supplied for all cases in the ADSB-Out User's Manual, nor would it always apply to the destination FACE Computing Environment. However, for this example, since the UoC being integrated is a BALSA UoC, the example provided in the User's Manual will suffice for the exercise.

The external client leveraged from the ADSB-Out User's Manual is as follows:

```
#include <ADSB_Out/ADSB_Out.hpp>

int main() {

// Declare ADSB-Out instance
BALSA::ADSB_Out            ADSB_out;

FACE::STRING_TYPE ADSB_Out_Config_name("ADSB_Out_Config");
FACE::STRING_TYPE Helpers_Config_name("Helpers");
FACE::STRING_TYPE TSS_Config_name("TSS");
FACE::STRING_TYPE ADSB_Out_IOS_name("ADSB_Out_IOS");

FACE::Configuration                    *ADSB_Out_Config;
FACE::Configuration                    *Helpers_Config;
FACE::TSS::Base                        *ADSB_Out_TSS;
FACE::TSS::BALSA::ATC_Data::TypedTS    *ADSB_Out_from_ATC_TypedTS;
FACE::IOSS::Generic::IO_Service        *ADSB_Out_IOS_Out_ptr;

ADSB_Out_Config          = new FIA::Configuration;
Helpers_Config           = new FIA::Configuration;
ADSB_Out_TSS             = new FIA::TSS_Base;
```

```
ADSB_Out_from_ATC_TypedTS = new BALSA::ATC_Data::TypedTS;
ADSB_Out_IOS_Out_ptr      = new FIA::IOSS::Generic::FIA_IO_Service;

FACE::RETURN_CODE_TYPE::Value    status;

bool                isOK;
FACE::GUID_TYPE     id;

FACE::Configuration::HANDLE_TYPE handle;

FACE::DM::BALSA::ComponentStateRequested Component_State_value;

  // Set Configuration for Component
  ADSB_out.Set_Reference(
    ADSB_Out_Config_name,
    *ADSB_Out_Config,
    id,
    status
  );

  // Set TSS instance for Component
  ADSB_out.Set_Reference(
    TSS_Config_name,
    *ADSB_Out_TSS,
    id,
    status
  );

  // Set TypedTS instances for Component
  ADSB_out.Set_Reference(
    TSS_Config_name,
    *ADSB_Out_from_ATC_TypedTS,
    id,
    status
  );

  // Set IOSS instance for Component
  ADSB_out.Set_Reference(
    ADSB_Out_IOS_name,
    *ADSB_Out_IOS_Out_ptr,
    id,
    status
  );

// Initialize and Open Configuration for Helpers
Helpers_Config->Initialize(Helpers_Config_name, status);

Helpers_Config->Open(Helpers_Config_name, handle, status);

// Initilize Helpers
isOK = FIA::Helpers::Initialize(handle, Helpers_Config);

// Initialize Component
ADSB_out.Initialize(status);

// Configure Connections
ADSB_out.Configure(ADSB_Out_Config_name, status);
```

```
// Run Component
Component_State_value.componentStateRequested =
FACE::DM::BALSA::ComponentRequestedOperationalState::ACTIVE;

ADSB_out.Request_State_Transition(
    Component_State_value, status
);
return 0;

}
```

Index