

The Open Group Guide

**Reference Implementation Guide for
FACE™ Technical Standard, Edition 3.0, Volume 2:
Computing Environment**



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

Vice Admiral (ret.) Arthur K. Cebrowski

THE[®] *Open* GROUP

Distribution Statement A – Approved for Public Release
Distribution Unlimited – (Control Number PR20190650)

Copyright © 2020, The Open Group LLC 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.

In the event of any discrepancy between text in this document and the official FACE Technical Standard, the FACE Technical Standard remains the authoritative version for certification, testing by examination, and other purposes. The official FACE Documents and Tools can be obtained at www.opengroup.org/face.

The Open Group Guide

Reference Implementation Guide for FACE™ Technical Standard, Edition 3.0, Volume 2: Computing Environment

ISBN: 1-947754-57-7

Document Number: G209 (Volume 2)

Published by The Open Group, May 2020.

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

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

or by electronic mail to:

ogface-admin@opengroup.org

Contents

| | | |
|--------|---|-----|
| 1 | Introduction..... | 1 |
| 1.1 | Background..... | 1 |
| 1.2 | FACE Approach | 1 |
| 1.3 | Scope of this Document..... | 2 |
| 2 | Operating System Segment | 3 |
| 2.1 | Introduction..... | 3 |
| 2.1.1 | Summary of Changes from FACE Technical Standard, Edition 2.1 | 3 |
| 2.1.2 | How to Read this Section | 3 |
| 2.2 | Operating System Concepts in a FACE Reference Architecture..... | 3 |
| 2.2.1 | Partitioning | 3 |
| 2.2.2 | Multicore | 9 |
| 2.2.3 | POSIX Multiprocess..... | 11 |
| 2.2.4 | Multi-Threading | 12 |
| 2.2.5 | APIs and Profiles..... | 20 |
| 2.2.6 | OSS Configuration | 20 |
| 2.2.7 | Shared Memory | 21 |
| 2.2.8 | Inter-Partition Communications | 23 |
| 2.2.9 | Intra-Partition Communications | 25 |
| 2.2.10 | Local Memory Allocation/Deallocation..... | 28 |
| 2.2.11 | Operating System Input/Output (I/O) Support | 30 |
| 2.2.12 | File System..... | 31 |
| 2.2.13 | Networking | 32 |
| 2.2.14 | Time Management..... | 33 |
| 2.2.15 | Health Monitoring/Fault Management | 34 |
| 2.3 | FACE POSIX Guidance | 35 |
| 2.3.1 | POSIX API Subsetting Golden Rules | 35 |
| 2.3.2 | The POSIX Standard and Undefined Behavior..... | 38 |
| 2.3.3 | POSIX Header Files Per FACE Profile..... | 42 |
| 2.3.4 | Allowed POSIX Constants per FACE Profile..... | 45 |
| 2.3.5 | Allowed POSIX Data Types and Structures per FACE Profile | 128 |
| 2.3.6 | Advice on POSIX API Usage | 152 |
| 2.4 | FACE ARINC 653 Guidance | 166 |
| 2.5 | FACE OSS and SCA AEP Conformance Guidance..... | 166 |
| 3 | Health Monitoring/Fault Management (HMF M) Guidance..... | 176 |
| 3.1 | Introduction..... | 176 |
| 3.1.1 | Summary of Changes from FACE Technical Standard, Edition 2.1 | 176 |
| 3.1.2 | How to Read this Section | 176 |
| 3.1.3 | FACE HMF M Services <i>versus</i> ARINC 653 Health Services | 176 |

| | | |
|-------|---|-----|
| 3.1.4 | HMFM Areas Addressed by the FACE Technical Standard..... | 177 |
| 3.1.5 | HMFM Areas Beyond the FACE Technical Standard | 177 |
| 3.2 | Role of Board Support Packages | 177 |
| 3.3 | HMFM Layering..... | 178 |
| 3.3.1 | Hardware-Detected Faults | 179 |
| 3.3.2 | Application-Detected Faults..... | 179 |
| 3.3.3 | Computing Platform Health Manager | 180 |
| 3.3.4 | External Systems | 180 |
| 3.4 | Operating System Configurations..... | 180 |
| 3.4.1 | ARINC 653 OS | 180 |
| 3.4.2 | Pure ARINC 653 OS | 180 |
| 3.4.3 | POSIX on ARINC 653 OS | 181 |
| 3.4.4 | POSIX OS | 181 |
| 3.4.5 | HMFM Services per OS Configuration..... | 181 |
| 3.5 | HMFM Event Flow..... | 182 |
| 3.6 | HMFM Implementation Guidance..... | 183 |
| 3.6.1 | Implementation on ARINC 653 OS | 183 |
| 3.6.2 | Implementation on ARINC 653 OS with POSIX APIs..... | 183 |
| 3.6.3 | Implementation on Pure POSIX OS..... | 183 |
| 3.7 | HMFM and Non-Volatile Storage | 185 |
| 3.8 | Fault Handler Guidance | 186 |
| 4 | Configuration Services API | 189 |
| 4.1 | Introduction..... | 189 |
| 4.1.1 | Summary of Changes from FACE Technical Standard, Edition 2.1 | 189 |
| 4.1.2 | How to Read this Section | 189 |
| 4.2 | FACE Configuration Categories..... | 189 |
| 4.2.1 | No Software Configurability Available..... | 189 |
| 4.2.2 | Legacy Local Configuration..... | 189 |
| 4.2.3 | API-Based Configuration | 190 |
| 4.2.4 | Hybrid Configuration | 192 |
| 4.3 | Recommended Configuration Services Artifacts..... | 192 |
| 4.3.1 | Component Configurability Definition | 192 |
| 4.3.2 | Component Configuration Set..... | 195 |
| 4.3.3 | Configuration Set Identifier List | 195 |
| 4.3.4 | Configuration Set Encoding Package | 198 |
| 4.4 | Configuration Services API Comparison to Operating System Services | 199 |
| 5 | Programming Language Run-Time Guidance | 201 |
| 5.1 | Introduction..... | 201 |
| 5.1.1 | Summary of Changes from FACE Technical Standard, Edition 2.1 | 201 |
| 5.1.2 | How to Read this Section | 202 |
| 5.2 | Conformance Testing Background | 202 |
| 5.3 | Use of Programming Language Run-Times | 202 |
| 5.4 | Accepted and Known Portability Issues | 203 |
| 5.4.1 | Device Dependencies | 204 |

| | | |
|--------|---|-----|
| 5.4.2 | For Conformance <i>versus</i> Production Build | 204 |
| 5.4.3 | Performance Variations..... | 204 |
| 5.4.4 | Bounded or Symmetric Multiprocessing..... | 204 |
| 5.5 | Programming Language Independent Guidance..... | 204 |
| 5.5.1 | Endianness..... | 204 |
| 5.5.2 | Word Size | 205 |
| 5.5.3 | Use of Non-Conformant Modes/Enumerations..... | 205 |
| 5.6 | C and C++ Guidance | 205 |
| 5.6.1 | Use of Conditionals | 205 |
| 5.6.2 | Inline Assembly..... | 207 |
| 5.7 | C Specific Guidance | 207 |
| 5.8 | C++ Specific Guidance..... | 207 |
| 5.8.1 | Overriding <i>new</i> and <i>delete</i> | 207 |
| 5.9 | Ada Guidance | 208 |
| 5.9.1 | Inline Assembly..... | 208 |
| 5.9.2 | Accessing POSIX Functions from Ada..... | 208 |
| 5.10 | Java Guidance | 210 |
| 5.10.1 | Java Run-Time Availability | 210 |
| 5.10.2 | Inline Assembly..... | 210 |
| 5.11 | Additional Considerations | 210 |
| 6 | I/O Services Segment..... | 211 |
| 6.1 | Scope..... | 211 |
| 6.2 | Migrating an IOSS UoC from Previous Editions | 211 |
| 6.3 | Design Goals..... | 212 |
| 6.4 | Using an I/O Service..... | 213 |
| 6.4.1 | Accepting the I/O Service Assignment | 213 |
| 6.4.2 | Configuring the I/O Service | 213 |
| 6.4.3 | Basic Read/Write Example | 215 |
| 6.4.4 | Asynchronous Read Example..... | 217 |
| 6.4.5 | Connection Parameters Example..... | 219 |
| 6.4.6 | Bus Parameters Example..... | 219 |
| 6.4.7 | Device I/O Use Case: Protocol Mediation | 222 |
| 6.5 | Providing an I/O Service..... | 223 |
| 6.5.1 | Implementing an I/O Service..... | 224 |
| 6.5.2 | Developing a New I/O Service Bus Architecture..... | 238 |
| 6.5.3 | Distributed Dispatch of I/O Services Interface Functions | 239 |
| 6.5.4 | IOSS Extending Parameters for an I/O Service | 243 |
| 6.5.5 | Realizing an I/O Service using LCM Services..... | 245 |
| 7 | Transport Services Segment..... | 248 |
| 7.1 | Purpose | 248 |
| 7.2 | Introduction..... | 248 |
| 7.2.1 | Summary of Changes from FACE Technical Standard, Edition 2.1 | 251 |
| 7.5 | Relationship to Data Architecture Artifacts..... | 268 |
| 7.5.1 | UoP Supplied Data Model..... | 268 |
| 7.5.2 | TSS Configuration Relationship to Integration Model..... | 270 |

| | | |
|--------|---|-----|
| 8 | Injectable Interface Guidance | 275 |
| 8.1 | Scope..... | 275 |
| 8.2 | Migrating from Previous Editions | 275 |
| 8.3 | Design Goals..... | 275 |
| 8.4 | Injectable Examples | 276 |
| 9 | Life Cycle Management Services Guidance | 282 |
| 9.1 | Scope..... | 282 |
| 9.2 | Migrating from Previous Editions | 282 |
| 9.3 | Design Goals..... | 282 |
| 9.4 | Examples for Using LCM Services | 284 |
| 9.4.1 | Multi-Component Startup and Shutdown..... | 285 |
| 9.4.2 | Multi-Component State Management | 288 |
| 9.5 | Examples for Providing LCM Services | 289 |
| 9.5.1 | Declarations for a Managed Component..... | 289 |
| 9.5.2 | Definitions for a Managed Component | 290 |
| 9.5.3 | Data Model for State Representation | 291 |
| 10 | Programming Language Mapping Rules Guidance | 292 |
| 10.1 | C Mapping Rules Guidance | 292 |
| 10.2 | C++ Mapping Rules Guidance | 292 |
| 10.2.1 | C++ Union Implementation..... | 292 |
| 10.3 | Ada Mapping Rules Guidance | 292 |
| 10.3.1 | IDL Module to Ada Packages | 292 |
| 10.4 | Mapping Rules Guidance..... | 295 |
| A | Acronyms | 296 |

List of Figures

| | |
|--|-----|
| Figure 1: Example Partition Schedule | 7 |
| Figure 2: Multiple Partitions Assigned Identical Partition Time Windows | 8 |
| Figure 3: Periodic Activity Using <i>sigwait()</i> Example | 17 |
| Figure 4: Periodic Activity Using <i>pthread_cond_timedwait()</i> Example | 18 |
| Figure 5: Periodic Activity Using <i>clock_nanosleep()</i> Example | 19 |
| Figure 6: POSIX Shared Memory Initialization Example..... | 23 |
| Figure 7: <i>pthread_create()</i> with NULL Attributes..... | 40 |
| Figure 8: Partial Initialization of Thread Attributes | 41 |
| Figure 9: Use <i>fcntl()</i> to Set Non-Blocking Mode | 153 |
| Figure 10: <i>select()</i> Method Interface Definition..... | 161 |
| Figure 11: HMFM Board Support Package and Real-Time Operating System Relationship | 178 |
| Figure 12: HMFM Layering | 179 |
| Figure 13: HMFM Event Flow..... | 182 |
| Figure 14: HMFM Fault Handler Pseudo-Code | 186 |
| Figure 15: Legacy Local Configuration | 190 |
| Figure 16: FACE API-Based Local Configuration | 191 |
| Figure 17: Configuration Services XML Example | 195 |
| Figure 18: CSIL Schema Pictorial Representation..... | 196 |
| Figure 19: CSIL Schema | 197 |
| Figure 20: Configuration Data Container..... | 200 |
| Figure 21: Language Run-Time Options..... | 203 |
| Figure 22: Basic Read/Write Sequence Diagram..... | 216 |
| Figure 23: Basic Asynchronous Read Sequence Diagram | 217 |
| Figure 24: Asynchronous Read with Callback Buffering | 218 |
| Figure 25: Asynchronous Read with Flag Assignment | 218 |
| Figure 26: Connection Parameters Sequence Diagram | 219 |
| Figure 27: Bus Parameters Sequence Diagram | 220 |
| Figure 28: Device Protocol Mediation Use Case Sequence Diagram | 223 |
| Figure 29: Distributed Dispatch to a FACE Non-Conformant Server | 240 |
| Figure 30: Distributed Dispatch to Remote PSSS UoC | 242 |
| Figure 31: Initializing a Managed I/O Service | 246 |
| Figure 32: Transport Services Segment Capabilities | 249 |
| Figure 33: Send Sequence Diagram | 254 |
| Figure 34: Send Using Type Abstraction Sequence Diagram..... | 255 |
| Figure 35: Receive Sequence Diagram | 257 |
| Figure 36: Request Reply Sequence Diagram..... | 258 |
| Figure 37: Callback Sequence Diagram | 260 |
| Figure 38: Initialize Sequence Diagram | 261 |
| Figure 39: Example ARINC 653 Application Initializing the TSS..... | 262 |
| Figure 40: Create Connection Sequence Diagram | 264 |
| Figure 41: Adding a TPM Sequence Diagram | 266 |
| Figure 42: Example Serialization Sequence Diagram | 266 |
| Figure 43: Send Message Async Sequence Diagram | 268 |
| Figure 44: USM Information Related to Integration Model Information..... | 269 |
| Figure 45: TSS Configuration Data Element Relationships..... | 271 |

| | |
|--|-----|
| Figure 46: Class Diagram for First Injectable Example | 276 |
| Figure 47: Sequence Diagram for First Injectable Example | 279 |
| Figure 48: Class Diagram for Second Injectable Example..... | 280 |
| Figure 49: Sequence Diagram for Second Injectable Example | 280 |
| Figure 50: Program Execution Points..... | 283 |
| Figure 51: Class Diagram for LCM Services Example of Startup and Shutdown | 285 |
| Figure 52: Sequence Diagram for Beginning Program Execution Points | 286 |
| Figure 53: Sequence Diagram for Remaining Program Execution Points | 287 |
| Figure 54: Class Diagram for LCM Services Example of State Management..... | 288 |
| Figure 55: Sequence Diagram for LCM Services Example of State Management | 289 |
| Figure 56: Sequence Diagram for Using Configuration Services in LCM Services | 291 |

List of Tables

| | |
|---|----|
| Table 1: APIs Allowing Code to be Scheduled Periodically..... | 14 |
| Table 2: Header Files Per Profile | 42 |
| Table 3: <aio.h> Constants..... | 46 |
| Table 4: <arpa/inet.h> Constants..... | 47 |
| Table 5: <complex.h> Constants..... | 47 |
| Table 6: <cpio.h> Constants..... | 47 |
| Table 7: <ctype.h> Constants..... | 48 |
| Table 8: <dlopen.h> Constants | 49 |
| Table 9: <errno.h> Constants | 49 |
| Table 10: <fcntl.h> Constants | 52 |
| Table 11: <fenv.h> Constants | 54 |
| Table 12: <float.h> Constants | 55 |
| Table 13: <fmtmsg.h> Constants | 56 |
| Table 14: <fnmatch.h> Constants | 57 |
| Table 15: <ftw.h> Constants | 58 |
| Table 16: <glob.h> Constants | 58 |
| Table 17: <inttypes.h> Constants | 59 |
| Table 18: <iso646.h> Constants | 65 |
| Table 19: <langinfo.h> Constants | 66 |
| Table 20: <limits.h> Constants | 68 |
| Table 21: <locale.h> Constants | 74 |
| Table 22: <math.h> Constants..... | 75 |
| Table 23: <ndbm.h> Constants | 77 |
| Table 24: <net/if.h> Constants | 77 |
| Table 25: <netdb.h> Constants..... | 77 |
| Table 26: <netinet/in.h> Constants | 79 |
| Table 27: <netinet/in.h> Additional Constants | 80 |
| Table 28: <netinet/tcp.h> Constants..... | 81 |
| Table 29: <nl_types.h> Constants | 81 |
| Table 30: <poll.h> Constants | 81 |
| Table 31: <pthread.h> Constants..... | 82 |
| Table 32: <regex.h> Constants..... | 84 |
| Table 33: <sched.h> Constants | 84 |
| Table 34: <search.h> Constants | 85 |
| Table 35: <semaphore.h> Constants | 85 |
| Table 36: <signal.h> Constants | 85 |
| Table 37: <spawn.h> Constants | 89 |
| Table 38: <stdarg.h> Constants..... | 89 |
| Table 39: <stdbool.h> Constants..... | 90 |
| Table 40: <stddef.h> Constants..... | 90 |
| Table 41: <stdint.h> Constants..... | 90 |
| Table 42: <stdio.h> Constants..... | 93 |
| Table 43: <stdlib.h> Constants..... | 93 |
| Table 44: <string.h> Constants | 94 |
| Table 45: <stropts.h> Constants | 94 |

| | |
|---|-----|
| Table 46: <sys/ioctl.h> Constants | 97 |
| Table 47: <sys/ipc.h> Constants | 97 |
| Table 48: <sys/mman.h> Constants | 97 |
| Table 49: <sys/msg.h> Constants..... | 98 |
| Table 50: <sys/resource.h> Constants..... | 99 |
| Table 51: <sys/select.h> Constants | 99 |
| Table 52: <sys/sem.h> Constants | 99 |
| Table 53: <sys/shm.h> Constants..... | 100 |
| Table 54: <sys/socket.h> Constants | 100 |
| Table 55: <sys/stat.h> Constants..... | 102 |
| Table 56: <sys/statvfs.h> Constants | 103 |
| Table 57: <sys/time.h> Constants | 104 |
| Table 58: <sys/wait.h> Constants | 104 |
| Table 59: <syslog.h> Constants | 105 |
| Table 60: <tar.h> Constants | 106 |
| Table 61: <termios.h> Constants..... | 107 |
| Table 62: <time.h> Constants | 111 |
| Table 63: <trace.h> Constants..... | 112 |
| Table 64: <trace.h> Constants..... | 113 |
| Table 65: <unistd.h> Constants..... | 114 |
| Table 66: <utmpx.h> Constants | 127 |
| Table 67: <wchar.h> Constants..... | 127 |
| Table 68: <wctype.h> Constants..... | 127 |
| Table 69: <wordexp.h> Constants..... | 127 |
| Table 70: <aio.h> Data Types | 128 |
| Table 71: <arpa/inet.h> Data Types | 129 |
| Table 72: <complex.h> Data Types | 129 |
| Table 73: <cctype.h> Data Types | 129 |
| Table 74: <dirent.h> Data Types..... | 129 |
| Table 75: <fcntl.h> Data Types..... | 130 |
| Table 76: <fenv.h> Data Types | 130 |
| Table 77: <ftw.h> Data Types..... | 131 |
| Table 78: <glob.h> Data Types | 131 |
| Table 79: <grp.h> Data Types | 131 |
| Table 80: <iconv.h> Data Types | 131 |
| Table 81: <inttypes.h> Data Types | 132 |
| Table 82: <langinfo.h> Data Types..... | 132 |
| Table 83: <locale.h> Data Types | 132 |
| Table 84: <math.h> Data Types | 132 |
| Table 85: <monetary.h> Data Types | 133 |
| Table 86: <monetary.h> Data Types | 133 |
| Table 87: <ndbm.h> Data Types | 133 |
| Table 88: <net/if.h> Data Types..... | 134 |
| Table 89: <netdb.h> Data Types | 134 |
| Table 90: <netinet/in.h> Data Types | 134 |
| Table 91: <nl_types.h> Data Types | 135 |
| Table 92: <poll.h> Data Types | 135 |
| Table 93: <pthread.h> Data Types | 136 |
| Table 94: <pwd.h> Data Types | 136 |
| Table 95: <regex.h> Data Types | 137 |
| Table 96: <sched.h> Data Types | 137 |

| | |
|--|-----|
| Table 97: <search.h> Data Types..... | 137 |
| Table 98: <semaphore.h> Data Types..... | 137 |
| Table 99: <setjmp.h> Data Types | 138 |
| Table 100: <signal.h> Data Types | 138 |
| Table 101: <spawn.h> Data Types | 139 |
| Table 102: <stdarg.h> Data Types | 139 |
| Table 103: <stdbool.h> Data Types | 139 |
| Table 104: <stddef.h> Data Types | 139 |
| Table 105: <stdint.h> Data Types | 140 |
| Table 106: <stdio.h> Data Types | 141 |
| Table 107: <stdlib.h> Data Types | 141 |
| Table 108: <string.h> Data Types | 141 |
| Table 109: <stropts.h> Data Types | 142 |
| Table 110: <sys/ipc.h> Data Types..... | 142 |
| Table 111: <sys/msg.h> Data Types | 143 |
| Table 112: <sys/resource.h> Data Types | 143 |
| Table 113: <sys/select.h> Data Types..... | 143 |
| Table 114: <sys/sem.h> Data Types | 144 |
| Table 115: <sys/shm.h> Data Types | 144 |
| Table 116: <sys/socket.h> Data Types..... | 144 |
| Table 117: <sys/stat.h> Data Types | 145 |
| Table 118: <sys/statvfs.h> Data Types | 145 |
| Table 119: <sys/time.h> Data Types..... | 146 |
| Table 120: <sys/times.h> Data Types | 146 |
| Table 121: <sys/types.h> Data Types | 146 |
| Table 122: <sys/uio.h> Data Types..... | 148 |
| Table 123: <sys/un.h> Data Types..... | 148 |
| Table 124: <sys/un.h> Data Types..... | 148 |
| Table 125: <sys/wait.h> Data Types | 148 |
| Table 126: <termios.h> Data Types | 149 |
| Table 127: <time.h> Data Types | 149 |
| Table 128: <trace.h> Data Types | 150 |
| Table 129: <utime.h> Data Types | 150 |
| Table 130: <utmpx.h> Data Types..... | 150 |
| Table 131: <wchar.h> Data Types | 151 |
| Table 132: <wctype.h> Data Types | 151 |
| Table 133: <wordexp.h> Data Types | 151 |
| Table 134: Recommendations for SCA Applications in a FACE OSS Security Profile..... | 167 |
| Table 135: Recommendations for SCA Applications in a FACE OSS Security Profile..... | 171 |
| Table 136: Recommendations for SCA Applications in a FACE OSS Security Profile..... | 173 |
| Table 137: HM Options per OS Configuration | 181 |
| Table 138: POSIX Signals Mapped to FACE Errors | 184 |
| Table 139: Configuration Parameter Example | 193 |
| Table 140: Configuration Container Terminology | 196 |
| Table 141: Potential Transport Technology Mapping to TSS Return Codes on Send | 255 |
| Table 142: Potential Transport Technology Mapping to TSS Return Codes on Receive | 258 |
| Table 143: Potential Transport Technology Mapping to Return Codes on Callback Handler... | 260 |
| Table 144: Potential Transport Technology Mapping to TSS Return Codes on Initialize..... | 262 |
| Table 145: Potential Transport Technology Mapping to TSS Return Codes on Create Connection | 264 |

Preface

The Open Group

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

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

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

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

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

This Document

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

Trademarks

ArchiMate®, DirecNet®, Making Standards Work®, Open O® logo, Open O and Check® Certification logo, OpenPegasus®, Platform 3.0®, The Open Group®, TOGAF®, UNIX®, UNIXWARE®, and the Open Brand X® logo are registered trademarks and Agile Architecture Framework™, Boundaryless Information Flow™, Build with Integrity Buy with Confidence™, Dependability Through Assuredness™, Digital Practitioner Body of Knowledge™, DPBoK™, EMMMTM, FACE™, the FACE™ logo, FBPTM, FHIM Profile Builder™, the FHIM logo, IT4IT™, the IT4IT™ logo, O-AAFTM, O-DEFTM, O-HERATM, O-PASTM, Open FAIR™, Open Platform 3.0™, Open Process Automation™, Open Subsurface Data Universe™, Open Trusted Technology Provider™, O-SDUTM, OSDUTM, Sensor Integration Simplified™, SOSA™, and the SOSA™ logo are trademarks of The Open Group.

CORBA®, DDS®, OMG®, UML®, and XMI® are registered trademarks and IDL™, Interface Definition Language™, and Unified Modeling Language™ are trademarks of Object Management Group, Inc. in the United States and/or other countries.

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

Linux® is a registered trademark of Linus Torvalds.

LynxOS® is a registered trademark of Lynx Software Technologies, Inc.

POSIX® is a registered trademark of the IEEE.

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

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

Additional Contributors

- Judy Cerenzia, The Open Group
- Christopher Crook, U.S. Army PEO Aviation
- Robert Jefferies, NAVAIR
- Paul Jennings, Presagis

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

Referenced Documents

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

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

- AC 20-148: Reusable Software Components, Federal Aviation Authority, 2004
- Aerospace Recommended Practice (ARP) 4754: Guidelines for Development of Civil Aircraft and Systems (Revision A), SAE International, 2010
- Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, SAE International, 1996
- ARINC Characteristic 739-1: Multi-Purpose Control and Display Unit (MCDU), June 1990
- ARINC Characteristic 739A-1: Multi-Purpose Control and Display Unit (MCDU), December 1998
- ARINC Report 661: Cockpit Display System Interfaces to User System, May 2010
- ARINC Specification 429: Mark 33 Digital Information Transfer System (DITS), May 2004
- ARINC Specification 653: Avionics Application Software Standard Interface, November 2010
- Clinger-Cohen Act (CCA), U.S. Federal Law, 1996 – formerly the Information Technology Management Reform Act of 1996 (ITMRA)
- Committee on National Security Systems (CNSS) 4009: Information Assurance (IA) Glossary, 2010
- Common Criteria (CC) for Information Technology Security Evaluation, Version 3.1, Revision 3, July 2009 (this is equivalent to ISO/IEC 15408:2009)
- Common Object Request Broker Architecture (CORBA[®]) Specification, Version 3.3, published by the Object Management Group (OMG[®]), November 2012
- Data Distribution Service for Real-time Systems Specification, Version 1.2, published by the Object Management Group (OMG[®]), July 2001
- Design Assurance Guidance for Airborne Electronic Hardware, DO-254, published by Radio Technical Commission for Aeronautics (RTCA), April 2000
- Electronic Industries Alliance (EIA) J-STD-016-1995: IEEE Standard for Information Technology – Software Lifecycle Processes – Software Development – Acquirer-Supplier Agreement (withdrawn)

- Enterprise Integration Patterns, 2012; refer to <http://www.eaipatterns.com/>
- FACE™ (Future Airborne Capability Environment): Conformance Policy, Version 2.0 (X1608), September 2016, published by The Open Group; refer to: www.opengroup.org/library/x1608
- FACE™ (Future Airborne Capability Environment) Contract Guide: Guidance in Writing Solicitations and Proposals with FACE Requirements, Version 2.0 (G18D), September 2018, published by The Open Group; refer to: www.opengroup.org/library/g18d
- FACE™ (Future Airborne Capability Environment) Shared Data Model Governance Plan (X1817), June 2018, published by The Open Group; refer to: www.opengroup.org/face
- FACE™ (Future Airborne Capability Environment) Technical Standard, Edition 3.0 (C17C), November 2017, published by The Open Group; refer to: www.opengroup.org/library/c17c
- Federal Information Security Management Act (FISMA), U.S. Federal Law, 2002
- IEEE Std 1003.1-2008: IEEE Standard for Information Technology – Portable Operating System Interface (POSIX®) – Base Specifications, Issue 7, December 1, 2008
- IETF RFC 5424: The Syslog Protocol, March 2009; refer to: <http://tools.ietf.org/html/rfc5424>
- IETF RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog, March 2009; refer to: <http://tools.ietf.org/html/rfc5425>
- ISO/IEC 12207:2008: Systems and Software Engineering – Software Lifecycle Processes; refer to: <https://www.iso.org/standard/43447.html>
- ISO/IEC 15408:2009: Information Technology – Security Techniques – Evaluation Criteria for IT Security; refer to: <https://www.iso.org/standard/50341.html>
- ISO/IEC/IEEE 24765:2017: Systems and Software Engineering Vocabulary; refer to: <https://www.iso.org/standard/71952.html>
- National Security Telecommunications and Information Systems Security Policy (NSTISSP) No. 11: National Policy Governing the Acquisition of Information Assurance (IA) and IA-Enabled Information Technology (IT) Products, January 2000 (revised June 2003) – NSTISSC is now known as the Committee on National Security Systems (CNSS)
- NSA SSE-100-1: National Security Agency Information Assurance Guidance for Systems Based on a Security Real-Time Operating System (RTOS), 2009
- OSGi Service Platform, Release 4, Version 4.3, OSGi Alliance, 2012; refer to: www.osgi.org/Release4
- Software Considerations in Airborne Systems and Equipment Certification, DO-178B, published by Radio Technical Commission for Aeronautics (RTCA) Inc., December 1992
- Software Considerations in Airborne Systems and Equipment Certification, DO-178C, published by Radio Technical Commission for Aeronautics (RTCA) Inc., January 2012
- STANAG 4586: Standard Interfaces of UA Control System (U.S.) for NATO UA Interoperability, NATO, April 2017

- U.S. Department of Defense Directive DoDI 8500.1, Information Assurance (IA), 2002
- U.S. Department of Defense Directive DoDI 8500.2: Information Assurance (IA) Implementation, 2003
- U.S. Department of Defense Military Standards: MIL-STD-498 (cancelled 1998) and MIL-STD-1553
- XML Metadata Interchange (XMI[®]) Specification, Version 2.4.2, published by the Object Management Group (OMG[®]), April 2014

1 Introduction

1.1 Background

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

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

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

1.2 FACE Approach

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

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

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

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

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

1.3 Scope of this Document

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

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

This volume has been designed to:

- Support the FACE Technical Standard, Edition 3.0
Note: In the event of conflict between the FACE Technical Standard and the FACE Reference Implementation Guide, the FACE Technical Standard takes precedence.
- Present best practices for designing and implementing the FACE Computing Environment, which consists of Units of Conformance (UoCs) in the Transport Services Segment (TSS), Input/Output Services Segment (IOSS), and Operating System Segment (OSS) of the FACE Reference Architecture
- Provide further information on FACE OSS Profiles, Programming Language Run-times, and how Health Monitoring/Fault Management (HMF) and Configuration Services may be implemented within a FACE OSS
- Provide guidance on implementing and using the Injectable Interface for FACE Interfaces
- Provide further information on the use and implementation of Life Cycle Management (LCM) Services
- Present guidance on Programming Language Mapping Rules

Reference Implementation Guide for FACE Technical Standard, Edition 3.0, Volume 1: General Guidance provides guidance and assistance for the following items:

- Designing and implementing PCS and PSSS UoCs
- Use of Graphics Services
- Safety and Security considerations in UoC designs

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

2 Operating System Segment

2.1 Introduction

This chapter contains guidance for using the FACE Operating System Segment (OSS). This contrasts with other chapters where the emphasis is on implementing the capabilities of the segment.

The OSS provides and controls access to the computing platform for the other FACE Segments. The OSS uses processor control mechanisms, such as memory management units and register access controls (e.g., user versus kernel privileges), to restrict FACE Segments to their required computing platform resources and operational capabilities. This level of control, when supported, permits various levels of independence between FACE UoCs, permitting greater portability across FACE computing environments.

2.1.1 Summary of Changes from FACE Technical Standard, Edition 2.1

The following changes to the OSS have occurred since FACE Technical Standard, Edition 2.1:

- Many requirements were reworked for improved clarity
- Added option of ARINC 653 2015 to enable multicore support
- Addition of support for IPv4 multicast
- Allow OSS UoC to directly support OpenGL
- FACE Technical Standard Appendix A improved to document communications APIs and note APIs deprecated by the POSIX® standard
- Wide character support removed

2.1.2 How to Read this Section

The information contained in this portion of the Reference Implementation Guide should be used when developing software hosted on operating systems that provide a FACE OSS. Although general guidance is included, the bulk of the material in this section highlights what is unique to the FACE specific tailoring of standards such as the POSIX standard and ARINC 653.

2.2 Operating System Concepts in a FACE Reference Architecture

2.2.1 Partitioning

2.2.1.1 Key Characteristics

The FACE Reference Architecture is intended to support multiple Units of Portability (UoPs) executing on a common computing platform. To manage multiple UoPs on a common

computing platform, the FACE Reference Architecture includes support for partitions and partitioning. Partitions are the base unit of isolation enforced by the OSS. System developers use partitions to isolate resources required to perform their intended function and isolate software components from other software components running in the same FACE Computing Environment. A UoP resides in a partition that contains, or provides access to, resources required by the UoP.

The following list provides the resources and capabilities the OSS may control within a partition:

- **Space** – the OSS controls how the physical address space is assigned among partitions
When memory partitioning is enforced, the OSS ensures partitions are assigned their configured quota of memory. With memory partitioning, one partition may not inadvertently encroach on the memory associated with another partition (e.g., Partition A is prevented from accessing or modifying the memory associated with Partition B). Two or more partitions may share memory by allocating a virtual address space range in each partition to the same physical memory. Mappings of virtual address space ranges to physical memory are created to allocate a partition's memory quota. A typical memory methodology makes use of a processor's memory management unit to create a virtual address space for each partition.
- **Access** – the OSS controls how accesses to memory and/or devices are assigned
When access controls are enforced, the OSS ensures a partition is restricted to only its authorized access controls. Typical access controls include read, write, and execute permissions. Some components may tolerate broader access controls (e.g., other components can have read-access), while other components require strict access controls (e.g., restricting read-access in order to protect the confidentiality of information). For shared memory, each partition may have unique access permissions (e.g., one with read-write, some with only read permissions, and others with no access at all) to the shared memory region.
- **Resource** – the OSS controls how operating system resources (e.g., kernel objects, open file handles, socket port handles) are allocated
When resource partitioning is enforced, the OSS ensures partitions are allocated their configured quota of resources.
- **Time** – the OSS controls how execution time on the processor is allocated
When time partitioning is enforced, the OSS ensures partitions are allocated their configured quota of time. A typical time enforcement methodology, as defined in ARINC 653, supports the definition of a module schedule of fixed duration (the major frame time) that the OSS enforces over time. The module schedule is subdivided further into fixed time windows each defined by a duration (variably sized) and an offset from the beginning of the module schedule. The association of each partition with one or more time windows provides the configured quota of time for a partition. For POSIX multiprocess allocations, see Section 2.2.3.

2.2.1.2 *Configurability*

For implementations of the FACE Computing Environment that utilize partitioning, individual partitions and resources associated with the partitions are statically defined as part of the configuration data. During initial start-up, the OSS creates the partitions as defined in the configuration data. During operation, the OSS controls when partitions are provided access to

the processor and prevents partition violations. The following characteristics need to be selected on a partition basis:

- Whether ARINC 653 or POSIX interfaces and operational environment are utilized (i.e., partitions are used to manage resources for all operational environments)
- Which FACE Profile (General Purpose, Safety Extended, Safety Base, or Security) is utilized

The configurability of partitions (in terms of capabilities that can be configured) is based on ARINC 653.

The following provides guidance on configuring memory and access:

- Configurations should be based on least-privilege; that is, a partition should be configured to have access to only the memory resources it requires (with the least access permissions required)
- Component-related memory resources (e.g., text and data sections, stacks) should not be shared with other partitions
- When supported by the processing hardware, create separate regions of memory based on component use (e.g., text, read-write data, read-only data, stack) and allocate only access permissions required for that specific use (e.g., execute permissions assigned only to text sections)
- Consider including spare capacity in each allocated memory region; this reduces the possibility of partition and process-level errors, and potentially reduces how often the configuration data will be changed

The following provides guidance on configuring partition time windows:

- An overall module schedule is defined – the duration of the module schedule is fixed and is referred to as the major time frame
- Activation times for each partition are defined by associating the partition with one or more time windows within the module schedule

Note: The overall structure of the OSS configuration data is implementation-defined. Some implementations may support the assignment of partitions to time windows. Other implementations may include offset-duration pairs within the partition definition as a means to assign execution time to the partition.

- Each time window is defined by offset from the start of the module schedule and an expected duration

For partitions configured to use ARINC 653 interfaces, the configuration of the time window also includes whether the start of the time window is intended to be a release point for ARINC 653 periodic processes. Figure 1 (adapted from the ARINC 653 Specification) illustrates an example schedule of a partition within a module schedule. In this example, the partition has been allocated four partition time windows. Two of the allocated partition time windows are annotated as release points for periodic ARINC 653 processes (i.e., the partition's period is half of the major frame time).

- During run-time, the OSS allocates processor execution time based on the module schedule; at the end of the major frame, the OSS starts again at the beginning of the module schedule
- Each partition can be allocated multiple partition time windows within the major frame
This permits some partitions to run at a faster rate than other partitions. By spacing a partition's time windows at precise intervals of the major frame, the partition is scheduled at that rate during run-time.
- Not all time has to be allocated
Intentionally not allocating all of the available time allows for future growth.
- Partitions executing at higher rates establish the framework of the module schedule; lower-rate partitions are scheduled at appropriate sub-intervals between the time windows allocated to high-rate partitions
- A partition's execution time should be assigned using contiguous time durations rather than being split across multiple smaller time durations to avoid unnecessary overhead due to context switches and cache flushes
When this cannot be avoided (due to scheduling requirements of other partitions), additional partition time windows can be defined (to provide the partition sufficient execution time). These additional time windows are typically not configured to be release points for ARINC 653 periodic processes.
- The major frame rate and partition time window rates should take into account processor clock speed, such that the clock speed is a multiple of the major frame and partition time window rates
- Partition time windows and execution rates should be based on UoP and system needs (e.g., additional spare for growth, overhead, partition switch time, caching effects, interrupts):
 - The execution rate is the lowest rate required for a UoP to perform its intended function (e.g., the rate the UoP needs to generate outputs to other UoPs)
Arbitrarily raising the execution rate results in excessive allocation of processor bandwidth. The execution rate typically should not need to be modified when porting to another computing platform (i.e., rate is related to the intended function, not available processor resources).
 - System characteristics (including overhead) must be considered when setting appropriate partition time window durations
- For platforms that support multicore partitions, scheduling of a partition's ARINC 653 processes concurrently on different processor cores is possible (see Section 2.2.2)

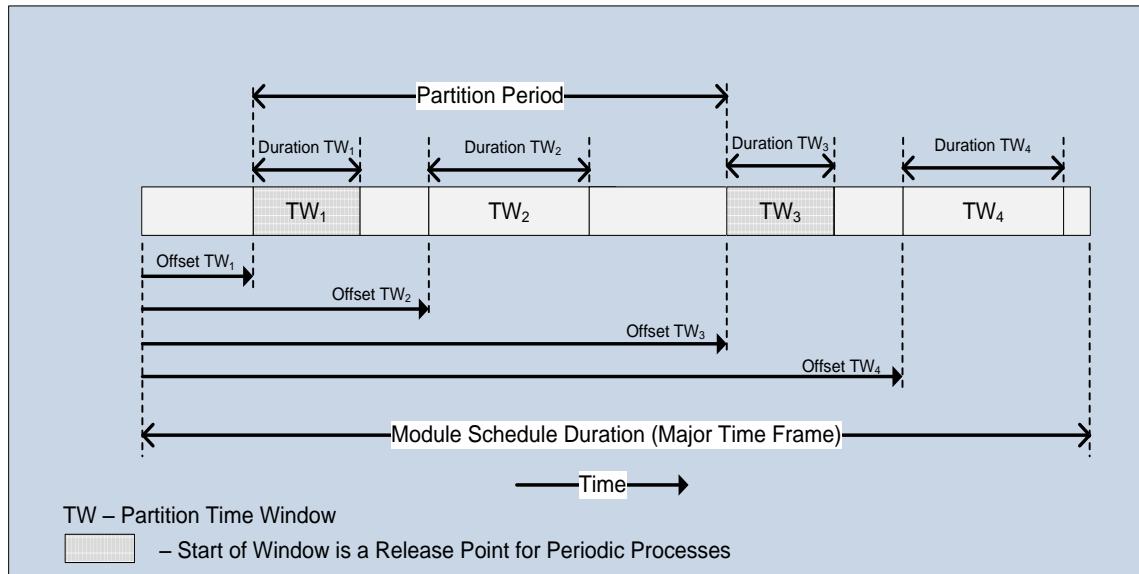


Figure 1: Example Partition Schedule

The FACE Technical Standard includes requirements for OSS support for two or more software components (including UoCs) to be associated with the same time windows. These requirements provide support for capabilities similar to POSIX multiprocess (i.e., software components with unique memory spaces but are scheduled to compete for processor execution time). The capability can be used to design a system where a server (e.g., file system, network stack, fault-independent TSS) in one partition can immediately respond to a client request residing in another partition. This capability also permits cooperative UoCs to be memory isolated but time scheduled together (see also POSIX multiprocess reasoning in Section 2.2.3). Characteristics of this capability include:

- A time window can be uniquely assigned to a single partition (i.e., not shared)
- A time window is shared by multiple software components when each of the associated partitions are configured to have identical time window offset and duration as part of a module schedule

Note: Time windows that partially overlap should be considered a configuration error.
- Within module schedule, a partition can have time windows that are uniquely assigned to it and other time windows that are shared with other partitions
- When time windows are shared, each partition retains its assigned memory and kernel resources (i.e., the memory and kernel resources are not shared)
- When a time window is shared, the active ARINC 653 processes and POSIX threads associated with the partitions assigned to it compete for processor execution time on a priority preemptive basis
- Sharing a time window between ARINC 653 and POSIX standard-based partitions will require ensuring both environments have compatible priority ranges (i.e., the POSIX range and direction should match the ARINC 653 specified range and direction)
- When a time window is shared, the sharing partitions are coupled through the time windows they have in common

This coupling may need to be taken into account when considering which partitions (and the UoCs they contain) may share a time window (e.g., an error in one sharing partition may prevent the processor from being released for use by one of the other sharing partitions; potential covert timing channel could be established between two or more of the sharing partitions).

- When a server is executing a request on behalf of a client, setting the server's thread priority to values that are higher than those used by the client, along with preventing the server from making Application Programming Interface (API) calls that block, ensures that the client is blocked until the request is complete

If a client does have a higher priority thread, this thread becoming eligible to run could interrupt a server thread.

- A server partition can make a request to another server partition
In this case the configurator must ensure that circular dependences are not created; otherwise, it might not be possible to evaluate the system.
- In multicore environments, this capability is not intended to require support for partitions concurrently executing on different cores

Figure 2 illustrates three partitions (Partition A, Partition B, and Partition C) which have two time windows each. The time windows are offset from each other and have different durations. The diagram shows the time windows for each partition and their relative offsets. The total duration of all time windows combined is labeled as the "Module Schedule Duration (Major Time Frame)".

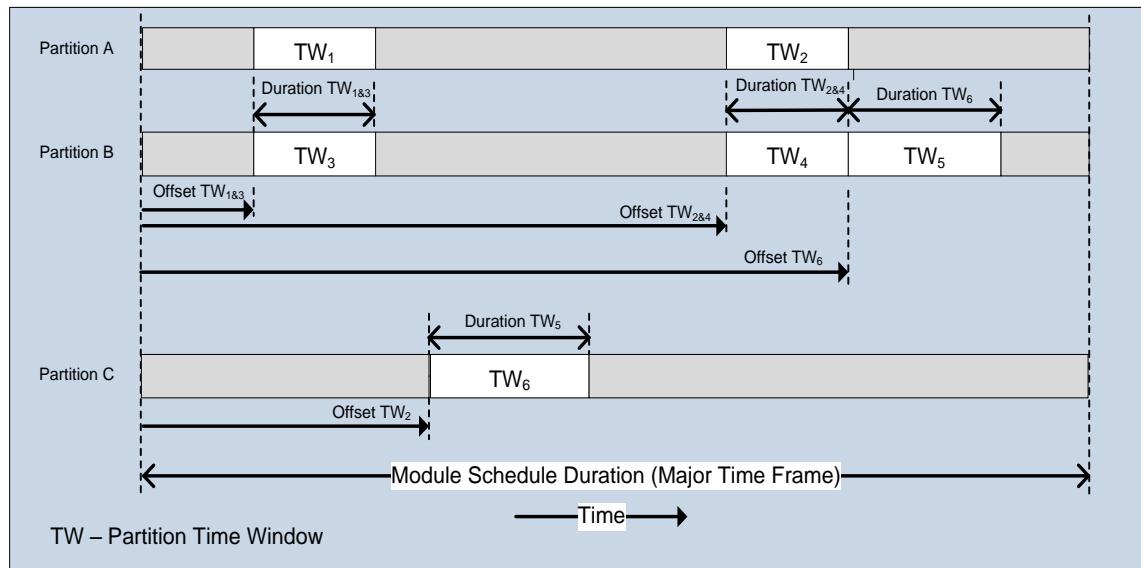


Figure 2: Multiple Partitions Assigned Identical Partition Time Windows

When using periodic ARINC 653 processes or POSIX threads (see Section 2.2.4), the period of the ARINC 653 process or POSIX thread should be precisely the same period as the partition

time windows. If the periods do not match, the release points of the ARINC 653 processes and POSIX threads may drift over time, resulting in the execution sequence being split across multiple partition time windows (starts late in one window, finishes in the next). For ARINC 653 processes, the OSS must support starting periodic ARINC 653 processes relative to the start of partition time windows configured as starting points for periodic processes. For POSIX threads, there are no standard-based POSIX APIs that support synchronizing with the start of partition time windows.

2.2.1.3 *Variability*

The degree of partitioning enforced by the OSS may vary based on the FACE Profiles. Development of components to either the Security or Safety Profiles necessitates the enforcement of partitioning principles (memory, access, resource, and time) in order to have an acceptable level of isolation between components. Memory partitioning is required for an OSS that supports the General Purpose Profile but time partitioning is optional. Time partitioning principles, when supported, are potentially beneficial even to the General Purpose Profile. Partitioning will prevent components from inadvertently impacting other partitions. It also permits each partition to have a safety or security level assignment appropriate to the component(s) allocated (i.e., not all components have to be developed and execute at the same level).

On many processors, memory accesses made by Dynamic Memory Access (DMA) devices are not under the control of the processor's Memory Management Unit (MMU). As such, a DMA device should be considered as being capable of reading and/or writing to anywhere in memory. To prevent violations of partitioning, the developer of the software used to configure the DMA transfer source and destination addresses should satisfy sufficient assurance activities to ensure that the software can be trusted to only utilize address ranges to which it is intended to have access. Some processors include additional hardware capabilities that can be programmed to limit the range(s) of memory the DMA device can access.

2.2.2 **Multicore**

The FACE Technical Standard is intended to be utilized with a variety of computing platforms, including those with multiple processor cores and others limited to a single processor core. The FACE Technical Standard includes APIs that can be used to run ARINC 653 processes (i.e., threads) concurrently on multiple cores. APIs utilized to manage the scheduling of multiple cores within a partition are part of the capabilities defined in Supplement 4 (and later supplements) of the ARINC 653 Part 1 (Required Services) standard.

The POSIX standard does not currently include APIs that can be used to manage the running of POSIX threads concurrently on multiple cores. As such, UoPs developed to utilize multiple processor cores (on hardware platforms that include multiple homogeneous processor cores) must be developed to the ARINC 653 Specification.

When supporting the Supplement 4 version of the ARINC 653 Part 1 standard, the OSS implementation will provide support to configure the number of cores assigned to a partition. Each partition is assigned access to one or more processor cores. When the partition becomes active (see Section 2.2.1) based on the current module schedule, the OSS will schedule the partition's ARINC 653 processes based on the number of cores assigned to the partition. The number of concurrent ARINC 653 processes will be bounded to the number of cores assigned to the partition (i.e., a partition assigned a single core will not have any concurrency).

A UoP manages the concurrent execution of ARINC 653 processes through settings for processor core affinity. Each partition has a range of logical processor cores (starting with 0) that is based on the partition's configured number of cores. A UoP creates its ARINC 653 processes while the partition is in the initialization state. When a process is created, the process has a default affinity for logical core 0. For partitions configured to utilize more than one core, the *Initialize_Process_Core_Affinity()* API is used to configure a process to have an affinity to run on a different logical core. A process can be assigned an affinity for a single logical core only. When the partition is transitioned to normal mode, the ARINC 653 processes will be scheduled on a priority preemptive basis (using the number of cores assigned to the partition) with consideration for the logical core to which they have been assigned an affinity.

For the FACE Technical Standard, only the multicore-related APIs associated with ARINC 653 Part 1 are required. As such, the core affinity of a process can only be set while in the initialization state and only to a single logical core. The additional multicore APIs (that support setting affinity during normal mode and to a value that permits scheduling on any processor assigned to the partition) defined in Supplement 3 (or later supplements) of the ARINC 653 Part 2 standard cannot be utilized by a FACE conformant UoP.

A UoP may create during the initialization state a process level error handler. When created, the operating system schedules this handler to run in response to a fault that has been configured to be handled at the process-level. As a default, logical core 0 will be used when this handler is invoked. The ARINC 653 Part 1 standard (Supplement 4 or later) includes the *Configure_Error_Handler()* API that can be used to:

- Set a specific logical core on which the error handler will be scheduled to run
- Set the error handler to run on any logical core available to the partition (CORE_AFFINITY_NO_PREFERENCE)

In this case the logical core assignment is managed by the operating system (e.g., uses the logical core on which the process with the fault was running). Once a logical core is selected, the process-level error handler will continue to run on this logical core until completion. At its next activation, it could be scheduled to run on the same or a different logical core.

- Control whether other non-faulty processes continue to run concurrently with the error handler or to pause waiting for the error handler to complete

The default behavior (if the API is not utilized) is for the other processes to pause while waiting for the error handler to complete.

The Supplement 3 version of the ARINC 653 Part 1 standard does not include the multicore-related APIs that are part of Supplement 4. All of the APIs defined in Supplement 3 are present in Supplement 4. A UoP developed to Supplement 3 will not require source code modification and will have the same functional behavior when ported to a single-core partition on a multicore platform.

A UoP designed initially for single core operation can be ported directly to a multicore platform by configuring the partition to utilize a single core only. When modifying a single-core UoC to be configured to utilize two or more cores, design analysis/code update activities are likely necessary to ensure correct operation. For example, a single-core UoP may have utilized ARINC 653 process priority as a mutual-exclusion technique instead of the ARINC 653 preemption lock/unlock capabilities. Such a technique may have worked correctly when running on a single

core. Such techniques are likely to fail as a mutual exclusion technique when multiple processes can run concurrently.

2.2.2.1 *Configurability*

The number of cores available for concurrent ARINC 653 process scheduling can be configured on a partition basis. All UoP usage of core assignments is on logical cores (i.e., a UoP has no visibility to the corresponding physical cores). An operating system implementation ensures the same physical cores are utilized for each of a partition's activation in the module schedule.

2.2.2.2 *Variability*

Multicore APIs are only available for ARINC 653 partitions because the POSIX standard does not currently include any multicore APIs that could have been included in the FACE Technical Standard.

2.2.3 **POSIX Multiprocess**

The FACE Technical Standard includes both the ARINC 653 and POSIX operating system standards. The POSIX multiprocess capability is only supported in the Safety Extended and General Purpose Profiles. Support for POSIX multiprocess is optional (i.e., UoCs requiring a larger POSIX API set but not dependent upon POSIX multiprocess should have more support options).

The POSIX standard uses the term “process” to describe an “aggregation of system resources”. This includes address space (memory), schedulable threads as well as various data structures (e.g. registered signal handlers). While a process does have some scheduling attributes, it is not itself schedulable. The process scheduling attributes may have an indirect effect on the threads that are part of the process.

The ARINC 653 Specification uses the term “partition” to refer to an analogous construct. The ARINC 653 partition, however, is schedulable. The ARINC 653 term “process” is logically equivalent to a POSIX thread.

The POSIX multiprocess model is a sort of hybrid sitting between the multi-thread model with no time or space partitioning and the ARINC 653 multi-partition model with both time and space partitioning.

The POSIX multiprocess programming model allows an operating system to make use of the MMU to provide space partitioning, even when multiple processes reside within the same partition. This provides additional safety in that a POSIX application within a single partition cannot accidentally (or maliciously) access memory belonging to another POSIX application within the same partition. This protection typically extends to the operating system as well, preventing an application from accessing memory belonging to the operating system.

The POSIX multiprocess model allows multiple distinct applications with the same safety or security level to coexist within a single partition, allowing them to cooperate temporally while still retaining the safety of separate address spaces. Additionally, a single application can be decomposed into separate sub-processes allowing for increased fault isolation. An error in one sub-process will only impact that process, allowing the other processes to continue executing. An application designed to make use of this ability could continue to function and only suffer from degraded performance or reduced functionality while waiting for the failed process to restart rather than forcing the entire application to restart.

Support for the multiprocess programming model makes available a number of POSIX APIs for application use. These include *fork()* and the “*exec*” family of functions for creating new processes and executing applications. In addition, the “*exit*” and “*wait*” families of functions may be used to terminate a process and report/retrieve process status. The *times()* function provides process-level execution time accounting. Functions to access process IDs and process group IDs are also available in profiles which support multiple processes.

Inter-process communication is possible using a larger selection of methods compared to those available when communicating between partitions although there is some overlap between the methods available to each. Standard UNIX® sockets, named, and unnamed pipes may be used, as well as message queues, shared memory, and semaphores using the “*mq_*”, “*shm_*” and “*sem_*” families of functions. The FACE Technical Standard limits the use of inter-process communications methods to between multiple processes within a single UoC.

2.2.3.1 *Configurability*

The POSIX standard does not define any operating system configuration requirements.

2.2.3.2 *Variability*

Only the Safety Extended and General Purpose Profiles include POSIX multiprocess support. Support for POSIX multiprocess is optional for these profiles.

2.2.4 Multi-Threading

2.2.4.1 *Key Characteristics*

The FACE Technical Standard includes both the ARINC 653 and POSIX operating system standards. Both standards include multi-threading capabilities but differ in size of API set, terminology, and dynamicism.

An important difference which sometimes leads to confusion is that these standards use different terms for what is commonly known as a thread or task. ARINC 653 uses the term process which is logically equivalent to a thread in the POSIX standard. The POSIX standard uses the term process to refer to a schedulable address space which contains multiple threads. ARINC 653 uses the term partition to refer to a schedulable address space which contains multiple ARINC 653 processes. The FACE Technical Standard also applies the term partition to POSIX operational environments (i.e., for resource allocation and isolation; see Section 2.2.1).

ARINC 653 Processes

ARINC 653 processes are created within a partition during warm or cold start by the *CREATE_PROCESS()* API. The process attribute set for a process (name, base priority, stack size, entry point, period, time capacity, and deadline) is specified via the *PROCESS_ATTRIBUTE_TYPE* parameter block passed to the *CREATE_PROCESS()* call – all attributes must be specified (there are no defaults). All ARINC 653 processes share the same resources of the partition they were created in – time, space, and I/O resources.

Once created, a process is not deleted. It may be suspended, resumed, stopped, and started.

Processes are scheduled to execute via the *START()* or *DELAYED_START()* API. Processes then transition to READY state at the first time window in which the partition is scheduled after

Partition transition to NORMAL mode (at the next time window if the partition is already in NORMAL mode).

Processes are created as either a periodic process or aperiodic process. Periodic processes execute at a specified time period (e.g., 10 milliseconds) and aperiodic processes execute until blocked or stopped. A process may have a deadline time specified – the maximum time allowed for a process (including an aperiodic process) to complete an activity cycle.

Processes are scheduled to execute within a partition by the operating system using priority preemptive scheduling. Process priorities are specified as 1 (low) to 239 (high). Priorities 0 and 240 to 255 are reserved for the operating system. Processes generally compete with other processes in the same partition (see also Section 2.2.1).

A process's stack size is of fixed size specified as part of the process's attributes in the *CREATE_PROCESS()* API – i.e., there is no dynamic stack growth. If the thread uses more stack than was reserved, then a run-time error may occur depending on the capabilities provided by the operating system. This may lead to undefined application behavior. It is common for an RTOS to include tools to report information on stack usage, including maximum and current usage.

POSIX Threads

POSIX threads are created and made ready to exist by the *pthread_create()* API. The attribute set for a thread is specified as an argument to this call. The application may use the default attribute values by specifying NULL for the attribute parameter. However, the default attribute set is unspecified by the POSIX standard so this is not recommended. The application should define its own instance of *pthread_attr_t*, initialize it using *pthread_attr_init()* which sets it to the default unspecified state, and then explicitly set every attribute value of interest explicitly with the appropriate *pthread_attr_XXX()* API.

An important thread attribute is the scheduling policy and thread priority. The POSIX standard defines multiple scheduling policies for the set of threads in a process. Each scheduling policy may have a different number of priority levels with the POSIX standard defining the minimum number of priorities to support. Each thread may be assigned a different policy and priority.

The application may also specify the stack size for a POSIX thread. Some implementations may allow the stack to grow dynamically but it should be assumed in real-time, safety-critical environments that thread stacks are of fixed size and should be specified. If the thread uses more stack than was reserved, then a run-time error will occur. This may lead to undefined application behavior. It is common for an RTOS to include tools to report information on stack usage, including maximum and current usage.

In embedded systems, it is common for threads to execute forever, never exiting and repeatedly performing the same actions. This can be implemented using POSIX threads with the *detached* attribute. In this type of system, the set of threads is created during initialization and never changes.

However, POSIX threads may also be created and deleted dynamically. A thread exits when it returns from its main body or invokes *pthread_exit()*. In either case, a thread returns a *void ** pointer value. If a thread is created as attached, then another thread may wait for it to exit using the *pthread_join()* method. The *pthread_join()* method returns the pointer value of the exiting thread. This pointer may be used to access an application-specific structure containing status

information or the results of computations. Note that *pthread_exit()* is not supported in all FACE Profiles.

The POSIX standard does not include a thread attribute that can be used to cause a thread to be scheduled periodically. There are three basic methods that cause code to be scheduled periodically (signal handling, timeouts, and sleeping). Each method (and each API) has trade-offs that may be unsuitable for some applications. Table 1 lists the various APIs and provides details about some of the trade-offs.

Table 1: APIs Allowing Code to be Scheduled Periodically

| API | Activation Type | Clock support (CLOCK_*) | Timer Type | Profile Availability | Additional Information |
|--------------------------------|-----------------------------|-------------------------|------------|----------------------|-------------------------------------|
| Signal handler | Specific signal | Any | Absolute | All | Async-signal safe calls only |
| <i>sigwait()</i> | Specific signal | Any | Absolute | All | |
| <i>sigwaitinfo()</i> | Specific signal | Any | Absolute | All | |
| <i>sem_wait()</i> | Specific signal | Any | Absolute | All | Semaphore required |
| <i>pthread_mutex_lock()</i> | Specific signal | Any | Absolute | All | Mutex required |
| <i>pthread_cond_wait()</i> | Specific signal | Any | Absolute | SB/SE/GP only | Mutex & condition variable required |
| <i>pthread_barrier_wait()</i> | Specific signal | Any | Absolute | GP only | Barrier required |
| <i>pthread_rwlock_rdlock()</i> | Specific signal | Any | Absolute | GP only | Rwlock required |
| <i>pthread_rwlock_wrlock()</i> | Specific signal | Any | Absolute | GP only | Rwlock required |
| <i>pause()</i> | Any handled signal | Any | Absolute | All | See Note. |
| <i>sigsuspend()</i> | Any unmasked handled signal | Any | Absolute | All | See Note. |
| <i>sigtimedwait()</i> | Timeout | Any | Relative | All | |

| API | Activation Type | Clock support (CLOCK_*) | Timer Type | Profile Availability | Additional Information |
|-------------------------------------|-----------------|-------------------------|------------|----------------------|-------------------------------------|
| <i>sem_timedwait()</i> | Timeout | REALTIME only | Absolute | All | Semaphore required |
| <i>pthread_mutex_timedlock()</i> | Timeout | REALTIME only | Absolute | All | Mutex required |
| <i>pthread_cond_timedwait()</i> | Timeout | Any | Absolute | SB/SE/GP only | Mutex & condition variable required |
| <i>pthread_rwlock_timedrdlock()</i> | Timeout | REALTIME only | Absolute | GP only | Rwlock required |
| <i>pthread_rwlock_timedwdlock()</i> | Timeout | REALTIME only | Absolute | GP only | Rwlock required |
| <i>aio_suspend()</i> | Timeout | MONOTONIC only | Absolute | GP only | Async I/O required |
| <i>mq_timedreceive()</i> | Timeout | REALTIME only | Absolute | SB/SE/GP only | Empty mq required |
| <i>mq_timedsend()</i> | Timeout | REALTIME only | Absolute | SB/SE/GP only | Full mq required |
| <i>clock_nanosleep()</i> | Sleep | Any | Either | GP only | |
| <i>nanosleep()</i> | Sleep | REALTIME only | Relative | All | |

Note: “Specific signal” behavior can be achieved by masking all signals except the specific signal.

Some of these APIs are limited to using the REALTIME clock which may be modified by other threads in the system, which could cause the API to timeout earlier or later than intended.

The use of relative timers requires the application to do extra work to calculate the correct delay time each time a new timer is activated.

Many of the APIs are limited as to the set of profiles in which they are supported.

Signal delivery tends to be a more heavyweight activity, leading to more operating system overhead compared to the other methods. Using signal delivery requires the execution of a signal handler before the periodic code can be executed (unless the periodic code is executed inside the signal handler which limits that set of APIs that can be called to those that are async-signal safe).

The following example shows an implementation using *sigwait()*:

```
#include <time.h>
#include <signal.h>
```

```

#define ONE_BILLION 1000000000L /* tv_nsec field must be less than
this */
#define MS_IN_NS     * 1000000    /* millisecond time in nanosecond
units */
#define PERIOD       100 MS_IN_NS /* periodic activation every 100ms */
#define SIGNAL        SIGRTMIN    /* use real-time signal for
notification */

/* this function will be called each time the timer expires */
static void periodic_activation_handler(int signo, siginfo_t *info,
void *ctx)
{
}

int main(int argc, char **argv)
{
    struct itimerspec time;
    struct sigevent    sigev;
    timer_t            timer_id;
    struct sigaction   sa;
    sigset_t           set;
    int                sig;

    /* Get start time for periodic timer */
    if (clock_gettime (CLOCK_MONOTONIC, &time.it_value) == -1)
        /* Handle error */;

    /* Create timer with notification using the real-time signal SIGNAL.
*/
    sigev.sigev_signo      = SIGNAL;
    sigev.sigev_value.sival_int = 0;
    sigev.sigev_notify     = SIGEV_SIGNAL;
    if (timer_create (CLOCK_MONOTONIC, &sigev, &timer_id) == -1)
        /* Handle error */;

    /* Set periodic_activation_handler() as the handler for signal SIGNAL
*/
    sa.sa_sigaction = periodic_activation_handler;
    sa.sa_flags     = SA_SIGINFO | SA_RESTART;
    if (sigemptyset (&sa.sa_mask) == -1)
        /* Handle error */;
    if (sigaction (sigev.sigev_signo, &sa, NULL) == -1)
        /* Handle error */;

    /* Set the periodic timer to fire every PERIOD nanoseconds */
    time.it_value.tv_nsec += PERIOD; /* first activation in PERIOD
nsecs */
    if (time.it_value.tv_nsec >= ONE_BILLION) { /* 0 <= tv_nsec <
ONE_BILLION */
        time.it_value.tv_sec  += 1;
        time.it_value.tv_nsec -= ONE_BILLION;
    }
    time.it_interval.tv_sec  = 0;
    time.it_interval.tv_nsec = PERIOD; /* activation every PERIOD nsecs
*/
    if (timer_settime (timer_id, TIMER_ABSTIME, &time, NULL) == -1)

```

```

    /* Handle error */;

    /* Block waiting for real-time signal SIGNAL to be delivered */
    if (sigemptyset (&set) == -1)
        /* Handle error */;
    if (sigaddset (&set, SIGNAL) == -1)
        /* Handle error */;
    while (1) {
        if (sigwait (&set, &sig) == -1)
            /* Handle error */;

        /* perform periodic processing */

    }
}

```

Figure 3: Periodic Activity Using *sigwait()* Example

The timeout method involves waiting (for a specified period of time) for something that will never occur. When the specified time elapses (or the specified time is reached), the function returns allowing the periodic code to execute.

The following example shows an implementation using *pthread_cond_timedwait()*:

```

#include <time.h>
#include <pthread.h>
#include <errno.h>

#define ONE_BILLION 1000000000L /* tv_nsec field must be less than
this */
#define MS_IN_NS      * 1000000 /* millisecond time in nanosecond
units */
#define PERIOD        100 MS_IN_NS /* periodic activation every 100ms */

int main()
{
    struct timespec time;
    pthread_condattr_t cond_attr;
    pthread_cond_t cond;
    pthread_mutexattr_t mutex_attr;
    pthread_mutex_t mutex;

    /* Get start time for periodic timer */
    if (clock_gettime(CLOCK_MONOTONIC, &time) == -1)
        /* Handle error */;

    /* Prepare the condition variable */
    if (pthread_condattr_init(&cond_attr) != 0)
        /* Handle error */;
    if (pthread_condattr_setclock(&cond_attr, CLOCK_MONOTONIC) != 0)
        /* Handle error */;
    if (pthread_cond_init(&cond, &cond_attr) != 0)
        /* Handle error */;

    /* Prepare the mutex */
    if (pthread_mutexattr_init(&mutex_attr) != 0)
        /* Handle error */;

```

```

if (pthread_mutex_init(&mutex, &mutex_attr) != 0)
    /* Handle error */;
if (pthread_mutex_lock(&mutex) != 0)
    /* Handle error */;

do {
/* Set the timer to fire PERIOD nanoseconds from now */
    time.tv_nsec += PERIOD;
    if (time.tv_nsec >= ONE_BILLION) {
        time.tv_sec += 1;
        time.tv_nsec -= ONE_BILLION;
    }

    /* Block waiting for a condition variable that will never be
signaled */
    if (pthread_cond_timedwait(&cond, &mutex, &time) != ETIMEDOUT)
        /* Handle error */;

    /* perform periodic processing */

} while (1);
}

```

Figure 4: Periodic Activity Using *pthread_cond_timedwait()* Example

The sleep-based APIs are the most straightforward method of generating periodic scheduling of code. These APIs cause the operating system to block the thread until the requested time (elapses or is reached).

The following example shows an implementation using *clock_nanosleep()*:

```

#include <time.h>

#define ONE_BILLION 1000000000L /* tv_nsec field must be less than
this */
#define MS_IN_NS      * 1000000 /* millisecond time in nanosecond
units */
#define PERIOD        100 MS_IN_NS /* periodic activation every 100ms */

int main()
{
    struct timespec time;

    /* Get start time for periodic timer */
    if (clock_gettime(CLOCK_MONOTONIC, &time) == -1)
        /* Handle error */;

    do {
        /* Set the timer to fire PERIOD nanoseconds from now */
        time.tv_nsec += PERIOD;
        if (time.tv_nsec >= ONE_BILLION) {
            time.tv_sec += 1;
            time.tv_nsec -= ONE_BILLION;
        }

        if (clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &time, NULL) != 0)

```

```

    /* Handle error */;

    /* perform periodic processing */

} while (1);
}

```

Figure 5: Periodic Activity Using *clock_nanosleep()* Example

2.2.4.2 *Configurability*

ARINC 653 does not define any configuration data parameters related to ARINC 653 processes.

The POSIX standard does not define anything pertaining to operating system configuration.

2.2.4.3 *Variability*

All of the profiles, including the Security Profile, include support for multi-threading.

When supporting the Security or Safety Profiles, the OSS may be limited to only supporting configuration of static allocations of memory. The safety (and security) of stacks can be improved by utilizing regions of memory where “guard” pages exist at the end of the stack. A guard page is a range of virtual memory addresses to which no corresponding physical memory has been allocated. When accessed (e.g., as part of a stack overflow), the access causes a memory access violation, causing an exception that can be reported to a health monitoring capability (see Chapter 3).

The FACE Safety Extended and General Purpose Profiles support the creation and use of guarded memory regions (e.g., *pthread_attr_setguardsize()*) that can be utilized for stacks. An alternative technique that can work on some architectures is to use a shared memory area for a thread’s stack. The location and size of these stacks can be established at run-time using the techniques described in Section 2.2.7. This information can be utilized with the supported FACE OS APIs (e.g., *CREATE_PROCESS()*, *pthread_attr_setstack()*) to create ARINC processes or POSIX threads that utilize the stacks. Stacks should not be shared (each simultaneously-defined ARINC process and POSIX thread should be allocated a unique stack). Use of this technique ensures the portability of the specification of guarded stacks across all FACE Profiles.

When possible, to improve security and safety, memory allocated for stacks should exclude execute permissions (i.e., stacks should be configured as *read-write-no_execute*).

The POSIX standard does not define the default attributes for threads and thread communication and synchronization objects such as mutexes and condition variables. If an object is created with a NULL pointer for the attribute set or the default values for the attribute set are used, the behavior is undefined. Portable multi-threaded POSIX programs should initialize all of the attributes of all thread and thread communication and synchronization objects.

The POSIX standard defines a minimum number of priorities associated with each policy. It does not define the numeric range. Further, the POSIX standard is ambiguous regarding where a low or high numeric value indicates logically a higher priority. Care should be taken when assigning priorities to threads to ensure the priorities used are available on the target operating system and that they have the expected meaning regarding the importance of the associated threads.

2.2.5 APIs and Profiles

2.2.5.1 Key Characteristics

The FACE Technical Standard defines sets of OS APIs that can be used by a FACE conformant component. A separate set of APIs is defined for each of the profiles (General Purpose, Safety Extended, Safety Base, and Security). APIs are defined for both ARINC 653 and the POSIX standard.

2.2.6 OSS Configuration

2.2.6.1 Key Characteristics

Per the FACE Technical Standard, “An OSS UoC provides support for ARINC 653-defined configuration data types ...”. This applies to all profiles and to systems containing ARINC 653 and POSIX UoCs.

The following configuration data types are defined by the ARINC 653 Specification – defined in the Module XML unless otherwise specified:

- Module Configuration:
 - Module Name
 - Partitions executed on the Module
 - Module schedule(s)
 - A list (or lists) of Partition Time Windows associated with a Module schedule.
Each Time Window specifying offset from start of module schedule, window duration, and a Periodic Process Start (i.e., release point) enable association lists between partitions and partition time windows.
 - Module-Level Health Monitor Table(s)
- Per Partition Configuration:
 - Name and Numerical Identifier
 - Memory Required
 - Period and Duration (optional in some RTOS implementations)
 - Ports (Inter-processor Communications)
 - Partition Health Monitor Table(s)
 - Number of cores available to partition (for multicore configurations)

Note: The above configuration data types have no FACE data model dependencies.

2.2.6.2 Variability

An operating system may implement non-standard extensions to the above and/or derive some of the above. Even though the ARINC 653 configuration is in XML, there is no schema defined by the standard.

The POSIX standard does not define any operating system configuration requirements.

2.2.7 Shared Memory

2.2.7.1 Key Characteristics

The FACE Technical Standard includes support for multiple partitions to have access to the same range of physical memory (i.e., region). For example, this may include multiple partitions being provided read-access to memory containing navigation data. Use of shared memory to communicate between partitions or UoCs is restricted to software in the Transport Services (TS) and Input/Output Services (IOS) Segments. Considerations include:

- Not all partitions require the same level of access
A key attribute is the ability to individually configure the access permissions of each partition. Any partition not configured to access the shared memory must be denied all types of access (read, write, and execute).
- In reader/writer usage scenarios, cache coherency can be an issue if the sharing partitions (or devices in the case that the source of the data is a hardware device such as a DMA engine) use different caching mechanisms

For example, data loss and/or inconsistency could occur if the writer is configured for write-back cache, the reader is configured for uncached, and the writer does not flush its data cache after writing the data.

- In reader/writer usage scenarios, access control and/or notification mechanisms are likely required in order to ensure each reader a consistent view of the data
This is especially important when there are multiple readers, all of whom may be in the process of reading a different message or section of message when the writer has another update.
- If the shared memory includes memory mapped registers for devices, special considerations apply (e.g., not mapping memory to multiple partitions)

2.2.7.2 Configurability

The configurability of shared memory regions is the same as for memory resources associated with the partition (see Section 2.2.1).

2.2.7.3 Variability

The FACE Technical Standard OS requirements may be satisfied by a variety of operating system implementations. These implementations, partially in support of safety and security trade-offs, may vary in methods available to define regions of memory shared between partitions. All implementations are capable of enforcing statically-defined shared memory as part of configuring the partitions for the platform. While some operating systems may have mechanisms to programmatically determine the location of shared memory, all should support use of APIs at run-time to determine location information.

The following APIs can be utilized to determine the location and access permissions for shared memory that has been allocated to a partition:

For ARINC 653-based applications:

- *Get_Memory_Block_Status()* – the user passes in the name of the memory block as assigned in the configuration data and the API returns the location, size, and access mode (read or read-write) of the associated memory block

The memory block names are aliases that are specific to each partition. Two partitions can utilize the same name to refer to different blocks of physical memory and two different names used by two partitions can refer to the same block of physical memory. The configuration data controls the physical memory that corresponds to the named memory block.

For POSIX standard-based applications, a series of APIs are required:

- *stat()* and *fstat()* – provides information about a shared memory object
In particular, stat returns a struct stat which includes fields for the size (st_size) and access permissions (st_mode) of the object associated with the specified file.
- *shm_open()* – opens a shared memory object
- *ftruncate()* – sets the size of the shared memory object
Note that this function is also supported for use with files as part of the file system.
- *mmap()* – provides a pointer to memory that maps to the shared memory object at a particular offset and for a particular length

There are no expected variations in the ARINC 653 API behavior. For the above POSIX APIs, there may be additional behaviors dependent upon the types of memory configuration capabilities supported by a FACE OSS implementation. For the FACE Reference Architecture, the intent of providing these APIs within the Security and Safety Profiles is to provide a means to the application developer to find the location, size, and access permissions of any memory regions (including shared memory regions) allocated for the application's use. The following code fragment example provides a means to access this information that will operate for all FACE OSS implementations regardless of behavior variations:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>

#define SHM_NAME "pathA/Shm_B"

int main(int argc, char **argv)
{
    int             shm_fd;
    struct stat    shm_status;
    void           *shm_ptr;
    mode_t          shm_mode;
    int            shm_access;
    int            shm_prot;

    /*
     * obtain the size and access permissions for a configured
     * shared memory
     */
}
```

```

if (stat (SHM_NAME, &shm_status) == -1)
    /* Handle error */;

/*
 * If application has write permission, then set up to use it
 */
if (shm_status.st_mode & S_IWUSR) {
    shm_mode = S_IWUSR;
    shm_access = O_RDWR;
    shm_prot = PROT_READ | PROT_WRITE;
}
else {
    shm_mode = S_IRUSR;
    shm_access = O_RDONLY;
    shm_prot = PROT_READ;
}

/* create shared memory object */
shm_fd = shm_open (SHM_NAME, shm_access, shm_mode);
if (shm_fd == -1)
    /* Handle error */;

/* size the shared memory based on its configured size */
if (ftruncate (shm_fd, shm_status.st_size) == -1)
    /* Handle error */;

/* obtain pointer to the start of the shared memory region */
shm_ptr = mmap (NULL, shm_status.st_size, shm_prot,
                MAP_SHARED, shm_fd, 0);
if (shm_ptr == MAP_FAILED)
    /* Handle error */;

/* shm_ptr points to the shared memory region */

```

Figure 6: POSIX Shared Memory Initialization Example

When possible, to improve security and safety, memory allocated for shared memory regions should exclude execute permissions (i.e., shared memory regions should be configured as *read-write-no_execute*).

2.2.8 Inter-Partition Communications

The FACE Technical Standard provides the capability of hosting ARINC 653 or POSIX standard-based applications in ARINC 653 partitions and communicating between them (i.e., inter-partition communications).

2.2.8.1 Key Characteristics

Inter-partition communications are the data transfer mechanisms available between partitions. Use of inter-partition communication is restricted to the TSS and IOSS. All FACE conformant PCS components must utilize the TS Interface for all communication outside of the UoP.

2.2.8.2 *Configurability*

ARINC 653 Inter-Partition Communications

For ARINC 653-based applications, each inter-partition information flow should be based on sampling ports or queuing ports. A component may be allocated multiple sampling and queuing ports. The sampling and queuing ports available to a partition are controlled through the configuration data. If a component attempts during initialization to create access to a sampling or queuing port that it has not been configured for, an error will be returned. All sampling and queuing ports must be successfully created during the initialization phase in order for the component to use the port during the normal operational mode.

With sampling ports, the receive side of the port contains the last data message received on the port. When a new data message is received, it atomically replaces the previous data message. Sampling ports are useful for system parameters that periodically vary over time (e.g., aircraft altitude, airspeed, attitude, and heading). The application assigned to the transmit side of the port is responsible for updating the data message at some periodic rate. Users of a sampling port always access the most recent sample of the system parameters. Reading the port again may result in the same data message being read if another data message has not yet been received. Multiple partitions can be assigned access to the receiving side of the same sampling port, if broadcast channels are supported by the OSS. A validity parameter is included with a message when it is read. This parameter is used to indicate whether the message is within or outside the configured refresh period for the port.

With queuing ports, the send and transmit side of the port are capable of containing a queue of a set of messages. When a queuing port is read, the data message is consumed. The queuing port mechanism prevents overflow of the allocated queues. When the send queue is full, the application is able to control through the call parameters whether the invoking ARINC 653 process should block always, block with a timeout, or return. A component that is reading from a queuing port ensures the data's freshness by reading from the port at some periodic rate. By reading all available messages, the application ensures that any new messages are no older than the time period since the port was last checked. If a queuing port is empty, the invoking application is able to control through the call parameters whether the invoking ARINC 653 process should block always, block with a timeout, or return.

POSIX Inter-Partition Communications

For POSIX standard-based applications communicating with ARINC 653-based applications, each inter-partition information flow should be based on sampling ports or queuing ports. For POSIX standard-based applications communicating with other POSIX standard-based applications, each inter-partition information flow should be based on sockets, message queues, or shared memory. A component may have multiple open sockets bound to different endpoints simultaneously. The OSS may limit, based on configuration data, which endpoints a specific application may bind to as part of enforcing access control over the application.

A socket is defined as a file of a particular type that is used as a communications endpoint for process-to-process communication as described in the System Interfaces volume of POSIX.1-2008. An address associated with a socket or remote endpoint, including an address family identifier and addressing information, is specific to that address family and may include multiple parts including network address associated with a host system and an identifier for a specific endpoint.

Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, and are mechanisms called, respectively, a shared memory identifier, shmid, a semaphore identifier, semid, and a message queue identifier, msqid which are returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of POSIX.1-2008, <sys/shm.h>, <sys/sem.h>, and <sys/msg.h> for their descriptions.

Each of the data structures contains both ownership information and an *ipc_perm* structure (see the Base Definitions volume of POSIX.1-2008, <sys/ipc.h>) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The mode member of the *ipc_perm* structure acts as a bit field which determines the permissions.

Real-time systems conforming to one of the POSIX.13 real-time profiles are expected (and desired) to be supported on systems with more than one type or pool of memory (for example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory may be accessible by one or more processors via one or more buses (ports). Memory mapped files, shared memory objects, and the language-specific storage allocation operators (*malloc()* for the ISO C standard, *new* for ISO Ada) fail to provide APIs versatile enough to allow applications to control their utilization of such diverse memory resources. Typed memory allocation and mapping has to coexist with storage allocation operators like *malloc()*, but systems are free to choose how to implement this coexistence.

2.2.8.3 *Variability*

Each component is likely to have one or more data flows with other components in the system. The number of data flows is component-dependent.

For portability, sampling and queuing port names should be selected based on the data flow content, not the source or destination of the data.

For portability, sockets should be established using name service APIs rather than hard coding to fixed addresses (that may conflict with another application's fixed addresses).

2.2.9 **Intra-Partition Communications**

2.2.9.1 *Key Characteristics*

The FACE Technical Standard provides the capability of hosting ARINC 653 or POSIX standard-based applications in ARINC 653 partitions.

ARINC 653 Intra-Partition Communications

Two communication mechanisms are available for intra-partition communication:

1. Partition-defined buffers and blackboards

2. Partition-defined counting semaphores, events, and mutexes

Intra-Partition communication services are divided into five groups:

- **Buffer Services** – a communication object used by processes of a same partition to send or receive messages

The number of total messages and total message size is defined at buffer creation and queued in First In First Out (FIFO) order. The buffer must be created during the partition initialization phase before it can be used. A name is given during buffer creation and is local to the partition and is not an attribute of the partition configuration table. The creation of buffers (e.g., names used, number of buffers) for one partition has no impact on the creation of buffers for other partitions.
- **Blackboard Services** – a communication object used by processes of the same partition to send or receive messages

A blackboard does not use message queues; each new occurrence of a message overwrites any other. The blackboard message size may be variable, but the maximum size is defined at blackboard creation. A blackboard must be created during the partition initialization phase before it can be used. A name is given at blackboard creation; this name is local to the partition and is not an attribute of the partition configuration table. The creation of blackboards (e.g., names used, number of blackboards) for one partition has no impact on the creation of blackboards for other partitions.
- **Counting Semaphore Services** – a counting semaphore is a synchronization object commonly used to provide access to partition resources

The semaphore must be created during the partition's initialization phase before it can be used. A name is given at semaphore creation; this name is local to the partition and is not an attribute of the partition configuration table. The creation of semaphores (e.g., names used, number of semaphores) for one partition has no impact on the creation of semaphores for other partitions. Counting semaphores are well matched to dealing with producer/consumer problems, including those that may exist between threads of different processes, or between a signal handler and a thread. In the former case, there may be little or no memory shared by the processes; in the latter case, communication is not between co-equal threads, but between a thread and an interrupt-like entity.
- **Event Services** – an event is a synchronization object used to notify the occurrence of a condition to processes that may be waiting for condition to occur

An event must be created during the partition's initialization phase before it can be used. A name is given at event creation; this name is local to the partition and is not an attribute of the partition configuration table. The creation of events (e.g., names used, number of events) for one partition has no impact on the creation of events for other partitions.
- **Mutexes** – a mutex is a special form of semaphore used to provide mutual exclusion

POSIX Intra-Partition Communications

Two communication mechanisms are available for intra-partition communication:

1. Partition-defined message queues
2. Partition-defined counting semaphores and mutexes

Intra-Partition communication services are divided into three groups:

- **Message Queue Services** – a communication object used by processes of a same partition to send or receive messages
The number of total messages and total message size is defined at message queue creation and queued in priority order. A name is given during message queue creation and may be local to a process or shared across multiple processes. The FACE Technical Standard restricts the use of message queues for intra-UoC communication except for the TSS and IOS. The creation of message queues (e.g., number of message buffers) for one partition has no impact on the creation of message queues for other partitions.
- **Counting Semaphore Services** – a counting semaphore is a synchronization object commonly used to provide access to partition resources
The semaphore must be created during the partition’s initialization phase before it can be used. A name is given at semaphore creation; this name is local to the partition and is not an attribute of the partition configuration table. The creation of semaphores (e.g., names used, number of semaphores) for one partition has no impact on the creation of semaphores for other partitions. Counting semaphores are well matched to dealing with producer/consumer problems, including those that may exist between threads of different processes, or between a signal handler and a thread. In the former case, there may be little or no memory shared by the processes; in the latter case, communication is not between co-equal threads, but between a thread and an interrupt-like entity. It is for these reasons that POSIX.1-2008 allows semaphores to be used by threads.
- **Mutexes** – a mutex is a special form of semaphore used to provide mutual exclusion
The POSIX standard also provides condition variables which can be used in conjunction with mutexes. Mutexes can be effectively used with and without priority inheritance, priority ceiling, and other attributes to synchronize threads that share memory. The efficiency of their implementation is comparable to or better than that of other synchronization primitives that are sometimes harder to use (for example, binary semaphores). Furthermore, there is at least one known implementation of Ada tasking that uses these primitives. Mutexes and condition variables together constitute an appropriate, sufficient, and complete set of inter-thread synchronization primitives.

2.2.9.2 Configurability

The memory size given in the configuration table should include the memory size necessary to manage all the ARINC 653 objects/resources in a partition.

The POSIX standard does not define anything pertaining to operating system configuration.

FACE conformant operating systems providing ARINC 653 capabilities will provide configuration mechanisms for memory resources used by both POSIX and ARINC 653 partitions.

2.2.9.3 Variability

ARINC 653 Part 1 Supplement 4 (2015) adds mutexes as an intra-partition communication and synchronization.

The POSIX standard does not define the default attributes for a mutex when it is created.

The FACE Technical Standard specifies that POSIX message queues are restricted to intra-partition communication only in the Security and Safety Base Profiles. For the Safety Extended and General Purpose Profiles, message queues may also be utilized within multiprocess UoCs.

2.2.10 Local Memory Allocation/Deallocation

2.2.10.1 Key Characteristics

Applications developed to the FACE Technical Standard may require memory allocation and deallocation mechanisms at run-time.

A key safety/security principle is to utilize memory buffers (a set of fixed-size elements) and/or pools provided to the partition for specific purposes instead of common memory heaps that are used by all parts of the application. When common memory heaps are used, several security/safety concepts may be violated:

- Resource reuse within the partition

In a multi-level secure application, part of the application is dealing with data of one classification, while other portions are dealing with another classification. If both portions utilize services that draw resources from a common heap/pool, and these resources may be returned to the heap/pool for use by other parts of the application, any state remaining in the resource may result in information leakage when reused by other portions of the application (at another classification level). This needs to be prevented (potential breach of confidentiality).
- Allocation from a memory heap may be open-ended in terms of number of bytes required
When open-ended, there is a greater potential for memory exhaustion.
- Traditional heaps for which allocation and deallocation services are provided have potential for memory fragmentation (deallocated memory range is located between other memory ranges that are still in use, limiting reallocation of the memory range to requests of the memory ranges length or smaller)

Memory fragmentation leads to the potential for either temporary loss of function (while resolving fragmentation) or complete loss of function (due to resources no longer being available). Loss of function results in violating the application's availability. The use of pools can still result in impacting the application's availability (due to pool resource being exhausted).
- Minimization of duplicate services (to minimize the exposure surface that must be evaluated, verified, and/or proven)

Services should be re-entrant (e.g., to prevent potential loss of function resulting from two or more threads attempting to utilize the services, with later threads corrupting internal data that the earlier threads were dependent upon).

For some applications, the size of some data structures is dependent upon other application and/or platform variables whose values are not known until run-time. These applications require the ability to create memory structures during initialization whose sizes are based on parameters that are not known until run-time.

Some applications will require access to memory regions whose locations, size, and access permissions are not known until run-time.

Note that some systems may utilize a variable-sized memory region (whose size is discovered at run-time) to resolve variable-size memory structure allocation at initialization.

2.2.10.2 *Configurability*

Regions of memory (i.e., range of consecutive locations) can be allocated and/or deallocated at run-time, and be configured on a partition-by-partition basis. Each partition may have memory regions of various sizes that can be utilized locally during run-time.

2.2.10.3 *Variability*

Trade-offs with regard to local memory allocation and deallocation that an application may take into account include:

- Using only static allocation

Here, an application selects as part of the design effort the size of each data structure. As part of compilation, linking, and integration, memory resources are reserved to contain the allocated data structures. During partition initialization, memory resources for the data structures are allocated. During application run-time, the data structures are utilized:

- The data structure can be a pool of elements that are allocated for a particular use and returned when that use is complete
- When all of the allocation decisions were completed by a single designer, and the data structure was adequately sized for worst-case memory requirements, there will be no memory allocation issues as there are no other consumers whose memory usage could result in memory exhaustion during run-time
- The system cannot adjust the size of the allocation based on system state that is only known at run-time

Note: Systems developed within the FACE Safety or Security Profiles should consider using static allocation.

- Using dynamic allocation only during initialization:

- The system can adjust the size of the allocation based on system state that is known at run-time during the application's initialization phase
- Since the allocation is dynamic, there could be system states which result in all of the memory available for dynamic allocation being exhausted

Since the system is just initializing, there may be fewer hazards associated with this erroneous condition. For a given mission (e.g., flight) the system state should not change. A power-cycle during operational flight is feasible. If the power-cycle during flight results in a different system state that has larger memory requirements, a memory exhaustion could occur during operational flight that did not occur when the aircraft was first powered up on the ground.

Note: Systems developed within the FACE Safety or Security Profiles should consider using dynamic allocation only during initialization.

- Permitting full dynamic allocation and deallocation during run-time:

- Some implementations may result in significant memory fragmentation

Recovery of memory fragmentation could have non-deterministic behaviors. For example, when a system has to search for available space for the next allocation, and moves current allocations to a different location in order to make sufficient consecutive memory locations available for the next allocation.

- Some implementations may result in significant memory under-utilization (e.g., fixed-size allocation that accounts for the largest possible memory allocation, but may be requested an allocation significantly smaller)

Note: Systems developed within the FACE Safety Extended or General Purpose Profiles may consider using full dynamic allocation and deallocation during run-time. FACE Security and Safety Base Profiles do not support *free()*.

If allocations and deallocations of memory are occurring during run-time for divergent uses, prediction of worst-case memory requirements may be infeasible. This could result in a memory exhaustion hazard occurring at any time a memory allocation is requested during run-time.

When possible, to improve security and safety, memory allocated for local memory regions should exclude execute permissions (i.e., local memory regions should be configured as *read-write-no_execute*).

2.2.11 Operating System Input/Output (I/O) Support

2.2.11.1 Key Characteristics

Applications developed to the FACE Technical Standard may require data to be sent and received at run-time over a variety of interfaces. The operating system provides a mechanism to interface with hardware devices as well as control the access to those hardware devices. This interface and access control is typically implemented in a device driver using the operating system's APIs which may or may not be common across all operating systems conforming to the FACE Technical Standard. Depending on the FACE Profile that the operating system is targeting, the access control may be implemented via file device access control or via operating system XML configuration files. POSIX standard-based operating systems such as Linux® use file device access control with user and group used to vet access. Operating systems targeting safety or security tend to use XML configuration files to control partition-level access to devices. In a POSIX operational environment executing on an ARINC 653 operating system, there may be a combination of both. Device driver implementation details are specific to the chosen operating system. The UoC developer has the option to choose the operating system vendor-defined interfaces to the hardware or to interface with the IOSS to abstract the OS APIs.

2.2.11.2 Configurability

An IOSS may access an operating system device driver via standard APIs or by using device-specific libraries.

Access control may be provided via file protections.

On ARINC 653-based operating systems, access to specific hardware devices is configured via the operating system-specific configuration XML file.

2.2.11.3 Variability

When transitioning an application from a non-time-partitioned to a time-partitioned system, the application is eligible to execute only during its partition time windows (see Section 2.2.1). In

the non-time-partitioned environment, an application may have been able to immediately respond to some events (e.g., interrupts). However, the application will now only respond during its allocated partition time windows. In the FACE Reference Architecture, this level of interaction has mainly been addressed by the IOS (i.e., components dedicated to handling specific I/O devices and the interactions with them) and TS (i.e., reliable endpoints for data transfer). As such, transitioning such an application requires some re-architecting effort to separate the I/O and transport operations from the other functional capabilities.

Device-specific software is prone to portability challenges due to variations in hardware platforms and operating system-specific device drivers.

2.2.12 File System

2.2.12.1 Key Characteristics

The FACE Reference Architecture may include file system capabilities. The file system capabilities permit applications to:

- Store data into and retrieve data from non-volatile storage devices (i.e., data preserved across power cycles)
- Separate data into uniquely identified files
- Separate sets of files into uniquely identified directory structures
- Have an amount of storage (referred to as a volume) reserved for the application's use
- Share stored information between components

File systems used within the FACE Reference Architecture are intended to support atomic updates. This characteristic permits the consistency of data structures stored within a file to be retained. Updates to a file within the defined atomic update size of the file system are ensured by the file system to either be completely stored, or to not be stored at all (e.g., when a power failure occurs during the write-cycle). This characteristic prevents a data structure (whose size is less than the defined atomic update size) from having some contents from a previous state and other partial contents from an update.

The FACE Technical Standard does not include requirements on safety or security attributes of the stored data when the equipment is in a powered off state. A particular installation may impose additional requirements on the data at rest (e.g., encryption to prevent unauthorized access, check sequences to detect and prevent use of data corrupted by device failures).

2.2.12.2 Configurability

The non-volatile storage capabilities can be allocated and sub-divided based on the configuration data. The individual allocations of non-volatile storage are referred to as file system volumes.

For each file system volume, the following can be configured:

- **Access Permissions**

The use of access permissions permits control over which FACE conformant components can access a particular file system volume. The access permissions of each file system volume are configurable on a per-partition basis:

- For the Safety Profile, the access permissions of a file system volume permit at most one partition with write permissions and one or more partitions with read permissions (there are no executing privileges for volumes used by the Safety Profile)
- For the General Purpose Profile, additional access permissions may be supported (access permissions defined for the Safety Profile are the minimum requirements)
- Multiple partitions can be allocated read-access permissions for the same file system volume
- A read-only file system volume is configured by not allocating write permission to any partition
- All access permissions are explicitly defined as part of the configuration data (i.e., there are no default access permissions)
- **Alias Name**
The use of an alias permits a FACE conformant component to not be impacted when reused across platforms. The file system configuration data includes specification of a volume alias name:
 - Each partition can have a unique alias name for each of the volumes to which it has access

2.2.12.3 *Variability*

File system APIs have not been included as part of the Security Profile. The Security Profile use case includes applications that may be required to process data of different security classifications within a single partition. A typical application use of a file system will include multiple files that are open and being utilized simultaneously. When coupled with the feature-rich capabilities of a typical file system (including temporary data cache management), ensuring data separation through a file system from a single partition is not feasible. This limitation can be managed by separating file system-related operations into other partitions that are only on one side or the other of a security boundary. This permits a file system to have to only handle a single classification of data, permitting data separation to be managed.

The FACE Safety Profile only permits a single partition to have write access to a volume. This restriction in behavior prevents data consistency and synchronization issues that could occur when multiple writers from different partitions attempt to utilize a single file (e.g., only part of the data is written to a log file during a partition's time window). Data from multiple partitions can be written into a single file by designing the contributing partitions to send their data to the partition that has write permissions for the volume. This partition will collect the data from the other partitions and write it into the file on their behalf. The writing partition ensures that complete data sets from the contributing partitions are written into the file.

2.2.13 **Networking**

The FACE Technical Standard defines a set of optional requirements that allow the sockets APIs that are available to POSIX standard-based partitions to be extended from use as an inter-partition communications mechanism to include communications over an external environment (e.g., Ethernet). The supported sets of Requests for Comment (RFCs) are defined in the FACE Technical Standard. The sets of RFCs are grouped into the following categories:

- Basic internetwork capabilities

- This covers IP/UDP and multicast (including multicast loopback within the local host)
- TCP capabilities
- IPv6 capabilities
- IPv4/IPv6 Transition Mechanisms

There are other RFCs that have been developed that cover additional networking-related topics. These RFCs were not included in the FACE Technical Standard because there are no standards-based definitions for the functional APIs that would be necessary for these topics. This lack of standards-based interfaces is the primary reason used to exclude these RFCs from the FACE Technical Standard.

All FACE Profiles include the sockets APIs necessary for UDP-based communications. The additional sockets APIs necessary for TCP-based communications are included in the Safety Extended and General Purpose Profiles only.

2.2.13.1 *Configurability*

The FACE Technical Standard does include configuration capabilities for managing inter-partition communications between POSIX standard-based partitions. The FACE Technical Standard does not include configuration capabilities for managing message transfers over an external network.

2.2.13.2 *Variability*

All operating system implementations will provide support for sockets APIs that can be utilized for inter-partition communications by POSIX standard-based partitions. When networking is supported, the same sockets APIs are utilized to communicate over the external network.

The FACE Technical Standard defines support for networking as a separate option (i.e., an OSS implementation does not have to support it). When networking is supported, IPv4 network capabilities are supported. Within support for networking, support for IPv4 network capabilities is a separate option. Within support for networking, support for integrated IPv4/IPv6 network transition mechanism is a separate option.

2.2.14 *Time Management*

The FACE Technical Standard includes multiple capabilities related to time. Time management is a critical part of real-time application construction. The FACE Technical Standard supports capabilities in the following time management areas:

- Date and Time
- Timezone
- System Uptime
- Relative Time
- Absolute Time

The FACE Technical Standard supports setting and retrieving the date and time using POSIX APIs. The ARINC 653 Specification does not include support for date and time. The FACE

Technical Standard requires that only one partition be able to set the date and time. Other partitions may only read the date and time.

The POSIX standard specifies that the environment variable TZ contains the current timezone for a process. Environment variables are accessed using the *getenv()* and *setenv()* methods. Each process has its own set of environment variables and modifying one using the *setenv()* method only impacts the single process. This is not likely to be a concern on an embedded system which is likely either not concerned with date and time or uses GMT and ignores the timezone.

The length of time since the system was powered on is commonly referred to as “uptime”. The POSIX standard does not directly support uptime but it provides CLOCK_MONOTONIC which cannot be set and may be the same as uptime.

Embedded systems are often constructed in terms of relative time such as a timeout which must occur 50 milliseconds from now. The exact uptime or date and time at the timeout is not relevant.

Absolute time may be specified based on the date and time or, on a POSIX system, in terms of CLOCK_MONOTONIC. Rate monotonic loops may be constructed using the POSIX method *clock_nanosleep()* and the CLOCK_MONOTONIC timer.

2.2.15 Health Monitoring/Fault Management

2.2.15.1 Key Characteristics

Although Health Monitoring is primarily associated with ARINC 653, the FACE Technical Standard supports the development of Health Monitoring/Fault Management (HMFN) in the POSIX environment. HMFN provides standardized methods for detecting, reporting, and handling faults, failures, current state, and state change of platform components. Component code is instrumented to report events (fault, failure, current state, and state change) to the HMFN. This approach separates component execution activities from error handling activities, making the code easier to develop and understand.

Once an event is received, the HMFN component reacts to the event by performing a set of actions that are specified by the System Integrator at configuration time.

One possible action is to log the event, allowing a profile/trace of the execution activities to be created. The log should be saved either in memory or in non-volatile memory. If the event logs need to be saved in a file, then a partition should be created that is responsible for managing the logs.

Since the logs are saved in memory, which is a finite resource, a log management policy should be created in the case a log becomes full. The two standard policies are either to stop logging or to wrap the log by overwriting the oldest event.

2.2.15.2 Configurability

The connection between each event that can be reported by a platform and the set of actions to take should be configurable, allowing the same HMFN to react differently from one instantiation to another.

The log management policies should also be configurable.

2.2.15.3 *Variability*

ARINC 653 clearly defines the Health Monitoring architectures and the basic instrumentation points. The implementer may augment the number of instrumentation points to provide a safer solution.

2.3 FACE POSIX Guidance

The FACE Technical Standard defines subsets of the POSIX standard. The use of these restricted subsets should not be a problem for new software written with the FACE Technical Standard in mind. However, they could negatively impact the effort required to make existing POSIX software FACE conformant. Existing POSIX software will not be designed to use only the subset of APIs allowed in any particular profile.

This section provides guidance on the rules that guided the development of these profiles as well as on the use of specific POSIX methods.

2.3.1 POSIX API Subsetting Golden Rules

When constructing the FACE POSIX Profiles, a set of guidelines or Golden Rules were used to determine which methods were allowed into the profiles. These Golden Rules are based on industry best practices with a view to safety and security certification requirements.

2.3.1.1 *POSIX Version and Profiles*

FACE POSIX Profiles were originally based on POSIX 2003 minimizing optional methods. POSIX 2013 made several POSIX 2003 optional methods mandatory. The FACE Technical Standard has since been updated to POSIX 2013 but methods that were optional in POSIX 2003 but required in POSIX.1-2008 were not added to FACE Profiles as part of this update.

PSE 51-54 are profiles of POSIX 2003. If a method was not included in the largest profile (PSE 54), then it is less likely to be included in any FACE Profile.

2.3.1.2 *Avoiding Use of Standard In, Out, and Error*

Methods which assume the use of stdin, stdout, or stderr are only available in the General Purpose Profile.

This is due to the lower profiles being focused on multi-partition deployments in which it is unclear which partition(s) would have access to these output streams and how these partitions would share the single IO stream. It is also unclear who the “user” is for many embedded systems.

Applications written to the lower profiles should be careful to open specific files to which to write messages.

2.3.1.3 *Minimize Redundancy*

To accommodate testing and efficiency, the FACE Technical Standard minimizes replicated code and unnecessary wrappers. This minimizes the exposed surface for what needs to be audited in security and safety certification testing.

Examples: Use `setvbuf()` instead of `setbuf()`. Use `fseek()` instead of `rewind()`.

2.3.1.4 *Reentrancy and Thread Safety*

When available, reentrant versions (`*_r`) of function calls are supported and non-reentrant time versions are only permitted in the General Purpose Profile.

Examples: Use `gmtime_r()` instead of `gmtime()`. Use `localtime_r()` instead of `localtime()`.

2.3.1.5 *Buffer Overflow Protection*

Methods which are prone to programming errors which result in buffer overflows are only supported in the General Purpose Profile. If POSIX 2003 included an alternative method with a length parameter, the FACE Profiles include that version.

Example: Use `snprintf()` versus `sprintf()` which is only supported in the General Purpose Profile.

2.3.1.6 *Networking*

TCP and supporting methods are only in the Safety Extended and General Purpose Profiles.

If using sockets to communicate exclusively between partitions on the same platform, it is safe to assume that the communication is reliable and UDP may be used.

Examples: `accept()` and `listen()` are only in the Safety Extended and General Purpose Profiles.

2.3.1.7 *Resource Allocation and Management*

Safety Base and Security assume allocation of resources during initialization with no dynamic reallocations:

- No close, destroy, or unlink (removal of resources) in Security or Safety Base
 - No exiting threads in Security or Safety Base (no removal of resources)
 - No removal of resources – random change in execution in Security or Safety Base
 - No removing of threads in Security or Safety Base (no reallocation of resources)
 - Memory allocation but no deallocation in Security or Safety Base
- Use dedicated pools of memory to mimic dynamic allocation and deallocation while ensuring hard limits are satisfied.

2.3.1.8 *Variable Argument List*

Methods with variable argument lists are only in Safety Extended and General Purpose Profiles due to complexity of implementation and associated potential certification challenges. The variable argument list also present a potential opportunity for unbounded data input.

Examples: `vfprintf()` and `vsnprintf()` are only in the Safety Extended and General Purpose Profiles.

2.3.1.9 *Locale*

The supported methods defined in `<locale.h>` are only present in the General Purpose Profile. These were not included in lower profiles to reduce the complexity of the API set provided. The default locale (C standard locale) is used for the other profiles.

In a complex system, this is a system integration issue. Locale is not expected to change dynamically in an avionics environment. Additionally, there are operational questions such as should an avionics system change locales when it changes locations? What locales should be supported? In multi-partition applications, is it possible for some of the partitions to adjust for locale and others not? These questions are system integration and certification challenges.

2.3.1.10 *Device Access Methods*

Only an IOS is allowed to use non-file system/time management device access methods.

Example: <termios.h> functions are not included in the FACE Technical Standard.

2.3.1.11 *Easily Analyzable Flow of Execution*

Methods that provide means to change program flow are mainly supported in the General Purpose Profile (and to a lesser extent in the Safety Extended).

Some methods for signals and cancellation are supported in all profiles. Usages should be carefully examined during code review/audits.

Example: The *setjmp()* and *longjmp()* methods are only in the General Purpose Profile.

2.3.1.12 *Optional POSIX Features*

Methods and features which were optional in POSIX 2003 tend to only be in the General Purpose Profile if they are included at all in the FACE Profiles.

2.3.1.13 *Undefined POSIX Characteristics*

The POSIX standard has multiple places where there are undefined characteristics, and these are a barrier to portability. These characteristics include behavior, required variables, and dependencies. The FACE Profiles attempt to limit the inclusion of these in the standard. The undefined areas are not completely avoidable due to cases such as the pthread default set attributes being undefined.

Example: SCHED_OTHER is not included in any profile.

2.3.1.14 *Multiple Processes*

Multiple process support is only associated with the Safety Extended and General Purpose Profiles.

2.3.1.15 *Wide Character Support*

Wide characters are not supported in any profile.

2.3.1.16 *Long Double and Complex Math*

The long double and complex math related methods are only in the General Purpose Profile.

2.3.1.17 *Avoid Obsolete and Deprecated Methods*

All profiles avoid inclusion of obsolete or deprecated methods.

2.3.1.18 Minimize API Exposure Space in Security Profile

The Security Profile is constructed towards high-assurance security applications that may need to consider the implications that can occur through the exposed API set.

Example: All methods using *FILE* * are not included in the Security Profile.

2.3.1.19 Symbolic Links

Symbolic links are supported only in the Safety Extended and General Purpose Profiles.

Example: *lstat()* is defined only for the Safety Extended and General Purpose Profiles.

2.3.1.20 Normal and Real-Time Signals

Normal signals are only in the Safety Extended and General Purpose Profiles. Real-Time Signals are in all profiles. This is due to the complexity of dispatching normal signals.

Example: *kill()* and *pthread_kill()* are not in the Security or Safety Base Profiles.

2.3.1.21 Calendar Time

Services related to calendar time are not available in the Security Profile.

Example: *time()* is not available.

2.3.2 The POSIX Standard and Undefined Behavior

This section describes some of the places in the POSIX standard where implementation-defined behavior is allowed. The POSIX standard defines three terms to denote implementation variance:

- Implementation-defined

“This definition is analogous to that of the ISO C standard and, together with “undefined” and “unspecified”, provides a range of specification of freedom allowed to the interface implementer.”¹

- Undefined
- Unspecified

There is a lengthy discussion following the *unspecified* entry in the Rationale section. Because of the precision of the text and the importance of understanding it, it is quoted here from POSIX.1-2008:²

The definitions for “unspecified” and “undefined” appear nearly identical at first examination, but are not. The term “unspecified” means that a conforming application may deal with the unspecified behavior, and it should not care what the outcome is. The term “undefined” says that a conforming application should not do it because no definition is provided for what it does (and implicitly it would care what the outcome was if it tried it). It is important to remember,

¹ Taken from http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xbd_chap01.html#tag_21_01_07.

² See http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xbd_chap01.html#tag_21_01_14.

however, that if the syntax permits the statement at all, it must have some outcome in a real implementation.

Thus, the terms “undefined” and “unspecified” apply to the way the application should think about the feature. In terms of the implementation, it is always “defined” – there is always some result, even if it is an error. The implementation is free to choose the behavior it prefers.

This also implies that an implementation, or another standard, could specify or define the result in a useful fashion. The terms apply to POSIX.1-2008 specifically.

The term “implementation-defined” implies requirements for documentation that are not required for “undefined” (or “unspecified”). Where there is no need for a conforming program to know the definition, the term “undefined” is used, even though “implementation-defined” could also have been used in this context. There could be a fourth term, specifying “this standard does not say what this does; it is acceptable to define it in an implementation, but it does not need to be documented”, and undefined would then be used very rarely for the few things for which any definition is not useful. In particular, implementation-defined is used where it is believed that certain classes of application will need to know such details to determine whether the application can be successfully ported to the implementation. Such applications are not always strictly portable, but nevertheless are common and useful; often the requirements met by the application cannot be met without dealing with the issues implied by “implementation-defined”. In some places the text refers to facilities supplied by the implementation that are outside the standard as implementation-supplied or implementation-provided. This is not intended to imply a requirement for documentation. If it were, the term “implementation-defined” would have been used.

In many places POSIX.1-2008 is silent about the behavior of some possible construct. For example, a variable may be defined for a specified range of values and behaviors are described for those values; nothing is said about what happens if the variable has any other value. That kind of silence can imply an error in the standard, but it may also imply that the standard was intentionally silent and that any behavior is permitted. There is a natural tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent. Silence is intended to be equivalent to the term “unspecified”.

As can be seen from the above, to identify every possible place where FACE software portability is impacted by this would require examining each part of the standard that applies to each FACE Profile. Neither the FACE Technical Standard nor this document attempt to provide exhaustive coverage of these cases.

2.3.2.1 Default Attributes for Threads and Concurrency Objects

The POSIX thread and thread concurrency objects are created with an attribute structure which specifies various characteristics for the instance of the object being created. A program using one of these APIs may specify the attribute set to the create call as NULL to explicitly indicate it wishes to use the default attribute set. Alternatively, it may initialize and assign specific values to the individual attributes for that object class. This pattern is used by the following POSIX threading and concurrency objects:

- Threads
- Barriers
- Condition variables

- Mutexes
- Read/write locks

The portability issue is that the default set of values for the individual attributes is not specified in the POSIX standard. This allows different POSIX implementations to use completely different default attributes for the various object classes. These differences can result in different execution behavior when porting software from one operating system to another.

Thus, any program using a NULL on the initialization/create call for one of the above object classes does not know the attributes associated with the object. The following example illustrates invoking *pthread_create* with the thread attribute argument (second argument) set to NULL. This results in *myThread* being created with the default attributes.

```
#include <pthread.h>

void *myThreadBody(void *argument)
{
    return NULL;
}

void example(void)
{
    int rc;
    pthread_t myThread;

    rc = pthread_create( &myThread, NULL, myThreadBody, NULL );
}
```

Figure 7: *pthread_create()* with NULL Attributes

At least it is possible to search for and identify the places where object create and initialization calls are passed NULL and are assigned the default attributes. Unfortunately, even if the program uses explicit attribute initialization but does not completely initialize every value, those untouched values will still have the defaults. The following example is similar to the previous one but explicitly passes in *myAttr* containing the desired attributes for *myThread*. Unfortunately, initializing it via *pthread_attr_init* and not setting any values is defined by the POSIX standard to result in the same default attribute set being associated with *myThread*.

```
#include <pthread.h>

void *myThreadBody(void *argument)
{
    return NULL;
}

void example(void)
{
    int             rc;
    pthread_t       myThread;
    pthread_attr_t  myAttr;

    pthread_attr_init( &myAttr );
    /* set anything less than the full set of variables */
```

```

    rc = pthread_create( &myThread, &myAttr, myThreadBody, NULL );
}

```

Figure 8: Partial Initialization of Thread Attributes

2.3.2.2 Scheduling Policies and Priorities

The POSIX standard defines four scheduling policies and their characteristics.³ These policies are described at a conceptual level with behavior and minimum characteristics. The methods defined to select and configure the characteristics of a policy for a thread are prototyped in `<sched.h>` and `<pthread.h>`. The policies are as follows:

- `SCHED_FIFO` – FIFO scheduling policy
- `SCHED_RR` – round robin scheduling policy
- `SCHED_SPORADIC` – sporadic server scheduling policy
- `SCHED_OTHER` – another implementation

`SCHED_FIFO` is a priority-based scheduler with full preemptability of threads. There is no timeslicing. The POSIX standard requires that there is a minimum of at least 32 priority levels for this scheduling policy defined by the values returned by `sched_get_priority_max()` and `sched_get_priority_min()`.

`SCHED_RR` is an extension of the `SCHED_FIFO` policy but adds the condition that when the running thread has been running for a time period longer than the value returned by `sched_rr_get_interval()` it is removed from the front of its priority thread list and placed at the end. The POSIX standard requires that there is a minimum of at least 32 priority levels for this scheduling policy defined by the values returned by `sched_get_priority_max()` and `sched_get_priority_min()`.

`SCHED_SPORADIC` is a policy where a thread is associated with a period and CPU time budget. The thread executes with greater importance (higher numerical priority) until it has consumed the allotted CPU time budget. It then executes at the specified lower numeric priority until the next replenishment period starts. This is an optional POSIX scheduling policy that is not required as part of any FACE Profile.

`SCHED_OTHER` is used to indicate that an implementation-defined scheduling policy is to be used. This policy may be the same as `SCHED_FIFO` or `SCHED_RR` but this is implementation-defined. The POSIX standard requires that the priority levels for this scheduling policy are within the range defined by the values returned by `sched_get_priority_max()` and `sched_get_priority_min()`. However, there is no minimum number of priorities required.

The Conformance Test Suite (CTS) ensures that a program is only using the allowed set of APIs but, since it cannot detect what the parameters are, there are multiple potential portability issues related to scheduling policies and priorities:

- A program may use `SCHED_SPORADIC` – this would be a point of non-conformance that is not detected by the CTS
- A program may use `SCHED_OTHER` – this would be conformant but result in undefined behavior

³ See http://pubs.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_08.html#tag_02_08_04_01.

- The POSIX standard allows for an implementation to define other scheduling policies – use of these would be a point of non-conformance not detected by the CTS
- A program must be careful to use no more than 32 priority levels
- A program must dynamically calculate its thread priorities to ensure they fall within the range defined by the values returned by *sched_get_priority_max()* and *sched_get_priority_min()* – there is no guarantee on the specific values of the priorities
- The priority ranges for each scheduling policy may or may not overlap

2.3.3 POSIX Header Files Per FACE Profile

This section defines the POSIX header files which should be expected to be present in each FACE Profile.

Table 2: Header Files Per Profile

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| <aio.h> | No | No | No | INCL |
| <arpa/inet.h> | INCL | INCL | INCL | INCL |
| <assert.h> | No | No | No | INCL |
| <complex.h> | No | No | No | No |
| <cpio.h> | No | No | No | No |
| <ctype.h> | INCL | INCL | INCL | INCL |
| <dirent.h> | No | INCL | INCL | INCL |
| <dlfcn.h> | No | No | No | No |
| <errno.h> | INCL | INCL | INCL | INCL |
| <fcntl.h> | No | INCL | INCL | INCL |
| <fenv.h> | No | No | No | INCL |
| <float.h> | INCL | INCL | INCL | INCL |
| <fmtmsg.h> | No | No | No | No |
| <fnmatch.h> | No | No | No | No |
| <ftw.h> | No | No | No | No |
| <glob.h> | No | No | No | No |
| <grp.h> | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| <iconv.h> | No | No | No | No |
| <inttypes.h> | No | INCL | INCL | INCL |
| <iso646.h> | No | No | No | INCL |
| <langinfo.h> | No | No | No | No |
| <libgen.h> | No | No | No | No |
| <limits.h> | INCL | INCL | INCL | INCL |
| <locale.h> | No | No | No | No |
| <math.h> | INCL | INCL | INCL | INCL |
| <monetary.h> | No | No | No | No |
| <mqueue.h> | No | INCL | INCL | INCL |
| <ndbm.h> | No | No | No | No |
| <net/if.h> | No | No | No | INCL |
| <netdb.h> | INCL | INCL | INCL | INCL |
| <netinet/in.h> | INCL | INCL | INCL | INCL |
| <netinet/tcp.h> | No | No | INCL | INCL |
| <nl_types.h> | No | No | No | No |
| <poll.h> | No | No | No | No |
| <pthread.h> | INCL | INCL | INCL | INCL |
| <pwd.h> | No | No | No | No |
| <regex.h> | No | No | No | No |
| <sched.h> | INCL | INCL | INCL | INCL |
| <search.h> | No | No | No | No |
| <semaphore.h> | INCL | INCL | INCL | INCL |
| <setjmp.h> | No | No | No | INCL |
| <signal.h> | INCL | INCL | INCL | INCL |
| <spawn.h> | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|-----------------|--------------------|------------------------|------------------------|
| <stdarg.h> | No | No | INCL | INCL |
| <stdbool.h> | INCL | INCL | INCL | INCL |
| <stddef.h> | INCL | INCL | INCL | INCL |
| <stdint.h> ⁴ | INCL | INCL | INCL | INCL |
| <stdio.h> | No | INCL | INCL | INCL |
| <stdlib.h> | INCL | INCL | INCL | INCL |
| <string.h> | INCL | INCL | INCL | INCL |
| <strings.h> | No | No | No | No |
| <stropts.h> | No | No | No | No |
| <sys/ioctl.h> | INCL | INCL | INCL | INCL |
| <sys/ipc.h> | No | No | No | No |
| <sys/mman.h> | INCL | INCL | INCL | INCL |
| <sys/msg.h> | No | No | No | No |
| <sys/resource.h> | No | No | No | No |
| <sys/select.h> | No | INCL | INCL | INCL |
| <sys/sem.h> | No | No | No | No |
| <sys/shm.h> | No | No | No | No |
| <sys/socket.h> | INCL | INCL | INCL | INCL |
| <sys/stat.h> | INCL | INCL | INCL | INCL |
| <sys/statvfs.h> | No | No | No | No |
| <sys/time.h> | No | No | No | No |
| <sys/times.h> | No | No | No | No |
| <sys/types.h> | INCL | INCL | INCL | INCL |
| <sys/uio.h> | No | No | No | No |
| <sys/un.h> | No | No | No | No |

⁴ <stdint.h> is caught by the requirement to provide the C99 fixed-width types, but it has no methods so is not an obvious inclusion.

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| <sys/utsname.h> | No | No | No | No |
| <sys/wait.h> | No | No | No | No |
| <syslog.h> | No | No | No | No |
| <tar.h> | No | No | No | No |
| <termios.h> | No | No | No | No |
| <tgmath.h> | No | No | No | No |
| <time.h> | INCL | INCL | INCL | INCL |
| <trace.h> | No | No | No | No |
| <ulimit.h> | No | No | No | No |
| <unistd.h> | INCL | INCL | INCL | INCL |
| <utime.h> | No | No | No | No |
| <utmpx.h> | No | No | No | No |
| <wchar.h> | No | No | No | No |
| <wctype.h> | No | No | No | No |
| <wordexp.h> | No | No | No | No |

2.3.4 Allowed POSIX Constants per FACE Profile

This section details the POSIX constants that are specified to be defined by each header file and whether a programmer should expect them to be available based upon the FACE Profile to which they are programming.

The FACE Profiles are defined in the FACE Technical Standard primarily in terms of API methods available per profile. For many constants, this is sufficient to determine if they should be available in a particular profile. However, for other constants, making this judgment requires knowledge of the specific capabilities required to be provided by the RTOS and what is reasonable to expect based on the target applications for that profile.

The information in this section is guidance and was constructed based on the following rules:

- If supporting APIs are not in a profile, then related constants should not be either
- If multiprocess capability is required to support the functionality, then it is not to be included in the Security and Safety Base Profiles
(As an alternative: include and define acceptable behavior in lower profiles.)

- If the behavior is undefined in the POSIX standard, then it should not be required in any FACE Profile
- If optional (like SCHED_SPORADIC), it tends to only be required in the General Purpose Profile, if at all
- Symbolic links are optional for the Security and Safety Base Profiles
- Rule on constants: does saying “yes” mean the operating system is required to support it? The POSIX standard allows an error to be returned in many cases to indicate a feature is not supported. The FACE Technical Standard does not change this. This means the constant may be present, but the operating system may not actually support the feature. This is particularly true regarding error and signal numbers.
- Fill in all boxes – use No or INCL
If this is merged into the Technical Standard, “No” will be replaced with blank. This helps track that everything in this document has had a review and decision. Blank could mean No or not reviewed in this document.
- Braces around a constant is a POSIX convention that means that it can vary by implementation – these often refer to numeric limits on the implementation
- In general, X/Open System Interface (XSI)-related constants will not be supported, but will be looked at on a one-by-one basis

2.3.4.1 <aio.h>

Table 3: <aio.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| AIO_ALLDONE | No | No | No | INCL |
| AIO_CANCELED | No | No | No | INCL |
| AIO_NOTCANCELED | No | No | No | INCL |
| LIO_NOP | No | No | No | INCL |
| LIO_NOWAIT | No | No | No | INCL |
| LIO_READ | No | No | No | INCL |
| LIO_WAIT | No | No | No | INCL |
| LIO_WRITE | No | No | No | INCL |

2.3.4.2 <arpa/inet.h>

Table 4: <arpa/inet.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| INET_ADDRSTRLEN | INCL | INCL | INCL | INCL |
| INET6_ADDRSTRLEN | INCL | INCL | INCL | INCL |

2.3.4.3 <assert.h>

This header file does not define any constants.

2.3.4.4 <complex.h>

Table 5: <complex.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| complex | No | No | No | INCL |
| _Complex_I | No | No | No | INCL |
| imaginary | No | No | No | INCL |
| _Imaginary_I | No | No | No | INCL |
| I | No | No | No | INCL |

2.3.4.5 <cpio.h>

Table 6: <cpio.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------|----------|-------------|-----------------|-----------------|
| C_IRUSR | No | No | No | No |
| C_IWUSR | No | No | No | No |
| C_IXUSR | No | No | No | No |
| C_IRGRP | No | No | No | No |
| C_IWGRP | No | No | No | No |
| C_IXGRP | No | No | No | No |
| C_IROTH | No | No | No | No |
| C_IWOTH | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| C_IXOTH | No | No | No | No |
| C_ISUID | No | No | No | No |
| C_ISGID | No | No | No | No |
| C_ISVTX | No | No | No | No |
| C_ISDIR | No | No | No | No |
| C_ISFIFO | No | No | No | No |
| C_ISREG | No | No | No | No |
| C_ISBLK | No | No | No | No |
| C_ISCHR | No | No | No | No |
| C_ISCTG | No | No | No | No |
| C_ISLNK | No | No | No | No |
| C_ISSOCK | No | No | No | No |
| MAGIC | No | No | No | No |

2.3.4.6 <ctype.h>

Note: `_toupper()` and `_tolower()` are function-like macros and addressed as part of the POSIX API Profiles.

Table 7: <ctype.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|-----------------|--------------------|------------------------|------------------------|
| <code>_toupper</code> | No | No | No | No |
| <code>_tolower</code> | No | No | No | No |

2.3.4.7 <dirent.h>

This header file does not define any constants.

2.3.4.8 <dlfcn.h>

This header file is not required by any FACE Profiles.

Table 8: <dlfcn.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| RTLD_LAZY | No | No | No | No |
| RTLD_NOW | No | No | No | No |
| RTLD_GLOBAL | No | No | No | No |
| RTLD_LOCAL | No | No | No | No |

2.3.4.9 **<errno.h>**

Note: The POSIX standard does not define use of *setsockopt()* for multicast support. The ETOOMANYREFS is required to support returning an error when the maximum number of multicast groups has already been added to a socket.

Table 9: <errno.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| E2BIG | INCL | INCL | INCL | INCL |
| EACCES | INCL | INCL | INCL | INCL |
| EADDRINUSE | INCL | INCL | INCL | INCL |
| EADDRNOTAVAIL | INCL | INCL | INCL | INCL |
| EAFNOSUPPORT | INCL | INCL | INCL | INCL |
| EAGAIN | INCL | INCL | INCL | INCL |
| EALREADY | INCL | INCL | INCL | INCL |
| EBADF | INCL | INCL | INCL | INCL |
| EBADMSG | No | No | No | No |
| EBUSY | INCL | INCL | INCL | INCL |
| ECANCELED | INCL | INCL | INCL | INCL |
| ECHILD | INCL | INCL | INCL | INCL |
| ECONNABORTED | INCL | INCL | INCL | INCL |
| ECONNREFUSED | INCL | INCL | INCL | INCL |
| ECONNRESET | INCL | INCL | INCL | INCL |
| EDEADLK | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| EDESTADDRREQ | INCL | INCL | INCL | INCL |
| EDOM | INCL | INCL | INCL | INCL |
| EDQUOT | INCL | INCL | INCL | INCL |
| EEXIST | INCL | INCL | INCL | INCL |
| EFAULT | INCL | INCL | INCL | INCL |
| EFBIG | INCL | INCL | INCL | INCL |
| EHOSTUNREACH | INCL | INCL | INCL | INCL |
| EIDRM | No | No | No | No |
| EILSEQ | INCL | INCL | INCL | INCL |
| EINPROGRESS | INCL | INCL | INCL | INCL |
| EINTR | INCL | INCL | INCL | INCL |
| EINVAL | INCL | INCL | INCL | INCL |
| EIO | INCL | INCL | INCL | INCL |
| EISCONN | INCL | INCL | INCL | INCL |
| EISDIR | INCL | INCL | INCL | INCL |
| ELOOP | No | No | INCL | INCL |
| EMFILE | INCL | INCL | INCL | INCL |
| EMLINK | INCL | INCL | INCL | INCL |
| EMSGSIZE | INCL | INCL | INCL | INCL |
| EMULTIHOP | No | No | No | No |
| ENAMETOOLONG | INCL | INCL | INCL | INCL |
| ENETDOWN | INCL | INCL | INCL | INCL |
| ENETRESET | INCL | INCL | INCL | INCL |
| ENETUNREACH | INCL | INCL | INCL | INCL |
| ENFILE | INCL | INCL | INCL | INCL |
| ENOBUFS | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| ENODATA | No | No | No | No |
| ENODEV | INCL | INCL | INCL | INCL |
| ENOENT | INCL | INCL | INCL | INCL |
| ENOEXEC | INCL | INCL | INCL | INCL |
| ENOLCK | INCL | INCL | INCL | INCL |
| ENOLINK | No | No | No | No |
| ENOMEM | INCL | INCL | INCL | INCL |
| ENOMSG | No | No | No | No |
| ENOPROTOOPT | INCL | INCL | INCL | INCL |
| ENOSPC | INCL | INCL | INCL | INCL |
| ENOSR | No | No | No | No |
| ENOSTR | No | No | No | No |
| ENOSYS | INCL | INCL | INCL | INCL |
| ENOTCONN | INCL | INCL | INCL | INCL |
| ENOTDIR | INCL | INCL | INCL | INCL |
| ENOTEMPTY | INCL | INCL | INCL | INCL |
| ENOTSOCK | INCL | INCL | INCL | INCL |
| ENOTSUP | INCL | INCL | INCL | INCL |
| ENOTTY | INCL | INCL | INCL | INCL |
| ENXIO | INCL | INCL | INCL | INCL |
| EOPNOTSUPP | No | No | INCL | INCL |
| EOVERFLOW | INCL | INCL | INCL | INCL |
| EPERM | INCL | INCL | INCL | INCL |
| EPIPE | INCL | INCL | INCL | INCL |
| EPROTO | No | No | INCL | INCL |
| EPROTONOSUPPORT | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| EPROTOTYP | INCL | INCL | INCL | INCL |
| ERANGE | INCL | INCL | INCL | INCL |
| EROFS | INCL | INCL | INCL | INCL |
| ESPIPE | INCL | INCL | INCL | INCL |
| ESRCH | INCL | INCL | INCL | INCL |
| ESTALE | INCL | INCL | INCL | INCL |
| ETIME | No | No | No | No |
| ETIMEDOUT | INCL | INCL | INCL | INCL |
| ETXTBSY | INCL | INCL | INCL | INCL |
| EWOULDBLOCK | INCL | INCL | INCL | INCL |
| EXDEV | INCL | INCL | INCL | INCL |
| ETOOMANYREFS | INCL | INCL | INCL | INCL |

2.3.4.10 <fcntl.h>

Note: The *open()* and *fcntl()* APIs use this header file. The *fcntl()* API is in the Safety Extended and General Purpose Profiles only (i.e., all of the F_* constants). For sockets, *posix_devctl()* must be used.

Note: The S_I* constants from <sys/stat.h> which are symbolic names for file modes in *mode_t* are defined by <fcntl.h> and <sys/stat.h>. See <sys/stat.h> for details. The SEEK_* constants are also defined in <stdio.h> for use with other APIs.

Table 10: <fcntl.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| F_DUPFD | No | No | INCL | INCL |
| F_DUPFD_CLOEXEC | No | No | INCL | INCL |
| F_GETFD | No | No | INCL | INCL |
| F_SETFD | No | No | INCL | INCL |
| F_GETFL | No | No | INCL | INCL |
| F_SETFL | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| F_GETLK | No | No | INCL | INCL |
| F_SETLK | No | No | INCL | INCL |
| F_SETLKW | No | No | INCL | INCL |
| F_GETOWN | No | No | No | No |
| F_SETOWN | No | No | No | No |
| FD_CLOEXEC | No | No | INCL | INCL |
| F_RDLCK | No | No | INCL | INCL |
| F_UNLCK | No | No | INCL | INCL |
| F_WRLCK | No | No | INCL | INCL |
| SEEK_SET | No | No | INCL | INCL |
| SEEK_CUR | No | No | INCL | INCL |
| SEEK_END | No | No | INCL | INCL |
| O_CLOEXEC | No | No | INCL | INCL |
| O_CREAT | No | INCL | INCL | INCL |
| O_DIRECTORY | No | No | No | No |
| O_EXCL | No | INCL | INCL | INCL |
| O_NOCTTY | No | No | No | No |
| O_NOFOLLOW | No | No | INCL | INCL |
| O_TRUNC | No | INCL | INCL | INCL |
| O_TTY_INIT | No | No | No | No |
| O_APPEND | No | INCL | INCL | INCL |
| O_DSYNC | No | No | No | No |
| O_NONBLOCK | No | No | INCL | INCL |
| O_RSYNC | No | No | No | No |
| O_SYNC | No | No | No | No |
| O_ACCMODE | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|-----------------|--------------------|------------------------|------------------------|
| O_EXEC | No | No | No | No |
| O_RDONLY | No | INCL | INCL | INCL |
| O_RDWR | No | INCL | INCL | INCL |
| O_SEARCH | No | No | No | No |
| O_WRONLY | No | INCL | INCL | INCL |
| AT_FDCWD | No | No | No | No |
| AT_EACCESS | No | No | No | No |
| AT_SYMLINK_NOFOLLOW | No | No | No | No |
| AT_SYMLINK_FOLLOW | No | No | No | No |
| AT_REMOVEDIR | No | No | No | No |
| POSIX_FADV_DONTNEED | No | No | No | No |
| POSIX_FADV_NOREUSE | No | No | No | No |
| POSIX_FADV_NORMAL | No | No | No | No |
| POSIX_FADV_RANDOM | No | No | No | No |
| POSIX_FADV_SEQUENTIAL | No | No | No | No |
| POSIX_FADV_WILLNEED | No | No | No | No |

2.3.4.11 <fenv.h>

Note: The floating point exception constants FE_DIVBYZERO, FE_INEXACT, FE_INVALID, FE_OVERFLOW, and FE_UNDERFLOW are only defined if the implementation supports that floating point exception. The constant FE_ALL_EXCEPT is a bitwise-inclusive OR of the supported floating point exception constants.

Table 11: <fenv.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FE_DIVBYZERO | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FE_INEXACT | No | No | No | INCL |
| FE_INVALID | No | No | No | INCL |
| FE_OVERFLOW | No | No | No | INCL |
| FE_UNDERFLOW | No | No | No | INCL |
| FE_ALL_EXCEPT | No | No | No | INCL |
| FE_DOWNWARD | No | No | No | INCL |
| FE_TONEAREST | No | No | No | INCL |
| FE_TOWARDZERO | No | No | No | INCL |
| FE_UPWARD | No | No | No | INCL |
| FE_DFL_ENV | No | No | No | INCL |

2.3.4.12 <float.h>

Table 12: <float.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FLT_RADIX | INCL | INCL | INCL | INCL |
| FLT_MANT_DIG | INCL | INCL | INCL | INCL |
| DBL_MANT_DIG | INCL | INCL | INCL | INCL |
| LDBL_MANT_DIG | INCL | INCL | INCL | INCL |
| DECIMAL_DIG | No | No | No | No |
| FLT_DIG | INCL | INCL | INCL | INCL |
| DBL_DIG | INCL | INCL | INCL | INCL |
| LDBL_DIG | INCL | INCL | INCL | INCL |
| FLT_MIN_EXP | INCL | INCL | INCL | INCL |
| DBL_MIN_EXP | INCL | INCL | INCL | INCL |
| LDBL_MIN_EXP | INCL | INCL | INCL | INCL |
| FLT_MIN_10_EXP | INCL | INCL | INCL | INCL |
| DBL_MIN_10_EXP | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| LDBL_MIN_10_EXP | INCL | INCL | INCL | INCL |
| FLT_MAX_EXP | INCL | INCL | INCL | INCL |
| DBL_MAX_EXP | INCL | INCL | INCL | INCL |
| LDBL_MAX_EXP | INCL | INCL | INCL | INCL |
| FLT_MAX_10_EXP | INCL | INCL | INCL | INCL |
| DBL_MAX_10_EXP | INCL | INCL | INCL | INCL |
| LDBL_MAX_10_EXP | INCL | INCL | INCL | INCL |
| FLT_MAX | INCL | INCL | INCL | INCL |
| DBL_MAX | INCL | INCL | INCL | INCL |
| LDBL_MAX | INCL | INCL | INCL | INCL |
| FLT_EPSILON | INCL | INCL | INCL | INCL |
| DBL_EPSILON | INCL | INCL | INCL | INCL |
| LDBL_EPSILON | INCL | INCL | INCL | INCL |
| FLT_MIN | INCL | INCL | INCL | INCL |
| DBL_MIN | INCL | INCL | INCL | INCL |
| LDBL_MIN | INCL | INCL | INCL | INCL |
| FLT_EVAL_METHOD | No | No | No | No |
| FLT_ROUNDS | No | No | No | INCL |

2.3.4.13 <fmtmsg.h>

The methods in this header file are not included in any FACE Profile.

Table 13: <fmtmsg.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| MM_HARD | No | No | No | No |
| MM_SOFT | No | No | No | No |
| MM_FIRM | No | No | No | No |
| MM_APPL | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| MM_UTIL | No | No | No | No |
| MM_OPSYS | No | No | No | No |
| MM_RECOVER | No | No | No | No |
| MM_NRECOV | No | No | No | No |
| MM_HALT | No | No | No | No |
| MM_ERROR | No | No | No | No |
| MM_WARNING | No | No | No | No |
| MM_INFO | No | No | No | No |
| MM_NOSEV | No | No | No | No |
| MM_PRINT | No | No | No | No |
| MM_CONSOLE | No | No | No | No |
| MM_NULLLBL | No | No | No | No |
| MM_NULLSEV | No | No | No | No |
| MM_NULLMC | No | No | No | No |
| MM_NULLTXT | No | No | No | No |
| MM_NULLACT | No | No | No | No |
| MM_NULLTAG | No | No | No | No |
| MM_OK | No | No | No | No |
| MM_NOTOK | No | No | No | No |
| MM_NOMSG | No | No | No | No |
| MM_NOCON | No | No | No | No |

2.3.4.14 <fnmatch.h>

The methods in this header file are not included in any FACE Profile.

Table 14: <fnmatch.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FNM_NOMATCH | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FNM_PATHNAME | No | No | No | No |
| FNM_PERIOD | No | No | No | No |
| FNM_NOESCAPE | No | No | No | No |

2.3.4.15 <ftw.h>

Note: <ftw.h> defines the symbolic names and file type test macros for the *st_mode* field of the *stat* structure that are defined in <sys/stat.h>. See <sys/stat.h> for details.

Table 15: <ftw.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FTW_F | No | No | No | No |
| FTW_D | No | No | No | No |
| FTW_DNR | No | No | No | No |
| FTW_DP | No | No | No | No |
| FTW_NS | No | No | No | No |
| FTW_SL | No | No | No | No |
| FTW_SLN | No | No | No | No |
| FTW_PHYS | No | No | No | No |
| FTW_MOUNT | No | No | No | No |
| FTW_DEPTH | No | No | No | No |
| FTW_CHDIR | No | No | No | No |

2.3.4.16 <glob.h>

Table 16: <glob.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| GLOB_APPEND | No | No | No | No |
| GLOB_DOOFFS | No | No | No | No |
| GLOB_ERR | No | No | No | No |
| GLOB_MARK | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| GLOB_NOCHECK | No | No | No | No |
| GLOB_NOESCAPE | No | No | No | No |
| GLOB_NOSORT | No | No | No | No |
| GLOB_ABORTED | No | No | No | No |
| GLOB_NOMATCH | No | No | No | No |
| GLOB_NOSPACE | No | No | No | No |

2.3.4.17 **<grp.h>**

This header file does not define any constants and the methods defined in it are not in any FACE Profile.

2.3.4.18 **<iconv.h>**

This header file does not define any constants and the methods defined in it are not in any FACE Profile.

2.3.4.19 **<inttypes.h>**

Note: Names with “N” in them in the POSIX standard expand to 8, 16, 32, and 64-bit variants based on the fixed width types for multiple *printf* specifiers.

Table 17: <inttypes.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| PRIId8 | No | INCL | INCL | INCL |
| PRIId16 | No | INCL | INCL | INCL |
| PRIId32 | No | INCL | INCL | INCL |
| PRIId64 | No | INCL | INCL | INCL |
| PRIIdLEAST8 | No | INCL | INCL | INCL |
| PRIIdLEAST16 | No | INCL | INCL | INCL |
| PRIIdLEAST32 | No | INCL | INCL | INCL |
| PRIIdLEAST64 | No | INCL | INCL | INCL |
| PRIIdFAST8 | No | INCL | INCL | INCL |
| PRIIdFAST16 | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| PRIIdFAST32 | No | INCL | INCL | INCL |
| PRIIdFAST64 | No | INCL | INCL | INCL |
| PRIIdMAX | No | INCL | INCL | INCL |
| PRIIdPTR | No | INCL | INCL | INCL |
| PRIi8 | No | INCL | INCL | INCL |
| PRIi16 | No | INCL | INCL | INCL |
| PRIi32 | No | INCL | INCL | INCL |
| PRIi64 | No | INCL | INCL | INCL |
| PRIiLEAST8 | No | INCL | INCL | INCL |
| PRIiLEAST16 | No | INCL | INCL | INCL |
| PRIiLEAST32 | No | INCL | INCL | INCL |
| PRIiLEAST64 | No | INCL | INCL | INCL |
| PRIiFAST8 | No | INCL | INCL | INCL |
| PRIiFAST16 | No | INCL | INCL | INCL |
| PRIiFAST32 | No | INCL | INCL | INCL |
| PRIiFAST64 | No | INCL | INCL | INCL |
| PRIiMAX | No | INCL | INCL | INCL |
| PRIiPTR | No | INCL | INCL | INCL |
| PRIo8 | No | INCL | INCL | INCL |
| PRIo16 | No | INCL | INCL | INCL |
| PRIo32 | No | INCL | INCL | INCL |
| PRIo64 | No | INCL | INCL | INCL |
| PRIoLEAST8 | No | INCL | INCL | INCL |
| PRIoLEAST16 | No | INCL | INCL | INCL |
| PRIoLEAST32 | No | INCL | INCL | INCL |
| PRIoLEAST64 | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| PRIoFAST8 | No | INCL | INCL | INCL |
| PRIoFAST16 | No | INCL | INCL | INCL |
| PRIoFAST32 | No | INCL | INCL | INCL |
| PRIoFAST64 | No | INCL | INCL | INCL |
| PRIoMAX | No | INCL | INCL | INCL |
| PRIoPTR | No | INCL | INCL | INCL |
| PRIu8 | No | INCL | INCL | INCL |
| PRIu16 | No | INCL | INCL | INCL |
| PRIu32 | No | INCL | INCL | INCL |
| PRIu64 | No | INCL | INCL | INCL |
| PRIuLEAST8 | No | INCL | INCL | INCL |
| PRIuLEAST16 | No | INCL | INCL | INCL |
| PRIuLEAST32 | No | INCL | INCL | INCL |
| PRIuLEAST64 | No | INCL | INCL | INCL |
| PRIuFAST8 | No | INCL | INCL | INCL |
| PRIuFAST16 | No | INCL | INCL | INCL |
| PRIuFAST32 | No | INCL | INCL | INCL |
| PRIuFAST64 | No | INCL | INCL | INCL |
| PRIuMAX | No | INCL | INCL | INCL |
| PRIuPTR | No | INCL | INCL | INCL |
| PRIx8 | No | INCL | INCL | INCL |
| PRIx16 | No | INCL | INCL | INCL |
| PRIx32 | No | INCL | INCL | INCL |
| PRIx64 | No | INCL | INCL | INCL |
| PRIxLEAST8 | No | INCL | INCL | INCL |
| PRIxLEAST16 | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| PRIxLEAST32 | No | INCL | INCL | INCL |
| PRIxLEAST64 | No | INCL | INCL | INCL |
| PRIxFast8 | No | INCL | INCL | INCL |
| PRIxFast16 | No | INCL | INCL | INCL |
| PRIxFast32 | No | INCL | INCL | INCL |
| PRIxFast64 | No | INCL | INCL | INCL |
| PRIxMAX | No | INCL | INCL | INCL |
| PRIxPTR | No | INCL | INCL | INCL |
| PRIX8 | No | INCL | INCL | INCL |
| PRIX16 | No | INCL | INCL | INCL |
| PRIX32 | No | INCL | INCL | INCL |
| PRIX64 | No | INCL | INCL | INCL |
| PRIXLEAST8 | No | INCL | INCL | INCL |
| PRIXLEAST16 | No | INCL | INCL | INCL |
| PRIXLEAST32 | No | INCL | INCL | INCL |
| PRIXLEAST64 | No | INCL | INCL | INCL |
| PRIxFast8 | No | INCL | INCL | INCL |
| PRIxFast16 | No | INCL | INCL | INCL |
| PRIxFast32 | No | INCL | INCL | INCL |
| PRIxFast64 | No | INCL | INCL | INCL |
| PRIxMAX | No | INCL | INCL | INCL |
| PRIxPTR | No | INCL | INCL | INCL |
| SCNd8 | No | No | INCL | INCL |
| SCNd16 | No | No | INCL | INCL |
| SCNd32 | No | No | INCL | INCL |
| SCNd64 | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SCNdLEAST8 | No | No | INCL | INCL |
| SCNdLEAST16 | No | No | INCL | INCL |
| SCNdLEAST32 | No | No | INCL | INCL |
| SCNdLEAST64 | No | No | INCL | INCL |
| SCNdFAST8 | No | No | INCL | INCL |
| SCNdFAST16 | No | No | INCL | INCL |
| SCNdFAST32 | No | No | INCL | INCL |
| SCNdFAST64 | No | No | INCL | INCL |
| SCNdMAX | No | No | INCL | INCL |
| SCNdPTR | No | No | INCL | INCL |
| SCNi8 | No | No | INCL | INCL |
| SCNi16 | No | No | INCL | INCL |
| SCNi32 | No | No | INCL | INCL |
| SCNi64 | No | No | INCL | INCL |
| SCNiLEAST8 | No | No | INCL | INCL |
| SCNiLEAST16 | No | No | INCL | INCL |
| SCNiLEAST32 | No | No | INCL | INCL |
| SCNiLEAST64 | No | No | INCL | INCL |
| SCNiFAST8 | No | No | INCL | INCL |
| SCNiFAST16 | No | No | INCL | INCL |
| SCNiFAST32 | No | No | INCL | INCL |
| SCNiFAST64 | No | No | INCL | INCL |
| SCNiMAX | No | No | INCL | INCL |
| SCNiPTR | No | No | INCL | INCL |
| SCNo8 | No | No | INCL | INCL |
| SCNo16 | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SCNo32 | No | No | INCL | INCL |
| SCNo64 | No | No | INCL | INCL |
| SCNoLEAST8 | No | No | INCL | INCL |
| SCNoLEAST16 | No | No | INCL | INCL |
| SCNoLEAST32 | No | No | INCL | INCL |
| SCNoLEAST64 | No | No | INCL | INCL |
| SCNoFAST8 | No | No | INCL | INCL |
| SCNoFAST16 | No | No | INCL | INCL |
| SCNoFAST32 | No | No | INCL | INCL |
| SCNoFAST64 | No | No | INCL | INCL |
| SCNoMAX | No | No | INCL | INCL |
| SCNoPTR | No | No | INCL | INCL |
| SCNu8 | No | No | INCL | INCL |
| SCNu16 | No | No | INCL | INCL |
| SCNu32 | No | No | INCL | INCL |
| SCNu64 | No | No | INCL | INCL |
| SCNuLEAST8 | No | No | INCL | INCL |
| SCNuLEAST16 | No | No | INCL | INCL |
| SCNuLEAST32 | No | No | INCL | INCL |
| SCNuLEAST64 | No | No | INCL | INCL |
| SCNuFAST8 | No | No | INCL | INCL |
| SCNuFAST16 | No | No | INCL | INCL |
| SCNuFAST32 | No | No | INCL | INCL |
| SCNuFAST64 | No | No | INCL | INCL |
| SCNuMAX | No | No | INCL | INCL |
| SCNuPTR | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SCNx8 | No | No | INCL | INCL |
| SCNx16 | No | No | INCL | INCL |
| SCNx32 | No | No | INCL | INCL |
| SCNx64 | No | No | INCL | INCL |
| SCNxLEAST8 | No | No | INCL | INCL |
| SCNxLEAST16 | No | No | INCL | INCL |
| SCNxLEAST32 | No | No | INCL | INCL |
| SCNxLEAST64 | No | No | INCL | INCL |
| SCNxFAST8 | No | No | INCL | INCL |
| SCNxFAST16 | No | No | INCL | INCL |
| SCNxFAST32 | No | No | INCL | INCL |
| SCNxFAST64 | No | No | INCL | INCL |
| SCNxMAX | No | No | INCL | INCL |
| SCNxPTR | No | No | INCL | INCL |

2.3.4.20 <iso646.h>

Table 18: <iso646.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| and | No | No | No | INCL |
| and_eq | No | No | No | INCL |
| bitand | No | No | No | INCL |
| bitor | No | No | No | INCL |
| compl | No | No | No | INCL |
| not | No | No | No | INCL |
| not_eq | No | No | No | INCL |
| or | No | No | No | INCL |
| or_eq | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| xor | No | No | No | INCL |
| xor_eq | No | No | No | INCL |

2.3.4.21 <langinfo.h>

The methods in this header file are not included in any FACE Profile.

Table 19: <langinfo.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| CODESET | No | No | No | No |
| D_T_FMT | No | No | No | No |
| D_FMT | No | No | No | No |
| T_FMT | No | No | No | No |
| T_FMT_AMPM | No | No | No | No |
| AM_STR | No | No | No | No |
| PM_STR | No | No | No | No |
| DAY_1 | No | No | No | No |
| DAY_2 | No | No | No | No |
| DAY_3 | No | No | No | No |
| DAY_4 | No | No | No | No |
| DAY_5 | No | No | No | No |
| DAY_6 | No | No | No | No |
| DAY_7 | No | No | No | No |
| ABDAY_1 | No | No | No | No |
| ABDAY_2 | No | No | No | No |
| ABDAY_3 | No | No | No | No |
| ABDAY_4 | No | No | No | No |
| ABDAY_5 | No | No | No | No |
| ABDAY_6 | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| ABDAY_7 | No | No | No | No |
| MON_1 | No | No | No | No |
| MON_2 | No | No | No | No |
| MON_3 | No | No | No | No |
| MON_4 | No | No | No | No |
| MON_5 | No | No | No | No |
| MON_6 | No | No | No | No |
| MON_7 | No | No | No | No |
| MON_8 | No | No | No | No |
| MON_9 | No | No | No | No |
| MON_10 | No | No | No | No |
| MON_11 | No | No | No | No |
| MON_12 | No | No | No | No |
| ABMON_1 | No | No | No | No |
| ABMON_2 | No | No | No | No |
| ABMON_3 | No | No | No | No |
| ABMON_4 | No | No | No | No |
| ABMON_5 | No | No | No | No |
| ABMON_6 | No | No | No | No |
| ABMON_7 | No | No | No | No |
| ABMON_8 | No | No | No | No |
| ABMON_9 | No | No | No | No |
| ABMON_10 | No | No | No | No |
| ABMON_11 | No | No | No | No |
| ABMON_12 | No | No | No | No |
| ERA | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| ERA_D_FMT | No | No | No | No |
| ERA_D_T_FMT | No | No | No | No |
| ERA_T_FMT | No | No | No | No |
| ALT_DIGITS | No | No | No | No |
| RADIXCHAR | No | No | No | No |
| THOUSEP | No | No | No | No |
| YESEXPR | No | No | No | No |
| NOEXPR | No | No | No | No |
| CRNCYSTR | No | No | No | No |

2.3.4.22 <libgen.h>

This header file does not define any constants and the methods defined in it are not in any FACE Profile.

2.3.4.23 <limits.h>

Table 20: <limits.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------|-----------------|--------------------|------------------------|------------------------|
| {AIO_LISTIO_MAX} | No | No | INCL | INCL |
| {AIO_MAX} | No | No | INCL | INCL |
| {AIO_PRIO_DELTA_MAX} | No | No | INCL | INCL |
| {ARG_MAX} | No | No | INCL | INCL |
| {ATEXIT_MAX} | No | No | INCL | INCL |
| {CHILD_MAX} | No | No | INCL | INCL |
| {DELAYTIMER_MAX} | No | No | INCL | INCL |
| {HOST_NAME_MAX} | No | No | INCL | INCL |
| {IOV_MAX} | No | No | INCL | INCL |
| {LOGIN_NAME_MAX} | No | No | INCL | INCL |
| {MQ_OPEN_MAX} | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------------------|-----------------|--------------------|------------------------|------------------------|
| {MQ_PRIO_MAX} | No | No | INCL | INCL |
| {OPEN_MAX} | No | No | INCL | INCL |
| {PAGE_SIZE} | No | No | INCL | INCL |
| {PTHREAD_DESTRUCT_OR_ITERATIONS} | No | No | INCL | INCL |
| {PTHREAD_KEYS_MAX} | No | No | INCL | INCL |
| {PTHREAD_STACK_MIN} | No | No | INCL | INCL |
| {PTHREAD_THREADS_MAX} | No | No | INCL | INCL |
| {RTSIG_MAX} | No | No | INCL | INCL |
| {SEM_NSEMS_MAX} | No | No | INCL | INCL |
| {SEM_VALUE_MAX} | No | No | INCL | INCL |
| {SIGQUEUE_MAX} | No | No | INCL | INCL |
| {SS_REPL_MAX} | No | No | No | INCL |
| {STREAM_MAX} | No | No | INCL | INCL |
| {SYMLOOP_MAX} | No | No | INCL | INCL |
| {TIMER_MAX} | No | No | INCL | INCL |
| {TRACE_EVENT_NAME_MAX} | No | No | No | No |
| {TRACE_NAME_MAX} | No | No | No | No |
| {TRACE_SYS_MAX} | No | No | No | No |
| {TRACE_USER_EVENT_MAX} | No | No | No | No |
| {TTY_NAME_MAX} | No | No | INCL | INCL |
| {TZNAME_MAX} | No | No | INCL | INCL |
| {FILESIZEBITS} | No | No | No | INCL |
| {LINK_MAX} | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------------|-----------------|--------------------|------------------------|------------------------|
| {MAX_CANON} | No | No | No | INCL |
| {MAX_INPUT} | No | No | No | INCL |
| {NAME_MAX} | No | No | No | INCL |
| {PATH_MAX} | No | No | No | INCL |
| {PIPE_BUF} | No | No | No | INCL |
| {POSIX_ALLOC_SIZE_MIN} | No | No | No | INCL |
| {POSIX_REC_INCR_XFER_SIZE} | No | No | No | INCL |
| {POSIX_REC_MAX_XFER_SIZE} | No | No | No | INCL |
| {POSIX_REC_MIN_XFER_SIZE} | No | No | No | INCL |
| {POSIX_REC_XFER_ALIGN} | No | No | No | INCL |
| {SYMLINK_MAX} | No | No | No | INCL |
| {BC_BASE_MAX} | No | No | No | No |
| {BC_DIM_MAX} | No | No | No | No |
| {BC_SCALE_MAX} | No | No | No | No |
| {BC_STRING_MAX} | No | No | No | No |
| {CHARCLASS_NAME_MAX} | No | No | No | No |
| {COLL_WEIGHTS_MAX} | No | No | No | No |
| {EXPR_NEST_MAX} | No | No | No | No |
| {LINE_MAX} | No | No | INCL | INCL |
| {NGROUPS_MAX} | No | No | INCL | INCL |
| {RE_DUP_MAX} | No | No | INCL | INCL |
| {_POSIX_CLOCKRES_MIN} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|-----------------|--------------------|------------------------|------------------------|
| {_POSIX_AIO_LISTIO_MAX} | No | No | No | INCL |
| {_POSIX_AIO_MAX} | No | No | No | INCL |
| {_POSIX_ARG_MAX} | No | No | INCL | INCL |
| {_POSIX_CHILD_MAX} | No | No | INCL | INCL |
| {_POSIX_DELAYTIMER_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_HOST_NAME_MAX} | No | INCL | INCL | INCL |
| {_POSIX_LINK_MAX} | No | No | INCL | INCL |
| {_POSIX_LOGIN_NAME_MAX} | No | No | No | INCL |
| {_POSIX_MAX_CANON} | No | No | No | No |
| {_POSIX_MAX_INPUT} | No | No | No | No |
| {_POSIX_MQ_OPEN_MAX} | No | INCL | INCL | INCL |
| {_POSIX_MQ_PRIO_MAX} | No | INCL | INCL | INCL |
| {_POSIX_NAME_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_NGROUPS_MAX} | No | No | INCL | INCL |
| {_POSIX_OPEN_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_PATH_MAX} | No | INCL | INCL | INCL |
| {_POSIX_PIPE_BUF} | No | No | INCL | INCL |
| {_POSIX_RE_DUP_MAX} | No | No | No | No |
| {_POSIX_RTSIG_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_SEM_NSEMS_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_SEM_VALUE_MAX} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--|-----------------|--------------------|------------------------|------------------------|
| {_POSIX_SIGQUEUE_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_SSIZE_MAX} | No | INCL | INCL | INCL |
| {_POSIX_SS_REPL_MAX} | No | No | No | INCL |
| {_POSIX_STREAM_MAX} | No | INCL | INCL | INCL |
| {_POSIX_SYMLINK_MAX} | No | No | INCL | INCL |
| {_POSIX_SYMLOOP_MAX} | No | No | INCL | INCL |
| {_POSIX_THREAD_DEST_RUCTOR_ITERATIONS} | No | No | INCL | INCL |
| {_POSIX_THREAD_KEY_MAX} | No | INCL | INCL | INCL |
| {_POSIX_THREAD_THREADS_MAX} | No | No | INCL | INCL |
| {_POSIX_TIMER_MAX} | INCL | INCL | INCL | INCL |
| {_POSIX_TRACE_EVENT_NAME_MAX} | No | No | INCL | INCL |
| {_POSIX_TRACE_NAME_MAX} | No | No | INCL | INCL |
| {_POSIX_TRACE_SYS_MAX} | No | No | INCL | INCL |
| {_POSIX_TRACE_USER_EVENT_MAX} | No | No | INCL | INCL |
| {_POSIX_TTY_NAME_MAX} | No | No | No | No |
| {_POSIX_TZNAME_MAX} | No | INCL | INCL | INCL |
| {_POSIX2_BC_BASE_MAX} | No | No | No | No |
| {_POSIX2_BC_DIM_MAX} | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------------------|-----------------|--------------------|------------------------|------------------------|
| {_POSIX2_BC_SCALE_MAX} | No | No | No | No |
| {_POSIX2_BC_STRING_MAX} | No | No | No | No |
| {_POSIX2_CHARCLASS_NAME_MAX} | No | No | No | No |
| {_POSIX2_COLL_WEIGHTS_MAX} | No | No | No | No |
| {_POSIX2_EXPR_NEST_MAX} | No | No | No | No |
| {_POSIX2_LINE_MAX} | No | No | No | No |
| {_POSIX2_RE_DUP_MAX} | No | No | No | No |
| {_XOPEN_IOV_MAX} | No | No | No | No |
| {_XOPEN_NAME_MAX} | No | No | No | No |
| {_XOPEN_PATH_MAX} | No | No | No | No |
| {CHAR_BIT} | INCL | INCL | INCL | INCL |
| {CHAR_MAX} | INCL | INCL | INCL | INCL |
| {CHAR_MIN} | INCL | INCL | INCL | INCL |
| {INT_MAX} | INCL | INCL | INCL | INCL |
| {INT_MIN} | INCL | INCL | INCL | INCL |
| {LLONG_MAX} | INCL | INCL | INCL | INCL |
| {LLONG_MIN} | INCL | INCL | INCL | INCL |
| {LONG_BIT} | No | No | No | No |
| {LONG_MAX} | INCL | INCL | INCL | INCL |
| {LONG_MIN} | INCL | INCL | INCL | INCL |
| {MB_LEN_MAX} | No | No | No | No |
| {SCHAR_MAX} | INCL | INCL | INCL | INCL |
| {SCHAR_MIN} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| {SHRT_MAX} | INCL | INCL | INCL | INCL |
| {SHRT_MIN} | INCL | INCL | INCL | INCL |
| {SSIZE_MAX} | No | INCL | INCL | INCL |
| {UCHAR_MAX} | INCL | INCL | INCL | INCL |
| {UINT_MAX} | INCL | INCL | INCL | INCL |
| {ULLONG_MAX} | INCL | INCL | INCL | INCL |
| {ULONG_MAX} | INCL | INCL | INCL | INCL |
| {USHRT_MAX} | INCL | INCL | INCL | INCL |
| {WORD_BIT} | No | No | No | No |
| {NL_ARGMAX} | No | No | No | No |
| {NL_LANGMAX} | No | No | No | No |
| {NL_MSGMAX} | No | No | No | No |
| {NL_SETMAX} | No | No | No | No |
| {NL_TEXTMAX} | No | No | No | No |
| {NZERO} | No | No | No | No |

2.3.4.24 <locale.h>

Note: NULL is defined by multiple files. However, <locale.h> is only supported in the General Purpose Profile, and thus should not be available through <locale.h> in the Security and Safety Profiles.

Table 21: <locale.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| LC_ALL | No | No | No | INCL |
| LC_COLLATE | No | No | No | INCL |
| LC_CTYPE | No | No | No | INCL |
| LC_MESSAGES | No | No | No | INCL |
| LC_MONETARY | No | No | No | INCL |
| LC_NUMERIC | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------|-----------------|--------------------|------------------------|------------------------|
| LC_TIME | No | No | No | INCL |
| LC_ALL_MASK | No | No | No | INCL |
| LC_COLLATE_MASK | No | No | No | INCL |
| LC_CTYPE_MASK | No | No | No | INCL |
| LC_MESSAGES_MASK | No | No | No | INCL |
| LC_MONETARY_MASK | No | No | No | INCL |
| LC_NUMERIC_MASK | No | No | No | INCL |
| LC_TIME_MASK | No | No | No | INCL |
| LC_GLOBAL_LOCALE | No | No | No | INCL |
| NULL | No | No | No | INCL |

2.3.4.25 <math.h>

Note: The M_* constants are defined by the POSIX standard as part of the optional XSI support (i.e., they are not defined in C99).

Table 22: <math.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| M_E | No | No | No | INCL |
| M_LOG2E | No | No | No | INCL |
| M_LOG10E | No | No | No | INCL |
| M_LN2 | No | No | No | INCL |
| M_LN10 | No | No | No | INCL |
| M_PI | No | No | No | INCL |
| M_PI_2 | No | No | No | INCL |
| M_PI_4 | No | No | No | INCL |
| M_1_PI | No | No | No | INCL |
| M_2_PI | No | No | No | INCL |
| M_2_SQRTPI | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------|-----------------|--------------------|------------------------|------------------------|
| M_SQRT2 | No | No | No | INCL |
| M_SQRT1_2 | No | No | No | INCL |
| MAXFLOAT | No | No | No | No |
| HUGE_VAL | INCL | INCL | INCL | INCL |
| HUGE_VALF | INCL | INCL | INCL | INCL |
| HUGE_VALL | INCL | INCL | INCL | INCL |
| INFINITY | INCL | INCL | INCL | INCL |
| NAN | INCL | INCL | INCL | INCL |
| FP_INFINITE | No | No | No | INCL |
| FP_NAN | No | No | No | INCL |
| FP_NORMAL | No | No | No | INCL |
| FP_SUBNORMAL | No | No | No | INCL |
| FP_ZERO | No | No | No | INCL |
| FP_FAST_FMA | No | No | No | No |
| FP_FAST_FMAF | No | No | No | No |
| FP_FAST_FMAL | No | No | No | No |
| FP_ILOGB0 | No | No | No | INCL |
| FP_ILOGBNAN | No | No | No | INCL |
| MATH_ERRNO | No | No | No | INCL |
| MATH_ERREXCEPT | No | No | No | INCL |
| math_errhandling | No | No | No | INCL |

2.3.4.26 <monetary.h>

This header file does not define any constants and the methods in this header file are not included in any FACE Profile.

2.3.4.27 <mqueue.h>

This header file does not define any constants.

2.3.4.28 <ndbm.h>

The methods in this header file are not included in any FACE Profile.

Table 23: <ndbm.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| DBM_INSERT | No | No | No | No |
| DBM_REPLACE | No | No | No | No |

2.3.4.29 <net/if.h>

Table 24: <net/if.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| IF_NAMESIZE | No | No | No | INCL |

2.3.4.30 <netdb.h>

The EAI_* constants are errors potentially used by *getaddrinfo()* and *getnameinfo()*. Some of the constants below are commonly used but are not defined by the POSIX standard. Capabilities associated with some errors are not supported by all implementations.

Table 25: <netdb.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| AI_PASSIVE | INCL | INCL | INCL | INCL |
| AI_CANONNAME | INCL | INCL | INCL | INCL |
| AI_NUMERICHOST | INCL | INCL | INCL | INCL |
| AI_NUMERICSERV | INCL | INCL | INCL | INCL |
| AI_V4MAPPED | INCL | INCL | INCL | INCL |
| AI_ALL | INCL | INCL | INCL | INCL |
| AI_ADDRCONFIG | INCL | INCL | INCL | INCL |
| NI_NOFQDN | INCL | INCL | INCL | INCL |
| NI_NUMERICHOST | INCL | INCL | INCL | INCL |
| NI_NAMEREQD | INCL | INCL | INCL | INCL |
| NI_NUMERICSERV | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| NI_NUMERICSCOPE | INCL | INCL | INCL | INCL |
| NI_DGRAM | INCL | INCL | INCL | INCL |
| EAI_ADDRFAMILY | INCL | INCL | INCL | INCL |
| EAI AGAIN | INCL | INCL | INCL | INCL |
| EAI_BADFLAGS | INCL | INCL | INCL | INCL |
| EAI_FAIL | INCL | INCL | INCL | INCL |
| EAI_FAMILY | INCL | INCL | INCL | INCL |
| EAI_MEMORY | INCL | INCL | INCL | INCL |
| EAI_NONAME | INCL | INCL | INCL | INCL |
| EAI_SERVICE | INCL | INCL | INCL | INCL |
| EAI_SOCKTYPE | INCL | INCL | INCL | INCL |
| EAI_SYSTEM | INCL | INCL | INCL | INCL |
| EAI_BADHINTS | INCL | INCL | INCL | INCL |
| EAI_PROTOCOL | INCL | INCL | INCL | INCL |
| EAI_MAX | INCL | INCL | INCL | INCL |
| EAI_NODATA | INCL | INCL | INCL | INCL |
| EAI_OVERFLOW | INCL | INCL | INCL | INCL |
| IPPORT_RESERVED | INCL | INCL | INCL | INCL |

2.3.4.31 <netinet/in.h>

Note: The IPv6 constants are only required when IPv6 is supported.

Note: The IPPROTO_IP constant is used as the *level* argument to *getsockopt()* and *setsockopt()* for IPv4 multicast options.

Note: The IPPROTO_IPV6 constant is used as the *level* argument to *getsockopt()* and *setsockopt()* for IPv6 multicast options. These are not currently included in the FACE Technical Standard Edition 2.1 or Edition 3.0, but are IPv6 equivalents of the IPv4 constants that are included.

Table 26: <netinet/in.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|----------|-------------|-----------------|-----------------|
| IN6ADDR_ANY_INIT | INCL | INCL | INCL | INCL |
| IN6ADDR_LOOPBACK_INIT | INCL | INCL | INCL | INCL |
| IPPROTO_IP | INCL | INCL | INCL | INCL |
| IPPROTO_IPV6 | No | No | No | No |
| IPPROTO_ICMP | No | No | No | No |
| IPPROTO_RAW | No | No | No | No |
| IPPROTO_TCP | No | No | INCL | INCL |
| IPPROTO_UDP | No | No | No | No |
| INADDR_ANY | INCL | INCL | INCL | INCL |
| INADDR_BROADCAST | INCL | INCL | INCL | INCL |
| INET_ADDRSTRLEN | No | No | No | No |
| INET6_ADDRSTRLEN | No | No | No | No |
| IPv6_JOIN_GROUP | No | No | No | No |
| IPv6_LEAVE_GROUP | No | No | No | No |
| IPv6_MULTICAST_HOPS | No | No | No | No |
| IPv6_MULTICAST_IF | INCL | INCL | INCL | INCL |
| IPv6_MULTICAST_LOOP | INCL | INCL | INCL | INCL |
| IPv6_UNICAST_HOPS | No | No | No | No |
| IPv6_V6ONLY | No | No | No | No |
| IN6_IS_ADDR_UNSPECIFIED | No | No | No | No |
| IN6_IS_ADDR_LOOPBACK | No | No | No | No |
| IN6_IS_ADDR_MULTICAST | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|-----------------|--------------------|------------------------|------------------------|
| IN6_IS_ADDR_LINKLOCAL | No | No | No | No |
| IN6_IS_ADDR_SITELocal | No | No | No | No |
| IN6_IS_ADDR_V4MAPPED | No | No | No | No |
| IN6_IS_ADDR_V4COMPAT | No | No | No | No |
| IN6_IS_ADDR_MC_NODELOCAL | No | No | No | No |
| IN6_IS_ADDR_MC_LINKLOCAL | No | No | No | No |
| IN6_IS_ADDR_MC_SITELOCAL | No | No | No | No |
| IN6_IS_ADDR_MC_ORGLOCAL | No | No | No | No |
| IN6_IS_ADDR_MC_GLOBAL | No | No | No | No |

These are in the FreeBSD <netinet/in.h> file but not defined in the POSIX standard. They are required by the FACE Technical Standard.

Table 27: <netinet/in.h> Additional Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|-----------------|--------------------|------------------------|------------------------|
| IP_MULTICAST_IF | INCL | INCL | INCL | INCL |
| IP_MULTICAST_TTL | INCL | INCL | INCL | INCL |
| IP_MULTICAST_LOOP | INCL | INCL | INCL | INCL |
| IP_ADD_MEMBERSHIP | INCL | INCL | INCL | INCL |
| IP_DROP_MEMBERSHIP | INCL | INCL | INCL | INCL |
| INADDR_LOOPBACK | INCL | INCL | INCL | INCL |

2.3.4.32 <netinet/tcp.h>

Note: This header file can be used with *getsockopt()* and *setsockopt()* for IPPROTO_TCP level.

Table 28: <netinet/tcp.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| TCP_NODELAY | No | No | INCL | INCL |

2.3.4.33 *<nl_types.h>*

The methods in this header file are not in any FACE Profile.

Table 29: <nl_types.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| NL_SETD | No | No | No | No |
| NL_CAT_LOCALE | No | No | No | No |

2.3.4.34 *<poll.h>*

The methods in this header file are not in any FACE Profile.

Table 30: <poll.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------|----------|-------------|-----------------|-----------------|
| POLLIN | No | No | No | No |
| POLLRDNORM | No | No | No | No |
| POLLRDBAND | No | No | No | No |
| POLLPRI | No | No | No | No |
| POLLOUT | No | No | No | No |
| POLLWRNORM | No | No | No | No |
| POLLWRBAND | No | No | No | No |
| POLLERR | No | No | No | No |
| POLLHUP | No | No | No | No |
| POLLNVAL | No | No | No | No |

2.3.4.35 *<pthread.h>*

The PTHREAD_MUTEX_INITIALIZER will initialize a mutex to the following default attributes:

- Protocol of PTHREAD_PRIO_PROTECT

- Type of PTHREAD_MUTEX_RECURSIVE
- Process-shared of PTHREAD_PROCESS_PRIVATE

The PTHREAD_COND_INITIALIZER will initialize a conditional variable to the following default attribute:

- Process-shared of PTHREAD_PROCESS_PRIVATE

The PTHREAD_RWLOCK_INITIALIZER will initialize a reader/writer lock to the following default attribute:

- Process-shared of PTHREAD_PROCESS_PRIVATE

Note: These are also the default attributes.

Table 31: <pthread.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------------|----------|-------------|-----------------|-----------------|
| PTHREAD_BARRIER_SERIAL_THREAD | No | No | No | INCL |
| PTHREAD_CANCEL_ASYNCHRONOUS | No | No | INCL | INCL |
| PTHREAD_CANCEL_ENABLE | No | No | INCL | INCL |
| PTHREAD_CANCEL_DEFERRED | No | No | INCL | INCL |
| PTHREAD_CANCEL_DISABLE | No | No | INCL | INCL |
| PTHREAD_CANCELED | No | No | INCL | INCL |
| PTHREAD_CREATE_DETACHED | No | No | INCL | INCL |
| PTHREAD_CREATE_JOINABLE | No | No | INCL | INCL |
| PTHREAD_EXPLICIT_SHARED | INCL | INCL | INCL | INCL |
| PTHREAD_INHERIT_SHARED | INCL | INCL | INCL | INCL |
| PTHREAD_MUTEX_DEFAULT | No | No | No | INCL |
| PTHREAD_MUTEX_ERRORCHECK | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------------|-----------------|--------------------|------------------------|------------------------|
| PTHREAD_MUTEX_NORMAL | No | No | No | INCL |
| PTHREAD_MUTEX_RECURSIVE | No | No | No | INCL |
| PTHREAD_MUTEX_ROBUST | No | No | No | No |
| PTHREAD_MUTEX_STALLED | No | No | No | No |
| PTHREAD_ONCE_INIT | INCL | INCL | INCL | INCL |
| PTHREAD_PRIO_INHERIT | No | No | No | INCL |
| PTHREAD_PRIO_NONE | No | No | No | INCL |
| PTHREAD_PRIO_PROTECT | INCL | INCL | INCL | INCL |
| PTHREAD_PROCESS_SHARED | No | No | No | MP |
| PTHREAD_PROCESS_PRIVATE | No | No | No | MP |
| PTHREAD_SCOPE_PROCESS | No | No | MP | MP |
| PTHREAD_SCOPE_SYSTEM | No | No | MP | MP |
| PTHREAD_COND_INITIALIZER | No | INCL | INCL | INCL |
| PTHREAD_MUTEX_INITIALIZER | INCL | INCL | INCL | INCL |
| PTHREAD_RWLOCK_INITIALIZER | No | No | No | INCL |

2.3.4.36 <pwd.h>

This header file does not define any constants and the methods are not in any FACE Profile.

2.3.4.37 <regex.h>

The methods in this header file are not in any FACE Profile.

Table 32: <regex.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| REG_EXTENDED | No | No | No | No |
| REG_ICASE | No | No | No | No |
| REG_NOSUB | No | No | No | No |
| REG_NEWLINE | No | No | No | No |
| REG_NOTBOL | No | No | No | No |
| REG_NOTEOL | No | No | No | No |
| REG_NOMATCH | No | No | No | No |
| REG_BADPAT | No | No | No | No |
| REG_ECOLLATE | No | No | No | No |
| REG_ECTYPE | No | No | No | No |
| REG_EESCAPE | No | No | No | No |
| REG_ESUBREG | No | No | No | No |
| REG_EBRACK | No | No | No | No |
| REG_EPAREN | No | No | No | No |
| REG_EBRACE | No | No | No | No |
| REG_BADBR | No | No | No | No |
| REG_ERANGE | No | No | No | No |
| REG_ESPACE | No | No | No | No |
| REG_BADRPT | No | No | No | No |

2.3.4.38 *<sched.h>***Table 33: <sched.h> Constants**

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| SCHED_FIFO | INCL | INCL | INCL | INCL |
| SCHED_RR | INCL | INCL | INCL | INCL |
| SCHED_SPORADIC | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SCHED_OTHER | No | No | No | No |

2.3.4.39 <search.h>

The methods in this header file are not in any FACE Profile.

Table 34: <search.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FIND | No | No | No | No |
| ENTER | No | No | No | No |
| preorder | No | No | No | No |
| postorder | No | No | No | No |
| endorder | No | No | No | No |
| leaf | No | No | No | No |

2.3.4.40 <semaphore.h>

Table 35: <semaphore.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SEM_FAILED | INCL | INCL | INCL | INCL |

2.3.4.41 <setjmp.h>

This header file does not define any constants.

2.3.4.42 <signal.h>

Note: Inclusion of a SIG* constant does not mean the signal is supported in every profile.

Table 36: <signal.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SIG_DFL | INCL | INCL | INCL | INCL |
| SIG_ERR | No | No | No | INCL |
| SIG_HOLD | No | No | No | INCL |
| SIG_IGN | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SIGEV_NONE | INCL | INCL | INCL | INCL |
| SIGEV_SIGNAL | INCL | INCL | INCL | INCL |
| SIGEV_THREAD | INCL | INCL | INCL | INCL |
| SIGRTMIN | INCL | INCL | INCL | INCL |
| SIGRTMAX | INCL | INCL | INCL | INCL |
| SIGABRT | INCL | INCL | INCL | INCL |
| SIGALRM | INCL | INCL | INCL | INCL |
| SIGBUS | INCL | INCL | INCL | INCL |
| SIGCHLD | INCL | INCL | INCL | INCL |
| SIGCONT | INCL | INCL | INCL | INCL |
| SIGFPE | INCL | INCL | INCL | INCL |
| SIGHUP | INCL | INCL | INCL | INCL |
| SIGILL | INCL | INCL | INCL | INCL |
| SIGINT | INCL | INCL | INCL | INCL |
| SIGKILL | INCL | INCL | INCL | INCL |
| SIGPIPE | INCL | INCL | INCL | INCL |
| SIGQUIT | INCL | INCL | INCL | INCL |
| SIGSEGV | INCL | INCL | INCL | INCL |
| SIGSTOP | INCL | INCL | INCL | INCL |
| SIGTERM | INCL | INCL | INCL | INCL |
| SIGTSTP | INCL | INCL | INCL | INCL |
| SIGTTIN | INCL | INCL | INCL | INCL |
| SIGTTOU | INCL | INCL | INCL | INCL |
| SIGUSR1 | INCL | INCL | INCL | INCL |
| SIGUSR2 | INCL | INCL | INCL | INCL |
| SIGPOLL | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SIGPROF | INCL | INCL | INCL | INCL |
| SIGSYS | INCL | INCL | INCL | INCL |
| SIGTRAP | INCL | INCL | INCL | INCL |
| SIGURG | INCL | INCL | INCL | INCL |
| SIGVTALRM | INCL | INCL | INCL | INCL |
| SIGXCPU | INCL | INCL | INCL | INCL |
| SIGXFSZ | INCL | INCL | INCL | INCL |
| SIG_BLOCK | INCL | INCL | INCL | INCL |
| SIG_UNBLOCK | INCL | INCL | INCL | INCL |
| SIG_SETMASK | INCL | INCL | INCL | INCL |
| SA_NOCLDSTOP | No | No | MP | MP |
| SA_ONSTACK | No | No | No | No |
| SA_RESETHAND | No | No | No | No |
| SA_RESTART | No | No | INCL | INCL |
| SA_SIGINFO | INCL | INCL | INCL | INCL |
| SA_NOCLDWAIT | No | No | No | No |
| SA_NODEFER | No | No | No | No |
| SS_ONSTACK | No | No | No | No |
| SS_DISABLE | No | No | No | No |
| MINSIGSTKSZ | No | No | No | No |
| SIGSTKSZ | No | No | No | No |
| ILL_ILOPC | No | No | No | INCL |
| ILL_ILOPN | No | No | No | INCL |
| ILL_ILLADR | No | No | No | INCL |
| ILL_ILLTRP | No | No | No | INCL |
| ILL_PRVOPC | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| ILL_PRVREG | No | No | No | INCL |
| ILL_COPROC | No | No | No | INCL |
| ILL_BADSTK | No | No | No | INCL |
| FPE_INTDIV | No | No | No | INCL |
| FPE_INTOVF | No | No | No | INCL |
| FPE_FLTDIV | No | No | No | INCL |
| FPE_FLTOVF | No | No | No | INCL |
| FPE_FLTUND | No | No | No | INCL |
| FPE_FLTRES | No | No | No | INCL |
| FPE_FLTINV | No | No | No | INCL |
| FPE_FLTSUB | No | No | No | INCL |
| SEGV_MAPERR | No | No | No | INCL |
| SEGV_ACCERR | No | No | No | INCL |
| BUS_ADRALN | No | No | No | INCL |
| BUS_ADRERR | No | No | No | INCL |
| BUS_OBJERR | No | No | No | INCL |
| TRAP_BRKPT | No | No | No | No |
| TRAP_TRACE | No | No | No | No |
| CLD_EXITED | No | No | No | INCL |
| CLD_KILLED | No | No | No | INCL |
| CLD_DUMPED | No | No | No | INCL |
| CLD_TRAPPED | No | No | No | INCL |
| CLD_STOPPED | No | No | No | INCL |
| CLD_CONTINUED | No | No | No | INCL |
| POLL_IN | No | No | No | INCL |
| POLL_OUT | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| POLL_MSG | No | No | No | INCL |
| POLL_ERR | No | No | No | INCL |
| POLL_PRI | No | No | No | INCL |
| POLL_HUP | No | No | No | INCL |
| SI_USER | No | No | INCL | INCL |
| SI_QUEUE | INCL | INCL | INCL | INCL |
| SI_TIMER | INCL | INCL | INCL | INCL |
| SI_ASYNCIO | No | No | No | INCL |
| SI_MESGQ | No | No | No | INCL |

2.3.4.43 <spawn.h>

Table 37: <spawn.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------------------|-----------------|--------------------|------------------------|------------------------|
| POSIX_SPAWN_RESETIDS | No | No | MP | MP |
| POSIX_SPAWN_SETPGROUP | No | No | No | MP |
| POSIX_SPAWN_SETSIGDEF | No | No | MP | MP |
| POSIX_SPAWN_SETSIGMASK | No | No | MP | MP |
| POSIX_SPAWN_SETSCHEDPARAM | No | No | MP | MP |
| POSIX_SPAWN_SETSCHEDULER | No | No | MP | MP |

2.3.4.44 <stdarg.h>

Table 38: <stdarg.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| va_start | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| va_copy | No | No | INCL | INCL |
| va_arg | No | No | INCL | INCL |
| va_end | No | No | INCL | INCL |

2.3.4.45 *<stdbool.h>*

Table 39: <stdbool.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------------|-----------------|--------------------|------------------------|------------------------|
| bool | INCL | INCL | INCL | INCL |
| true | INCL | INCL | INCL | INCL |
| false | INCL | INCL | INCL | INCL |
| __bool_true_false_are_defined | INCL | INCL | INCL | INCL |

2.3.4.46 *<stddef.h>*

Table 40: <stddef.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------------------|-----------------|--------------------|------------------------|------------------------|
| NULL | INCL | INCL | INCL | INCL |
| offsetof(type, member-designator) | INCL | INCL | INCL | INCL |

2.3.4.47 *<stdint.h>*

Note: This header does not define methods and thus is not addressed by the FACE TS except for a requirement to provide the C99 fixed width integer types.

Note: Names with “N” in them expand to 8, 16, 32, and 64-bit variants based on the fixed-width types.

Table 41: <stdint.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| {INT8_MIN} | INCL | INCL | INCL | INCL |
| {INT16_MIN} | INCL | INCL | INCL | INCL |
| {INT32_MIN} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|-----------------|--------------------|------------------------|------------------------|
| {INT8_MAX} | INCL | INCL | INCL | INCL |
| {INT16_MAX} | INCL | INCL | INCL | INCL |
| {INT32_MAX} | INCL | INCL | INCL | INCL |
| {UINT8_MAX} | INCL | INCL | INCL | INCL |
| {UINT16_MAX} | INCL | INCL | INCL | INCL |
| {UINT32_MAX} | INCL | INCL | INCL | INCL |
| {INT_LEAST8_MIN} | INCL | INCL | INCL | INCL |
| {INT_LEAST16_MIN} | INCL | INCL | INCL | INCL |
| {INT_LEAST32_MIN} | INCL | INCL | INCL | INCL |
| {INT_LEAST8_MAX} | INCL | INCL | INCL | INCL |
| {INT_LEAST16_MAX} | INCL | INCL | INCL | INCL |
| {INT_LEAST32_MAX} | INCL | INCL | INCL | INCL |
| {UINT_LEAST8_MAX} | INCL | INCL | INCL | INCL |
| {UINT_LEAST16_MAX} | INCL | INCL | INCL | INCL |
| {UINT_LEAST32_MAX} | INCL | INCL | INCL | INCL |
| {INT_FAST8_MIN} | INCL | INCL | INCL | INCL |
| {INT_FAST16_MIN} | INCL | INCL | INCL | INCL |
| {INT_FAST32_MIN} | INCL | INCL | INCL | INCL |
| {INT_FAST8_MAX} | INCL | INCL | INCL | INCL |
| {INT_FAST16_MAX} | INCL | INCL | INCL | INCL |
| {INT_FAST32_MAX} | INCL | INCL | INCL | INCL |
| {UINT_FAST8_MAX} | INCL | INCL | INCL | INCL |
| {UINT_FAST16_MAX} | INCL | INCL | INCL | INCL |
| {UINT_FAST32_MAX} | INCL | INCL | INCL | INCL |
| {INTPTR_MIN} | INCL | INCL | INCL | INCL |
| {INTPTR_MAX} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------|-----------------|--------------------|------------------------|------------------------|
| {UINTPTR_MAX} | INCL | INCL | INCL | INCL |
| {INTMAX_MIN} | INCL | INCL | INCL | INCL |
| {INTMAX_MAX} | INCL | INCL | INCL | INCL |
| {UINTMAX_MAX} | INCL | INCL | INCL | INCL |
| {PTRDIFF_MIN} | INCL | INCL | INCL | INCL |
| {PTRDIFF_MAX} | INCL | INCL | INCL | INCL |
| {SIG_ATOMIC_MIN} | INCL | INCL | INCL | INCL |
| {SIG_ATOMIC_MAX} | INCL | INCL | INCL | INCL |
| {SIZE_MAX} | INCL | INCL | INCL | INCL |
| {WCHAR_MIN} | No | No | No | No |
| {WCHAR_MAX} | No | No | No | No |
| {WINT_MIN} | No | No | No | No |
| {WINT_MAX} | No | No | No | No |
| INT8_C() | INCL | INCL | INCL | INCL |
| INT16_C() | INCL | INCL | INCL | INCL |
| INT32_C() | INCL | INCL | INCL | INCL |
| INT64_C() | INCL | INCL | INCL | INCL |
| UINT8_C() | INCL | INCL | INCL | INCL |
| UINT16_C() | INCL | INCL | INCL | INCL |
| UINT32_C() | INCL | INCL | INCL | INCL |
| UINT64_C() | INCL | INCL | INCL | INCL |
| INTMAX_C() | INCL | INCL | INCL | INCL |
| UINTMAX_C() | INCL | INCL | INCL | INCL |

2.3.4.48 <stdio.h>

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| BUFSIZ | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| L_ctermid | No | No | No | No |
| L_tmpnam | No | No | No | No |
| _IOFBF | No | No | No | INCL |
| _IOLBF | No | No | No | INCL |
| _IONBF | No | No | No | INCL |
| SEEK_CUR | No | INCL | INCL | INCL |
| SEEK_END | No | INCL | INCL | INCL |
| SEEK_SET | No | INCL | INCL | INCL |
| {FILENAME_MAX} | No | INCL | INCL | INCL |
| {FOPEN_MAX} | No | INCL | INCL | INCL |
| {TMP_MAX} | No | No | No | INCL |
| EOF | No | INCL | INCL | INCL |
| NULL | No | INCL | INCL | INCL |
| P_tmpdir | No | No | No | No |
| stderr | No | No | No | INCL |
| stdin | No | No | No | INCL |
| stdout | No | No | No | INCL |

Table 42: <stdio.h> Constants

2.3.4.49 <stdlib.h>

Note: The W* constants are also defined in <sys/wait.h>.

Note: NULL is also defined in <stddef.h>.

Table 43: <stdlib.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| EXIT_FAILURE | No | No | INCL | INCL |
| EXIT_SUCCESS | No | No | INCL | INCL |
| {RAND_MAX} | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| {MB_CUR_MAX} | No | No | No | No |
| NULL | INCL | INCL | INCL | INCL |
| WEXITSTATUS | No | No | INCL | INCL |
| WIFEXITED | No | No | INCL | INCL |
| WIFSIGNALED | No | No | INCL | INCL |
| WIFSTOPPED | No | No | INCL | INCL |
| WNOHANG | No | No | INCL | INCL |
| WSTOPSIG | No | No | INCL | INCL |
| WTERMSIG | No | No | INCL | INCL |
| WUNTRACED | No | No | INCL | INCL |

2.3.4.50 <string.h>

Note: NULL is also defined in <stddef.h>.

Table 44: <string.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| NULL | INCL | INCL | INCL | INCL |

2.3.4.51 <strings.h>

This header file does not define any constants and the methods defined are not in any FACE Profile.

2.3.4.52 <stropts.h>

The methods defined in this header file are not in any FACE Profile.

Table 45: <stropts.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| I_ATMARK | No | No | No | No |
| I_CANPUT | No | No | No | No |
| I_CKBAND | No | No | No | No |
| I_FDINSERT | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| L_FIND | No | No | No | No |
| L_FLUSH | No | No | No | No |
| L_FLUSHBAND | No | No | No | No |
| L_GETBAND | No | No | No | No |
| L_GETCLTIME | No | No | No | No |
| L_GETSIG | No | No | No | No |
| L_GRDOPT | No | No | No | No |
| L_GWROPT | No | No | No | No |
| L_LINK | No | No | No | No |
| L_LIST | No | No | No | No |
| L_LOOK | No | No | No | No |
| L_NREAD | No | No | No | No |
| L_PEEK | No | No | No | No |
| L_PLINK | No | No | No | No |
| L_POP | No | No | No | No |
| L_PUNLINK | No | No | No | No |
| L_PUSH | No | No | No | No |
| L_RECVFD | No | No | No | No |
| L_SENDFD | No | No | No | No |
| L_SETCLTIME | No | No | No | No |
| L_SETSIG | No | No | No | No |
| L_SRDOPT | No | No | No | No |
| L_STR | No | No | No | No |
| L_SWROPT | No | No | No | No |
| L_UNLINK | No | No | No | No |
| FMNAMESZ | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FLUSHR | No | No | No | No |
| FLUSHRW | No | No | No | No |
| FLUSHW | No | No | No | No |
| S_BANDURG | No | No | No | No |
| S_ERROR | No | No | No | No |
| S_HANGUP | No | No | No | No |
| S_HIPRI | No | No | No | No |
| S_INPUT | No | No | No | No |
| S_MSG | No | No | No | No |
| S_OUTPUT | No | No | No | No |
| S_RDBAND | No | No | No | No |
| S_RDNORM | No | No | No | No |
| S_WRBAND | No | No | No | No |
| S_WRNORM | No | No | No | No |
| RS_HIPRI | No | No | No | No |
| RMSGD | No | No | No | No |
| RMSGN | No | No | No | No |
| RNORM | No | No | No | No |
| RPROTDAT | No | No | No | No |
| RPROTDIS | No | No | No | No |
| RPROTNORM | No | No | No | No |
| SNDZERO | No | No | No | No |
| ANYMARK | No | No | No | No |
| LASTMARK | No | No | No | No |
| MUXID_ALL | No | No | No | No |
| MORECTL | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| MOREDATA | No | No | No | No |
| MSG_ANY | No | No | No | No |
| MSG_BAND | No | No | No | No |
| MSG_HIPRI | No | No | No | No |

2.3.4.53 <sys/ioctl.h>

Although commonly available in operating systems that implement the POSIX standard, this header file is not part of the POSIX standard. The FACE Technical Standard requires this header file to be present and include these constants.

Table 46: <sys/ioctl.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| FIONBIO | INCL | INCL | INCL | INCL |

2.3.4.54 <sys/IPC.h>

The methods defined in this header file are not in any FACE Profile.

Table 47: <sys/IPC.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| IPC_CREAT | No | No | No | No |
| IPC_EXCL | No | No | No | No |
| IPC_NOWAIT | No | No | No | No |
| IPC_PRIVATE | No | No | No | No |
| IPC_RMID | No | No | No | No |
| IPC_SET | No | No | No | No |
| IPC_STAT | No | No | No | No |

2.3.4.55 <sys/mman.h>

Table 48: <sys/mman.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| PROT_EXEC | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------------------|-----------------|--------------------|------------------------|------------------------|
| PROT_NONE | No | No | No | INCL |
| PROT_READ | INCL | INCL | INCL | INCL |
| PROT_WRITE | INCL | INCL | INCL | INCL |
| MAP_FIXED | No | No | No | INCL |
| MAP_PRIVATE | No | No | No | INCL |
| MAP_SHARED | INCL | INCL | INCL | INCL |
| MS_ASYNC | No | No | No | INCL |
| MS_INVALIDATE | No | No | No | INCL |
| MS_SYNC | No | No | No | INCL |
| MCL_CURRENT | No | No | No | INCL |
| MCL_FUTURE | No | No | No | INCL |
| POSIX_MADV_DONTNE ED | No | No | No | No |
| POSIX_MADV_NORMAL | No | No | No | No |
| POSIX_MADV_RANDOM | No | No | No | No |
| POSIX_MADV_SEQUEN TIAL | No | No | No | No |
| POSIX_MADV_WILLNEE D | No | No | No | No |
| POSIX_TYPED_MEM_AL LOCATE | No | No | No | No |
| POSIX_TYPED_MEM_AL LOCATE_CONTIG | No | No | No | No |
| POSIX_TYPED_MEM_M AP_ALLOCATABLE | No | No | No | No |

2.3.4.56 <sys/msg.h>

Table 49: <sys/msg.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| MSG_NOERROR | No | No | No | No |

2.3.4.57 <sys/resource.h>

Table 50: <sys/resource.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| PRIO_PROCESS | No | No | No | No |
| PRIO_PGRP | No | No | No | No |
| PRIO_USER | No | No | No | No |
| RLIM_INFINITY | No | No | No | No |
| RLIM_SAVED_MAX | No | No | No | No |
| RLIM_SAVED_CUR | No | No | No | No |
| RUSAGE_SELF | No | No | No | No |
| RUSAGE_CHILDREN | No | No | No | No |
| RLIMIT_CORE | No | No | No | No |
| RLIMIT_CPU | No | No | No | No |
| RLIMIT_DATA | No | No | No | No |
| RLIMIT_FSIZE | No | No | No | No |
| RLIMIT_NOFILE | No | No | No | No |
| RLIMIT_STACK | No | No | No | No |
| RLIMIT_AS | No | No | No | No |

2.3.4.58 <sys/select.h>

Table 51: <sys/select.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------|----------|-------------|-----------------|-----------------|
| FD_SETSIZE | No | INCL | INCL | INCL |

2.3.4.59 <sys/sem.h>

Table 52: <sys/sem.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------|----------|-------------|-----------------|-----------------|
| SEM_UNDO | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| GETNCNT | No | No | No | No |
| GETPID | No | No | No | No |
| GETVAL | No | No | No | No |
| GETALL | No | No | No | No |
| GETZCNT | No | No | No | No |
| SETVAL | No | No | No | No |
| SETALL | No | No | No | No |

2.3.4.60 <sys/shm.h>

Table 53: <sys/shm.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SHM_RDONLY | No | No | No | No |
| SHM_RND | No | No | No | No |
| SHMLBA | No | No | No | No |

2.3.4.61 <sys/socket.h>

Note: The IPv6 constants are only required when IPv6 is supported.

Table 54: <sys/socket.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SCM_RIGHTS | No | No | No | No |
| SOCK_DGRAM | INCL | INCL | INCL | INCL |
| SOCK_RAW | No | No | No | INCL |
| SOCK_SEQPACKET | No | No | No | INCL |
| SOCK_STREAM | No | No | INCL | INCL |
| SOL_SOCKET | INCL | INCL | INCL | INCL |
| SO_ACCEPTCONN | No | No | No | No |
| SO_BROADCAST | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| SO_DEBUG | No | No | No | No |
| SO_DONTROUTE | No | No | No | No |
| SO_ERROR | No | No | No | No |
| SO_KEEPALIVE | No | No | No | No |
| SO_LINGER | No | No | No | No |
| SO_OOBINLINE | No | No | No | No |
| SO_RCVBUF | INCL | INCL | INCL | INCL |
| SO_RCVLOWAT | No | No | No | No |
| SO_RCVTIMEO | No | No | No | No |
| SO_REUSEADDR | INCL | INCL | INCL | INCL |
| SO_SNDBUF | INCL | INCL | INCL | INCL |
| SO SNDLOWAT | No | No | No | No |
| SO SNDTIMEO | No | No | No | No |
| SO_TYPE | No | No | No | No |
| SOMAXCONN | No | No | INCL | INCL |
| MSG_CTRUNC | No | No | No | INCL |
| MSG_DONTROUTE | No | No | No | No |
| MSG_EOR | No | No | No | No |
| MSG_OOB | No | No | No | No |
| MSG_NOSIGNAL | No | No | No | No |
| MSG_PEEK | No | No | No | No |
| MSG_TRUNC | No | No | No | INCL |
| MSG_WAITALL | No | No | No | No |
| AF_INET | INCL | INCL | INCL | INCL |
| AF_INET6 | INCL | INCL | INCL | INCL |
| AF_UNIX | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| AF_UNSPEC | No | No | No | No |
| SHUT_RD | INCL | INCL | INCL | INCL |
| SHUT_RDWR | INCL | INCL | INCL | INCL |
| SHUT_WR | INCL | INCL | INCL | INCL |

2.3.4.62 <sys/stat.h>

Note: *stat()* is in all profiles so mode_t constants should be in all. This is not for sockets (use *getsockopt()* instead).

Note: For the FACE Security Profile, only for shared memory information (read/write access permissions and length).

Table 55: <sys/stat.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| S_IFMT | No | INCL | INCL | INCL |
| S_IFBLK | No | INCL | INCL | INCL |
| S_IFCHR | No | INCL | INCL | INCL |
| S_IFIFO | No | INCL | INCL | INCL |
| S_IFREG | No | INCL | INCL | INCL |
| S_IFDIR | No | INCL | INCL | INCL |
| S_IFLNK | No | INCL | INCL | INCL |
| S_IFSOCK | No | No | No | INCL |
| S_IRWXU | No | INCL | INCL | INCL |
| S_IRUSR | INCL | INCL | INCL | INCL |
| S_IWUSR | INCL | INCL | INCL | INCL |
| S_IXUSR | No | No | INCL | INCL |
| S_IRWXG | No | INCL | INCL | INCL |
| S_IRGRP | No | INCL | INCL | INCL |
| S_IWGRP | No | INCL | INCL | INCL |
| S_IXGRP | No | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------|-----------------|--------------------|------------------------|------------------------|
| S_IRWXO | No | INCL | INCL | INCL |
| S_IROTH | No | INCL | INCL | INCL |
| S_IWOTH | No | INCL | INCL | INCL |
| S_IXOTH | No | INCL | INCL | INCL |
| S_ISUID | No | INCL | INCL | INCL |
| S_ISGID | No | INCL | INCL | INCL |
| S_ISVTX | No | INCL | INCL | INCL |
| S_ISBLK(m) | No | INCL | INCL | INCL |
| S_ISCHR(m) | No | INCL | INCL | INCL |
| S_ISDIR(m) | No | INCL | INCL | INCL |
| S_ISFIFO(m) | No | INCL | INCL | INCL |
| S_ISREG(m) | No | INCL | INCL | INCL |
| S_ISLNK(m) | No | INCL | INCL | INCL |
| S_ISSOCK(m) | No | No | No | INCL |
| S_TYPEISMQ(buf) | No | No | No | No |
| S_TYPEISSEM(buf) | No | No | No | No |
| S_TYPEISSHM(buf) | No | No | No | No |
| S_TYPEISTMO(buf) | No | No | No | No |

2.3.4.63 <sys/statvfs.h>

Table 56: <sys/statvfs.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| ST_RDONLY | No | No | No | No |
| ST_NOSUID | No | No | No | No |

2.3.4.64 <sys/time.h>

Table 57: <sys/time.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| ITIMER_REAL | No | No | No | No |
| ITIMER_VIRTUAL | No | No | No | No |
| ITIMER_PROF | No | No | No | No |

2.3.4.65 <sys/times.h>

This header file does not define any constants.

2.3.4.66 <sys/types.h>

Note: This header file does not define any constants. It does define required types.

2.3.4.67 <sys/uio.h>

This header file does not define any constants.

2.3.4.68 <sys/un.h>

This header file does not define any constants.

2.3.4.69 <sys/utsname.h>

This header file does not define any constants.

2.3.4.70 <sys/wait.h>

Table 58: <sys/wait.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| WCONTINUED | No | No | No | INCL |
| WNOHANG | No | No | INCL | INCL |
| WUNTRACED | No | No | INCL | INCL |
| WEXITSTATUS | No | No | INCL | INCL |
| WIFCONTINUED | No | No | No | INCL |
| WIFEXITED | No | No | INCL | INCL |
| WIFSIGNALLED | No | No | INCL | INCL |
| WIFSTOPPED | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| WSTOPSIG | No | No | INCL | INCL |
| WTERMSIG | No | No | INCL | INCL |
| WEXITED | No | No | No | No |
| WNOWAIT | No | No | No | No |
| WSTOPPED | No | No | No | No |

2.3.4.71 <syslog.h>

Table 59: <syslog.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| LOG_PID | No | No | No | No |
| LOG_CONS | No | No | No | No |
| LOG_NDELAY | No | No | No | No |
| LOG_ODELAY | No | No | No | No |
| LOG_NOWAIT | No | No | No | No |
| LOG_KERN | No | No | No | No |
| LOG_USER | No | No | No | No |
| LOG_MAIL | No | No | No | No |
| LOG_NEWS | No | No | No | No |
| LOG_UUCP | No | No | No | No |
| LOG_DAEMON | No | No | No | No |
| LOG_AUTH | No | No | No | No |
| LOG_CRON | No | No | No | No |
| LOG_LPR | No | No | No | No |
| LOG_LOCAL0 | No | No | No | No |
| LOG_LOCAL1 | No | No | No | No |
| LOG_LOCAL2 | No | No | No | No |
| LOG_LOCAL3 | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| LOG_LOCAL4 | No | No | No | No |
| LOG_LOCAL5 | No | No | No | No |
| LOG_LOCAL6 | No | No | No | No |
| LOG_LOCAL7 | No | No | No | No |
| LOG_MASK(pri) | No | No | No | No |
| LOG_EMERG | No | No | No | No |
| LOG_ALERT | No | No | No | No |
| LOG_CRIT | No | No | No | No |
| LOG_ERR | No | No | No | No |
| LOG_WARNING | No | No | No | No |
| LOG_NOTICE | No | No | No | No |
| LOG_INFO | No | No | No | No |
| LOG_DEBUG | No | No | No | No |

2.3.4.72 <tar.h>

Table 60: <tar.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| TMAGIC | No | No | No | No |
| TMAGLEN | No | No | No | No |
| TVERSION | No | No | No | No |
| TVERSLEN | No | No | No | No |
| REGTYPE | No | No | No | No |
| AREGTYPE | No | No | No | No |
| LNKTYPE | No | No | No | No |
| SYMTYPE | No | No | No | No |
| CHRTYPE | No | No | No | No |
| BLKTYPE | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| DIRTYPE | No | No | No | No |
| FIFOTYPE | No | No | No | No |
| CONTTYPE | No | No | No | No |
| TSUID | No | No | No | No |
| TSGID | No | No | No | No |
| TSVTX | No | No | No | No |
| TUREAD | No | No | No | No |
| TUWRITE | No | No | No | No |
| TUEXEC | No | No | No | No |
| TGREAD | No | No | No | No |
| TGWRITE | No | No | No | No |
| TGEXEC | No | No | No | No |
| TOREAD | No | No | No | No |
| TOWRITE | No | No | No | No |
| TOEXEC | No | No | No | No |

2.3.4.73 <termios.h>

Table 61: <termios.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| VEOF | No | No | No | No |
| VEOL | No | No | No | No |
| VERASE | No | No | No | No |
| VINTR | No | No | No | No |
| VKILL | No | No | No | No |
| VMIN | No | No | No | No |
| VQUIT | No | No | No | No |
| VSTART | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| VSTOP | No | No | No | No |
| VSUSP | No | No | No | No |
| VTIME | No | No | No | No |
| BRKINT | No | No | No | No |
| ICRNL | No | No | No | No |
| IGNBRK | No | No | No | No |
| IGNCR | No | No | No | No |
| IGNPAR | No | No | No | No |
| INLCR | No | No | No | No |
| INPCK | No | No | No | No |
| ISTRIP | No | No | No | No |
| IXANY | No | No | No | No |
| IXOFF | No | No | No | No |
| IXON | No | No | No | No |
| PARMRK | No | No | No | No |
| OPOST | No | No | No | No |
| ONLCR | No | No | No | No |
| OCRNL | No | No | No | No |
| ONOCR | No | No | No | No |
| ONLRET | No | No | No | No |
| OFDEL | No | No | No | No |
| OFILL | No | No | No | No |
| NLDLY | No | No | No | No |
| NL0 | No | No | No | No |
| NL1 | No | No | No | No |
| CRDLY | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| CR0 | No | No | No | No |
| CR1 | No | No | No | No |
| CR2 | No | No | No | No |
| CR3 | No | No | No | No |
| TABDLY | No | No | No | No |
| TAB0 | No | No | No | No |
| TAB1 | No | No | No | No |
| TAB2 | No | No | No | No |
| TAB3 | No | No | No | No |
| BSDLY | No | No | No | No |
| BS0 | No | No | No | No |
| BS1 | No | No | No | No |
| VTDLY | No | No | No | No |
| VT0 | No | No | No | No |
| VT1 | No | No | No | No |
| FFDLY | No | No | No | No |
| FF0 | No | No | No | No |
| FF1 | No | No | No | No |
| B0 | No | No | No | No |
| B50 | No | No | No | No |
| B75 | No | No | No | No |
| B110 | No | No | No | No |
| B134 | No | No | No | No |
| B150 | No | No | No | No |
| B200 | No | No | No | No |
| B300 | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| B600 | No | No | No | No |
| B1200 | No | No | No | No |
| B1800 | No | No | No | No |
| B2400 | No | No | No | No |
| B4800 | No | No | No | No |
| B9600 | No | No | No | No |
| B19200 | No | No | No | No |
| B38400 | No | No | No | No |
| CSIZE | No | No | No | No |
| CS5 | No | No | No | No |
| CS6 | No | No | No | No |
| CS7 | No | No | No | No |
| CS8 | No | No | No | No |
| CSTOPB | No | No | No | No |
| CREAD | No | No | No | No |
| PARENB | No | No | No | No |
| PARODD | No | No | No | No |
| HUPCL | No | No | No | No |
| CLOCAL | No | No | No | No |
| ECHO | No | No | No | No |
| ECHOE | No | No | No | No |
| ECHOK | No | No | No | No |
| ECHONL | No | No | No | No |
| ICANON | No | No | No | No |
| IEXTEN | No | No | No | No |
| ISIG | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| NOFLSH | No | No | No | No |
| TOSTOP | No | No | No | No |
| TCSANOW | No | No | No | No |
| TCSADRAIN | No | No | No | No |
| TCSAFLUSH | No | No | No | No |
| TCIFlush | No | No | No | No |
| TCIOFlush | No | No | No | No |
| TCOFlush | No | No | No | No |
| TCIOFF | No | No | No | No |
| TCION | No | No | No | No |
| TCOFF | No | No | No | No |
| TCOON | No | No | No | No |
| UL_GETFSIZE | No | No | No | No |
| UL_SETFSIZE | No | No | No | No |

2.3.4.74 <tgmath.h>

This header file and its contents are not required in any FACE Profile.

2.3.4.75 <time.h>

Note: Only CLOCK_MONOTONIC and CLOCK_REALTIME are available for use with POSIX condition variables.

Table 62: <time.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|-----------------|--------------------|------------------------|------------------------|
| CLOCKS_PER_SEC | No | No | INCL | INCL |
| CLOCK_MONOTONIC | INCL | INCL | INCL | INCL |
| CLOCK_PROCESS_CPUTIME_ID | No | No | No | MP |
| CLOCK_REALTIME | INCL | INCL | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|-----------------|--------------------|------------------------|------------------------|
| CLOCK_THREAD_CPUTIME_ID | INCL | INCL | INCL | INCL |
| TIMER_ABSTIME | INCL | INCL | INCL | INCL |

2.3.4.76 <trace.h>

Table 63: <trace.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------------|-----------------|--------------------|------------------------|------------------------|
| POSIX_TRACE_ALL_EVENTS | No | No | No | No |
| POSIX_TRACE_APPEND | No | No | No | No |
| POSIX_TRACE_CLOSE_FOR_CHILD | No | No | No | No |
| POSIX_TRACE_FILTER | No | No | No | No |
| POSIX_TRACE_FLUSH | No | No | No | No |
| POSIX_TRACE_FLUSH_START | No | No | No | No |
| POSIX_TRACE_FLUSH_STOP | No | No | No | No |
| POSIX_TRACE_FLUSHING | No | No | No | No |
| POSIX_TRACE_FULL | No | No | No | No |
| POSIX_TRACE_LOOP | No | No | No | No |
| POSIX_TRACE_NO_OVERFLOW | No | No | No | No |
| POSIX_TRACE_NOT_FLUSHING | No | No | No | No |
| POSIX_TRACE_NOT_FULL | No | No | No | No |
| POSIX_TRACE_INHERITED | No | No | No | No |
| POSIX_TRACE_NOT_TRUNCATED | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------------|-----------------|--------------------|------------------------|------------------------|
| POSIX_TRACE_OVERFLOW | No | No | No | No |
| POSIX_TRACE_OVERRUN | No | No | No | No |
| POSIX_TRACE_RESUME | No | No | No | No |
| POSIX_TRACE_RUNNING | No | No | No | No |
| POSIX_TRACE_START | No | No | No | No |
| POSIX_TRACE_STOP | No | No | No | No |
| POSIX_TRACE_SUSPENDED | No | No | No | No |
| POSIX_TRACE_SYSTEM_EVENTS | No | No | No | No |
| POSIX_TRACE_TRUNCATED_READ | No | No | No | No |
| POSIX_TRACE_TRUNCATED_RECORD | No | No | No | No |
| POSIX_TRACE_UNNAMED_USER_EVENT | No | No | No | No |
| POSIX_TRACE_UNTIL_FULL | No | No | No | No |
| POSIX_TRACE_WOPID_EVENTS | No | No | No | No |

2.3.4.77 <ulimit.h>

Table 64: <trace.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| UL_GETFSIZE | No | No | No | No |
| UL_SETFSIZE | No | No | No | No |

2.3.4.78 <unistd.h>

The _CS*, _PC*, and _SC* constants used with *sysconf()*, *fpathconf()*, *pathconf()*, and *confstr()* must be supported in order to indicate whether the feature is supported or not (-1 is returned when the feature is not supported).

Table 65: <unistd.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|----------|-------------|-----------------|-----------------|
| _POSIX_VERSION | INCL | INCL | INCL | INCL |
| _POSIX2_VERSION | INCL | INCL | INCL | INCL |
| _XOPEN_VERSION | INCL | INCL | INCL | INCL |
| _POSIX_ADVISORY_INFO | No | No | No | No |
| _POSIX_ASYNCHRONOUS_IO | No | No | No | INCL |
| _POSIX_BARRIERS | No | No | No | INCL |
| _POSIX_CHOWN_RESTRICTED | No | No | INCL | INCL |
| _POSIX_CLOCK_SELECTION | No | INCL | INCL | INCL |
| _POSIX_CPUTIME | No | No | No | INCL |
| _POSIX_FSYNC | No | INCL | INCL | INCL |
| _POSIX_IPV6 | INCL | INCL | INCL | INCL |
| _POSIX_JOB_CONTROL | No | No | INCL | INCL |
| _POSIX_MAPPED_FILES | No | No | INCL | INCL |
| _POSIX_MEMLOCK | No | No | No | INCL |
| _POSIX_MEMLOCK_RANGE | No | No | No | INCL |
| _POSIX_MEMORY_PROTECTION | No | No | No | INCL |
| _POSIX_MESSAGE_PASSING | No | INCL | INCL | INCL |
| _POSIX_MONOTONIC_CLOCK | INCL | INCL | INCL | INCL |
| _POSIX_NO_TRUNC | No | No | INCL | INCL |
| _POSIX_PRIORITIZED_IO | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---|-----------------|--------------------|------------------------|------------------------|
| _POSIX_PRIORITY_SCHEDULING | INCL | INCL | INCL | INCL |
| _POSIX_RAW_SOCKETS | No | No | No | No |
| _POSIX_READER_WRITER_LOCKS | No | No | No | INCL |
| _POSIX_REALTIME_SIGNALS | INCL | INCL | INCL | INCL |
| _POSIX_REGEXP | No | No | No | No |
| _POSIX_SAVED_IDS | No | No | INCL | INCL |
| _POSIX_SEMAPHORES | INCL | INCL | INCL | INCL |
| _POSIX_SHARED_MEMORY_OBJECTS ₂ | INCL | INCL | INCL | INCL |
| _POSIX_SHELL | No | No | No | No |
| _POSIX_SPAWN | No | No | INCL | INCL |
| _POSIX_SPIN_LOCKS | No | No | No | No |
| _POSIX_SPORADIC_SERVER | No | No | No | INCL |
| _POSIX_SYNCHRONIZE_D_IO | No | No | No | INCL |
| _POSIX_THREAD_ATTR_STACKADDR | No | No | No | No |
| _POSIX_THREAD_ATTR_STACKSIZE | No | INCL | INCL | INCL |
| _POSIX_THREAD_CPUTIME | INCL | INCL | INCL | INCL |
| _POSIX_THREAD_PRIO_INHERIT | No | No | INCL | INCL |
| _POSIX_THREAD_PRIO_PROTECT | INCL | INCL | INCL | INCL |
| _POSIX_THREAD_PRIORITY_SCHEDULING | INCL | INCL | INCL | INCL |
| _POSIX_THREAD_PROCESS_SHARED | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------------------|-----------------|--------------------|------------------------|------------------------|
| _POSIX_THREAD_ROBUST_PRIO_INHERIT | No | No | No | No |
| _POSIX_THREAD_ROBUST_PRIO_PROTECT | No | No | No | No |
| _POSIX_THREAD_SAFE_FUNCTIONS | INCL | INCL | INCL | INCL |
| _POSIX_THREAD_SPORADIC_SERVER | No | No | No | INCL |
| _POSIX_THREADS | INCL | INCL | INCL | INCL |
| _POSIX_TIMEOUTS | INCL | INCL | INCL | INCL |
| _POSIX_TIMERS | INCL | INCL | INCL | INCL |
| _POSIX_TRACE | No | No | No | No |
| _POSIX_TRACE_EVENT_FILTER | No | No | No | No |
| _POSIX_TRACE_INHERIT | No | No | No | No |
| _POSIX_TRACE_LOG | No | No | No | No |
| _POSIX_TYPED_MEMORY_OBJECTS | No | No | No | No |
| _POSIX_V6_ILP32_OFF32 ₁ | No | No | No | No |
| _POSIX_V6_ILP32_OFFBIG ₁ | No | No | No | No |
| _POSIX_V6_LP64_OFF64 ₁ | No | No | No | No |
| _POSIX_V6_LPBIG_OFFBIG ₁ | No | No | No | No |
| _POSIX_V7_ILP32_OFF32 ₁ | No | No | No | No |
| _POSIX_V7_ILP32_OFFBIG ₁ | No | No | No | No |
| _POSIX_V7_LP64_OFF64 ₁ | No | No | No | No |
| _POSIX_V7_LPBIG_OFFBIG ₁ | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|-----------------|--------------------|------------------------|------------------------|
| _POSIX2_C_BIND | No | No | No | No |
| _POSIX2_C_DEV | No | No | No | No |
| _POSIX2_CHAR_TERM | No | No | No | No |
| _POSIX2_FORT_DEV | No | No | No | No |
| _POSIX2_FORT_RUN | No | No | No | No |
| _POSIX2_LOCALEDEF | No | No | No | No |
| _POSIX2_PBS | No | No | No | No |
| _POSIX2_PBS_ACCOUNTING | No | No | No | No |
| _POSIX2_PBS_CHECKPOINT | No | No | No | No |
| _POSIX2_PBS_LOCATE | No | No | No | No |
| _POSIX2_PBS_MESSAGE | No | No | No | No |
| _POSIX2_PBS_TRACK | No | No | No | No |
| _POSIX2_SW_DEV | No | No | No | No |
| _POSIX2_UPE | No | No | No | No |
| _XOPEN_CRYPT | No | No | No | No |
| _XOPEN_ENH_I18N | No | No | No | No |
| _XOPEN_REALTIME | No | No | No | No |
| _XOPEN_REALTIME_THREADS | No | No | No | No |
| _XOPEN_SHM | No | No | No | No |
| _XOPEN_STREAMS | No | No | No | No |
| _XOPEN_UUCP | No | No | No | No |
| _POSIX_ASYNC_IO | No | No | No | INCL |
| _POSIX_PRIO_IO | No | No | No | No |
| _POSIX_SYNC_IO | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------------------------------|-----------------|--------------------|------------------------|------------------------|
| _POSIX_TIMESTAMP_RE SOLUTION | No | No | No | No |
| _POSIX2_SYMLINKS | No | No | No | No |
| F_OK | No | INCL | INCL | INCL |
| R_OK | No | INCL | INCL | INCL |
| W_OK | No | INCL | INCL | INCL |
| X_OK | No | INCL | INCL | INCL |
| _CS_PATH | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FF32_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FF32_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FF32_LIBS | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FFBIG_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FFBIG_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_ILP32_O FFBIG_LIBS | No | No | No | INCL |
| _CS_POSIX_V7_LP64_OF F64_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_LP64_OF F64_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_LP64_OF F64_LIBS | No | No | No | INCL |
| _CS_POSIX_V7_LPBIG_O FFBIG_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_LPBIG_O FFBIG_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_LPBIG_O FFBIG_LIBS | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------------------------|-----------------|--------------------|------------------------|------------------------|
| _CS_POSIX_V7_THREAD_S_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_THREAD_S_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V7_WIDTH_RESTRICTED_ENVS | No | No | No | INCL |
| _CS_V7_ENV | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFF32_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFF32_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFF32_LIBS | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_ILP32_OFFBIG_LIBS | No | No | No | INCL |
| _CS_POSIX_V6_LP64_OF_F64_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_LP64_OF_F64_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_LP64_OF_F64_LIBS | No | No | No | INCL |
| _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS | No | No | No | INCL |
| _CS_POSIX_V6_LPBIG_OFFBIG_LIBS | No | No | No | INCL |
| _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS | No | No | No | INCL |
| _CS_V6_ENV | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|------------------------|-----------------|--------------------|------------------------|------------------------|
| F_LOCK | No | No | No | No |
| F_TEST | No | No | No | No |
| F_TLOCK | No | No | No | No |
| F_ULOCK | No | No | No | No |
| _PC_2_SYMLINKS | No | No | No | INCL |
| _PC_ALLOC_SIZE_MIN | No | No | No | INCL |
| _PC_ASYNC_IO | No | No | No | INCL |
| _PC_CHOWN_RESTRICTED | No | No | No | INCL |
| _PC_FILESIZEBITS | No | No | No | INCL |
| _PC_LINK_MAX | No | No | No | INCL |
| _PC_MAX_CANON | No | No | No | INCL |
| _PC_MAX_INPUT | No | No | No | INCL |
| _PC_NAME_MAX | No | No | No | INCL |
| _PC_NO_TRUNC | No | No | No | INCL |
| _PC_PATH_MAX | No | No | No | INCL |
| _PC_PIPE_BUF | No | No | No | INCL |
| _PC_PRIO_IO | No | No | No | INCL |
| _PC_REC_INCR_XFER_SIZE | No | No | No | INCL |
| _PC_REC_MAX_XFER_SIZE | No | No | No | INCL |
| _PC_REC_MIN_XFER_SIZE | No | No | No | INCL |
| _PC_REC_XFER_ALIGN | No | No | No | INCL |
| _PC_SYMLINK_MAX | No | No | No | INCL |
| _PC_SYNC_IO | No | No | No | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|-----------------|--------------------|------------------------|------------------------|
| _PC_TIMESTAMP_RESOLUTION | No | No | No | INCL |
| _PC_VDISABLE | No | No | No | INCL |
| _SC_2_C_BIND | No | No | INCL | INCL |
| _SC_2_C_DEV | No | No | INCL | INCL |
| _SC_2_CHAR_TERM | No | No | INCL | INCL |
| _SC_2_FORT_DEV | No | No | INCL | INCL |
| _SC_2_FORT_RUN | No | No | INCL | INCL |
| _SC_2_LOCALEDEF | No | No | INCL | INCL |
| _SC_2_PBS | No | No | INCL | INCL |
| _SC_2_PBS_ACCOUNTING | No | No | INCL | INCL |
| _SC_2_PBS_CHECKPOINT | No | No | INCL | INCL |
| _SC_2_PBS_LOCATE | No | No | INCL | INCL |
| _SC_2_PBS_MESSAGE | No | No | INCL | INCL |
| _SC_2_PBS_TRACK | No | No | INCL | INCL |
| _SC_2_SW_DEV | No | No | INCL | INCL |
| _SC_2_UPE | No | No | INCL | INCL |
| _SC_2_VERSION | No | No | INCL | INCL |
| _SC_ADVISORY_INFO | No | No | INCL | INCL |
| _SC_AIO_LISTIO_MAX | No | No | INCL | INCL |
| _SC_AIO_MAX | No | No | INCL | INCL |
| _SC_AIO_PRIO_DELTA_MAX | No | No | INCL | INCL |
| _SC_ARG_MAX | No | No | INCL | INCL |
| _SC_ASYNCHRONOUS_IO | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------|-----------------|--------------------|------------------------|------------------------|
| _SC_ATEXIT_MAX | No | No | INCL | INCL |
| _SC_BARRIERS | No | No | INCL | INCL |
| _SC_BC_BASE_MAX | No | No | INCL | INCL |
| _SC_BC_DIM_MAX | No | No | INCL | INCL |
| _SC_BC_SCALE_MAX | No | No | INCL | INCL |
| _SC_BC_STRING_MAX | No | No | INCL | INCL |
| _SC_CHILD_MAX | No | No | INCL | INCL |
| _SC_CLK_TCK | No | No | INCL | INCL |
| _SC_CLOCK_SELECTION | No | No | INCL | INCL |
| _SC_COLL_WEIGHTS_MAX | No | No | INCL | INCL |
| _SC_CPUTIME | No | No | INCL | INCL |
| _SC_DELAYTIMER_MAX | No | No | INCL | INCL |
| _SC_EXPR_NEST_MAX | No | No | INCL | INCL |
| _SC_FSYNC | No | No | INCL | INCL |
| _SC_GETGR_R_SIZE_MAX | No | No | INCL | INCL |
| _SC_GETPW_R_SIZE_MAX | No | No | INCL | INCL |
| _SC_HOST_NAME_MAX | No | No | INCL | INCL |
| _SC_IOV_MAX | No | No | INCL | INCL |
| _SC_IPV6 | No | No | INCL | INCL |
| _SC_JOB_CONTROL | No | No | INCL | INCL |
| _SC_LINE_MAX | No | No | INCL | INCL |
| _SC_LOGIN_NAME_MAX | No | No | INCL | INCL |
| _SC_MAPPED_FILES | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|-----------------|--------------------|------------------------|------------------------|
| _SC_MEMLOCK | No | No | INCL | INCL |
| _SC_MEMLOCK_RANGE | No | No | INCL | INCL |
| _SC_MEMORY_PROTECTION | No | No | INCL | INCL |
| _SC_MESSAGE_PASSING | No | No | INCL | INCL |
| _SC_MONOTONIC_CLOCK | No | No | INCL | INCL |
| _SC_MQ_OPEN_MAX | No | No | INCL | INCL |
| _SC_MQ_PRIO_MAX | No | No | INCL | INCL |
| _SC_NGROUPS_MAX | No | No | INCL | INCL |
| _SC_OPEN_MAX | No | No | INCL | INCL |
| _SC_PAGE_SIZE | No | No | INCL | INCL |
| _SC_PAGESIZE | No | No | INCL | INCL |
| _SC_PRIORITIZED_IO | No | No | INCL | INCL |
| _SC_PRIORITY_SCHEDULING | No | No | INCL | INCL |
| _SC_RAW_SOCKETS | No | No | INCL | INCL |
| _SC_RE_DUP_MAX | No | No | INCL | INCL |
| _SC_READER_WRITER_LOCKS | No | No | INCL | INCL |
| _SC_REALTIME_SIGNALS | No | No | INCL | INCL |
| _SC_REGEXP | No | No | INCL | INCL |
| _SC_RTSIG_MAX | No | No | INCL | INCL |
| _SC_SAVED_IDS | No | No | INCL | INCL |
| _SC_SEM_NSEMS_MAX | No | No | INCL | INCL |
| _SC_SEM_VALUE_MAX | No | No | INCL | INCL |
| _SC_SEMAPHORES | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------------------|-----------------|--------------------|------------------------|------------------------|
| _SC_SHARED_MEMORY_OBJECTS | No | No | INCL | INCL |
| _SC_SHELL | No | No | INCL | INCL |
| _SC_SIGQUEUE_MAX | No | No | INCL | INCL |
| _SC_SPAWN | No | No | INCL | INCL |
| _SC_SPIN_LOCKS | No | No | INCL | INCL |
| _SC_SPORADIC_SERVER | No | No | INCL | INCL |
| _SC_SS_REPL_MAX | No | No | INCL | INCL |
| _SC_STREAM_MAX | No | No | INCL | INCL |
| _SC_SYMLOOP_MAX | No | No | INCL | INCL |
| _SC_SYNCHRONIZED_IO | No | No | INCL | INCL |
| _SC_THREAD_ATTR_STACKADDR | No | No | INCL | INCL |
| _SC_THREAD_ATTR_STACKSIZE | No | No | INCL | INCL |
| _SC_THREAD_CPUTIME | No | No | INCL | INCL |
| _SC_THREAD_DESTRUCTOR_ITERATIONS | No | No | INCL | INCL |
| _SC_THREAD_KEYS_MAX | No | No | INCL | INCL |
| _SC_THREAD_PRIO_INHERIT | No | No | INCL | INCL |
| _SC_THREAD_PRIO_PROTECT | No | No | INCL | INCL |
| _SC_THREAD_PRIORITY_SCHEDULING | No | No | INCL | INCL |
| _SC_THREAD_PROCESS_SHARED | No | No | INCL | INCL |
| _SC_THREAD_ROBUST_PRIO_INHERIT | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------------|-----------------|--------------------|------------------------|------------------------|
| _SC_THREAD_ROBUST_PRIO_PROTECT | No | No | INCL | INCL |
| _SC_THREAD_SAFE_FUNCTIONS | No | No | INCL | INCL |
| _SC_THREAD_SPORADIC_SERVER | No | No | INCL | INCL |
| _SC_THREAD_STACK_MIN | No | No | INCL | INCL |
| _SC_THREAD_THREADS_MAX | No | No | INCL | INCL |
| _SC_THREADS | No | No | INCL | INCL |
| _SC_TIMEOUTS | No | No | INCL | INCL |
| _SC_TIMER_MAX | No | No | INCL | INCL |
| _SC_TIMERS | No | No | INCL | INCL |
| _SC_TRACE | No | No | INCL | INCL |
| _SC_TRACE_EVENT_FILTER | No | No | INCL | INCL |
| _SC_TRACE_EVENT_NAME_MAX | No | No | INCL | INCL |
| _SC_TRACE_INHERIT | No | No | INCL | INCL |
| _SC_TRACE_LOG | No | No | INCL | INCL |
| _SC_TRACE_NAME_MAX | No | No | INCL | INCL |
| _SC_TRACE_SYS_MAX | No | No | INCL | INCL |
| _SC_TRACE_USER_EVENT_MAX | No | No | INCL | INCL |
| _SC_TTY_NAME_MAX | No | No | INCL | INCL |
| _SC_TYPED_MEMORY_OBJECTS | No | No | INCL | INCL |
| _SC_TZNAME_MAX | No | No | INCL | INCL |
| _SC_V7_ILP32_OFF32 | No | No | INCL | INCL |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------------------------|-----------------|--------------------|------------------------|------------------------|
| _SC_V7_ILP32_OFFBIG | No | No | INCL | INCL |
| _SC_V7_LP64_OFF64 | No | No | INCL | INCL |
| _SC_V7_LPBIG_OFFBIG | No | No | INCL | INCL |
| _SC_V6_ILP32_OFF32 | No | No | INCL | INCL |
| _SC_V6_ILP32_OFFBIG | No | No | INCL | INCL |
| _SC_V6_LP64_OFF64 | No | No | INCL | INCL |
| _SC_V6_LPBIG_OFFBIG | No | No | INCL | INCL |
| _SC_VERSION | No | No | INCL | INCL |
| _SC_XOPEN_CRYPT | No | No | INCL | INCL |
| _SC_XOPEN_ENH_I18N | No | No | INCL | INCL |
| _SC_XOPEN_REALTIME | No | No | INCL | INCL |
| _SC_XOPEN_REALTIME_THREADS | No | No | INCL | INCL |
| _SC_XOPEN_SHM | No | No | INCL | INCL |
| _SC_XOPEN_STREAMS | No | No | INCL | INCL |
| _SC_XOPEN_UNIX | No | No | INCL | INCL |
| _SC_XOPEN_UUCP | No | No | INCL | INCL |
| _SC_XOPEN_VERSION | No | No | INCL | INCL |
| STDERR_FILENO | No | No | No | INCL |
| STDIN_FILENO | No | No | No | INCL |
| STDOUT_FILENO | No | No | No | INCL |

Note: At most one of these can be defined.

Note: The minimum required support for the Security and Safety Base Profiles does not require the traditional shared memory object behaviors.

2.3.4.79 <utime.h>

This header file does not define any constants and the methods defined in it are not in any FACE Profile.

2.3.4.80 <utmpx.h>

Table 66: <utmpx.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| EMPTY | No | No | No | No |
| BOOT_TIME | No | No | No | No |
| OLD_TIME | No | No | No | No |
| NEW_TIME | No | No | No | No |
| USER_PROCESS | No | No | No | No |
| INIT_PROCESS | No | No | No | No |
| LOGIN_PROCESS | No | No | No | No |
| DEAD_PROCESS | No | No | No | No |

2.3.4.81 <wchar.h>

Note: WEOF is per the definition in <wctype.h>.

Table 67: <wchar.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| WCHAR_MAX | No | No | No | No |
| WCHAR_MIN | No | No | No | No |
| WEOF | No | No | No | No |

2.3.4.82 <wctype.h>

Table 68: <wctype.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|----------|----------|-------------|-----------------|-----------------|
| WEOF | No | No | No | No |

2.3.4.83 <wordexp.h>

Table 69: <wordexp.h> Constants

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| WRDE_APPEND | No | No | No | No |

| Constant | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|-----------------|--------------------|------------------------|------------------------|
| WRDE_DOOFFS | No | No | No | No |
| WRDE_NOCMD | No | No | No | No |
| WRDE_REUSE | No | No | No | No |
| WRDE_SHOWERR | No | No | No | No |
| WRDE_UNDEF | No | No | No | No |
| WRDE_BADCHAR | No | No | No | No |
| WRDE_BADVAL | No | No | No | No |
| WRDE_CMDSUB | No | No | No | No |
| WRDE_NOSPACE | No | No | No | No |
| WRDE_SYNTAX | No | No | No | No |

2.3.5 Allowed POSIX Data Types and Structures per FACE Profile

This section lists all data types defined by the POSIX standard and in which FACE Profiles they are required to be available. Some POSIX data types are defined in a primary header file but also as a side-effect of including another secondary header file. Data types and structures that are defined indirectly as a side-effect of being in a data structure are denoted by being in *italics*.

2.3.5.1 *<aio.h>*

Table 70: <aio.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------------|-----------------|--------------------|------------------------|------------------------|
| struct aiocb | No | No | No | INCL |
| <i>off_t</i> | No | No | No | INCL |
| <i>pthread_attr_t</i> | No | No | No | INCL |
| <i>size_t</i> | No | No | No | INCL |
| <i>ssize_t</i> | No | No | No | INCL |
| <i>struct timespec</i> | No | No | No | INCL |
| <i>struct sigevent</i> | No | No | No | INCL |
| <i>union sigval</i> | No | No | No | INCL |

2.3.5.2 `<arpa/inet.h>`

Table 71: <arpa/inet.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>in_port_t</i> | INCL | INCL | INCL | INCL |
| <i>in_addr_t</i> | NO | NO | NO | INCL |
| <i>in_addr</i> | NO | NO | NO | INCL |
| <i>uint32_t</i> | INCL | INCL | INCL | INCL |
| <i>uint16_t</i> | INCL | INCL | INCL | INCL |
| <i>socklen_t</i> | INCL | INCL | INCL | INCL |

2.3.5.3 `<assert.h>`

This header file does not define any data types.

2.3.5.4 `<complex.h>`

Table 72: <complex.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| complex | No | No | No | INCL |
| imaginary | No | No | No | INCL |

2.3.5.5 `<cpio.h>`

This header file does not define any data types.

2.3.5.6 `<cctype.h>`

Table 73: <cctype.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| <i>locale_t</i> | No | No | No | No |

2.3.5.7 `<dirent.h>`

Table 74: <dirent.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| DIR | No | INCL | INCL | INCL |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| struct dirent | No | No | No | No |
| <i>ino_t</i> | No | No | No | No |

Note: *ino_t* is needed for a field in *struct dirent* which is not in any FACE Profile.

2.3.5.8 <dlfcn.h>

This header file does not define any data types.

2.3.5.9 <errno.h>

This header file does not define any data types.

2.3.5.10 <fcntl.h>

Note: The *open()* and *fcntl()* APIs use this header file. The *fcntl()* API is found in the Safety Extended and General Purpose Profiles only (i.e., all of the F_* data types). For sockets, *posix_devctl()* must be used.

Note: The S_I* data types from <sys/stat.h> which are symbolic names for file modes in *mode_t* are defined by <fcntl.h> and <sys/stat.h>. See <sys/stat.h> for details. The SEEK_* data types are also defined in <stdio.h> for use with other APIs.

Table 75: <fcntl.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| struct flock | NO | NO | NO | NO |
| <i>mode_t</i> | NO | INCL | INCL | INCL |
| <i>off_t</i> | NO | NO | NO | NO |
| <i>pid_t</i> | NO | NO | NO | NO |

Note: *struct flock* is needed in support of F_RDLCK, F_WRLCK, and F_UNLCK which are not required. The types *off_t* and *pid_t* are for field in *struct flock* and thus follow it.

Note: *mode_t* is required as a parameter to *create()*.

2.3.5.11 <fenv.h>

Table 76: <fenv.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>fenv_t</i> | No | No | No | INCL |
| <i>fexcept_t</i> | No | No | No | INCL |

2.3.5.12 `<float.h>`

This header file does not define any data types.

2.3.5.13 `<fmtmsg.h>`

This header file does not define any data types.

2.3.5.14 `<fnmatch.h>`

This header file does not define any data types.

2.3.5.15 `<ftw.h>`

Table 77: <ftw.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------|----------|-------------|-----------------|-----------------|
| struct FTW | No | No | No | No |

2.3.5.16 `<glob.h>`

Table 78: <glob.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| struct glob_t | No | No | No | No |
| <i>size_t</i> | No | No | No | No |

2.3.5.17 `<grp.h>`

Table 79: <grp.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| struct group | No | No | No | No |
| <i>gid_t</i> | No | No | No | No |
| <i>pid_t</i> | No | No | No | No |

2.3.5.18 `<iconv.h>`

Table 80: <iconv.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| iconv_t | No | No | No | No |
| <i>size_t</i> | No | No | No | No |

2.3.5.19 <inttypes.h>

Table 81: <inttypes.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| imaxdiv_t | No | No | No | INCL |
| wchar_t | No | No | No | No |

Note: *imaxdiv_t* is required for the return value from *imaxdiv()*.

2.3.5.20 <iso646.h>

This header file does not define any data types.

2.3.5.21 <langinfo.h>

Table 82: <langinfo.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| locale_t | No | No | No | No |
| nl_item | No | No | No | No |

2.3.5.22 <libgen.h>

This header file does not define any data types.

2.3.5.23 <limits.h>

This header file does not define any data types.

2.3.5.24 <locale.h>

Table 83: <locale.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| struct lconv | No | No | No | INCL |
| locale_t | No | No | No | No |

2.3.5.25 <math.h>

Table 84: <math.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| float_t | No | No | No | INCL |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| double_t | No | No | No | INCL |

2.3.5.26 <*monetary.h*>

Table 85: <*monetary.h*> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| <i>locale_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>ssize_t</i> | No | No | No | No |

2.3.5.27 <*mqueue.h*>

Table 86: <*monetary.h*> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------------|----------|-------------|-----------------|-----------------|
| <i>mqd_t</i> | No | INCL | INCL | INCL |
| <i>pthread_attr_t</i> | No | No | No | No |
| <i>size_t</i> | No | INCL | INCL | INCL |
| <i>ssize_t</i> | No | INCL | INCL | INCL |
| <i>struct timespec</i> | No | INCL | INCL | INCL |
| <i>struct sigevent</i> | No | INCL | INCL | INCL |
| <i>struct mq_attr</i> | No | INCL | INCL | INCL |

2.3.5.28 <*ndbm.h*>

The methods in this header file are not included in any FACE Profile.

Table 87: <*ndbm.h*> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------|----------|-------------|-----------------|-----------------|
| <i>struct datum</i> | No | No | No | No |
| <i>ssize_t</i> | No | No | No | No |
| DBM | No | No | No | No |

2.3.5.29 <net/if.h>

Table 88: <net/if.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------|----------|-------------|-----------------|-----------------|
| struct if_nameindex | No | No | No | INCL |

2.3.5.30 <netdb.h>

Table 89: <netdb.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| in_port_t | No | No | No | No |
| in_addr_t | No | No | No | No |
| struct hostent | No | No | No | No |
| struct netent | No | No | No | INCL |
| uint32_t | No | No | No | INCL |
| struct protoent | No | No | No | INCL |
| struct servent | No | No | No | INCL |
| struct addrinfo | INCL | INCL | INCL | INCL |
| socklen_t | INCL | INCL | INCL | INCL |
| struct sockaddr | INCL | INCL | INCL | INCL |

Note: *in_port_t* and *in_addr_t* are optional and do not appear to be needed by any method.

2.3.5.31 <netinet/in.h>

Note: The IPv6 data types are only required when IPv6 is supported.

Table 90: <netinet/in.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| in_port_t | INCL | INCL | INCL | INCL |
| in_addr_t | INCL | INCL | INCL | INCL |
| sa_family_t | INCL | INCL | INCL | INCL |
| uint8_t | INCL | INCL | INCL | INCL |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------|----------|-------------|-----------------|-----------------|
| uint16_t | INCL | INCL | INCL | INCL |
| struct in_addr | INCL | INCL | INCL | INCL |
| struct sockaddr_in | INCL | INCL | INCL | INCL |
| struct in6_addr | INCL | INCL | INCL | INCL |
| struct sockaddr_in6 | INCL | INCL | INCL | INCL |
| struct ipv6_mreq | NO | NO | NO | NO |

The *struct ipv6_mreq* is used in support of IPV6_JOIN_GROUP and IPV6_LEAVE_GROUP which are not required by the FACE Technical Standard.

The *struct in6_addr* and *struct sockaddr_in6* are included in every FACE Profile but the functionality implied is conditional on IPv6 support.

Note: This header file defines no methods.

2.3.5.32 <netinet/tcp.h>

This header file does not define any data types.

2.3.5.33 <nl_types.h>

Table 91: <nl_types.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| nl_catd | No | No | No | No |
| nl_item | No | No | No | No |

2.3.5.34 <poll.h>

Table 92: <poll.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| struct pollfd | No | No | No | No |
| nfds_t | No | No | No | No |

2.3.5.35 <pthread.h>

Table 93: <pthread.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------------------|----------|-------------|-----------------|-----------------|
| <i>pthread_attr_t</i> | INCL | INCL | INCL | INCL |
| <i>pthread_barrier_t</i> | No | No | No | INCL |
| <i>pthread_barrierattr_t</i> | No | No | No | INCL |
| <i>pthread_cond_t</i> | No | INCL | INCL | INCL |
| <i>pthread_condattr_t</i> | No | INCL | INCL | INCL |
| <i>pthread_key_t</i> | No | INCL | INCL | INCL |
| <i>pthread_mutex_t</i> | No | INCL | INCL | INCL |
| <i>pthread_mutexattr_t</i> | No | INCL | INCL | INCL |
| <i>pthread_once_t</i> | INCL | INCL | INCL | INCL |
| <i>pthread_rwlock_t</i> | No | No | No | INCL |
| <i>pthread_rwlockattr_t</i> | No | No | No | INCL |
| <i>pthread_spinlock_t</i> | No | No | No | |
| <i>pthread_t</i> | INCL | INCL | INCL | INCL |

2.3.5.36 <pwd.h>

Table 94: <pwd.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| struct passwd | No | No | No | No |
| <i>gid_t</i> | No | No | No | No |
| <i>uid_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |

2.3.5.37 <regex.h>

Table 95: <regex.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------|----------|-------------|-----------------|-----------------|
| struct regex_t | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| regoff_t | No | No | No | No |
| struct regmatch_t | No | No | No | No |

2.3.5.38 <sched.h>

Table 96: <sched.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|----------|-------------|-----------------|-----------------|
| <i>pid_t</i> | No | No | INCL | INCL |
| <i>time_t</i> | No | No | INCL | INCL |
| struct timespec | No | No | INCL | INCL |
| struct sched_param | No | No | INCL | INCL |

2.3.5.39 <search.h>

Table 97: <search.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| ACTION | No | No | No | No |
| VISIT | No | No | No | No |

2.3.5.40 <semaphore.h>

Table 98: <semaphore.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| sem_t | INCL | INCL | INCL | INCL |

2.3.5.41 <setjmp.h>

Table 99: <setjmp.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------|----------|-------------|-----------------|-----------------|
| jmp_buf | No | No | No | INCL |
| sigjmp_buf | No | No | No | INCL |

2.3.5.42 <signal.h>

Table 100: <signal.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|----------|-------------|-----------------|-----------------|
| <i>pthread_t</i> | No | No | INCL | INCL |
| <i>size_t</i> | No | No | No | No |
| <i>uid_t</i> | INCL | INCL | INCL | INCL |
| <i>struct timespec</i> | INCL | INCL | INCL | INCL |
| <i>struct sigaction</i> | INCL | INCL | INCL | INCL |
| <i>sa_handler</i> | INCL | INCL | INCL | INCL |
| <i>sa_sigaction</i> | INCL | INCL | INCL | INCL |
| <i>sig_atomic_t</i> | INCL | INCL | INCL | INCL |
| <i>sigset_t</i> | INCL | INCL | INCL | INCL |
| <i>pid_t</i> | INCL | INCL | INCL | INCL |
| <i>pthread_attr_t</i> | INCL | INCL | INCL | INCL |
| <i>struct sigevent</i> | INCL | INCL | INCL | INCL |
| <i>sigval</i> | INCL | INCL | INCL | INCL |
| <i>mcontext_t</i> | No | No | No | No |
| <i>struct ucontext_t</i> | No | No | No | No |
| <i>struct stack_t</i> | No | No | No | No |
| <i>struct siginfo_t</i> | INCL | INCL | INCL | INCL |
| <i>union sigval</i> | INCL | INCL | INCL | INCL |

Note: The type *sig_atomic_t* is defined by the POSIX standard as part of the *signal()* method definition but is defined in a way that applies to it being used in any signal handler.

Note: The type *struct sigevent* is used by at least *timer_create()*.

2.3.5.43 <spawn.h>

Table 101: <spawn.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------------------|----------|-------------|-----------------|-----------------|
| <i>posix_spawnattr_t</i> | No | No | INCL | INCL |
| <i>posix_spawn_file_actions_t</i> | No | No | No | INCL |
| <i>mode_t</i> | No | No | INCL | INCL |
| <i>pid_t</i> | No | No | INCL | INCL |
| <i>sigset_t</i> | No | No | INCL | INCL |
| <i>struct sched_param</i> | No | No | INCL | INCL |

2.3.5.44 <stdarg.h>

Table 102: <stdarg.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| <i>va_list</i> | No | No | INCL | INCL |

2.3.5.45 <stdbool.h>

Table 103: <stdbool.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------|----------|-------------|-----------------|-----------------|
| <i>_Bool</i> | INCL | INCL | INCL | INCL |

2.3.5.46 <stddef.h>

Table 104: <stddef.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>ptrdiff_t</i> | INCL | INCL | INCL | INCL |
| <i>wchar_t</i> | NO | NO | NO | NO |
| <i>size_t</i> | INCL | INCL | INCL | INCL |

2.3.5.47 *<stdint.h>***Table 105: <stdint.h> Data Types**

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|-----------------|--------------------|------------------------|------------------------|
| int8_t | INCL | INCL | INCL | INCL |
| int16_t | INCL | INCL | INCL | INCL |
| int32_t | INCL | INCL | INCL | INCL |
| int64_t | INCL | INCL | INCL | INCL |
| uint8_t | INCL | INCL | INCL | INCL |
| uint16_t | INCL | INCL | INCL | INCL |
| uint32_t | INCL | INCL | INCL | INCL |
| uint64_t | INCL | INCL | INCL | INCL |
| int_least8_t | INCL | INCL | INCL | INCL |
| int_least16_t | INCL | INCL | INCL | INCL |
| int_least32_t | INCL | INCL | INCL | INCL |
| int_least64_t | INCL | INCL | INCL | INCL |
| uint_least8_t | INCL | INCL | INCL | INCL |
| uint_least16_t | INCL | INCL | INCL | INCL |
| uint_least32_t | INCL | INCL | INCL | INCL |
| uint_least64_t | INCL | INCL | INCL | INCL |
| int_fast8_t | INCL | INCL | INCL | INCL |
| int_fast16_t | INCL | INCL | INCL | INCL |
| int_fast32_t | INCL | INCL | INCL | INCL |
| int_fast64_t | INCL | INCL | INCL | INCL |
| intptr_t | INCL | INCL | INCL | INCL |
| uintptr_t | INCL | INCL | INCL | INCL |

Note: The entire list is required based on C99 independent of the POSIX standard.

2.3.5.48 `<stdio.h>`

Table 106: <stdio.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| FILE | No | INCL | INCL | INCL |
| fpos_t | No | No | No | No |
| off_t | No | No | No | No |
| <u>size_t</u> | No | No | No | INCL |
| ssize_t | No | No | No | INCL |
| va_list | No | No | No | INCL |

2.3.5.49 `<stdlib.h>`

Table 107: <stdlib.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------|----------|-------------|-----------------|-----------------|
| div_t | INCL | INCL | INCL | INCL |
| ldiv_t | INCL | INCL | INCL | INCL |
| lldiv_t | No | No | No | INCL |
| size_t | INCL | INCL | INCL | INCL |
| wchar_t | No | No | No | No |

2.3.5.50 `<string.h>`

Note: NULL is also defined in `<stddef.h>`.

Table 108: <string.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------|----------|-------------|-----------------|-----------------|
| <u>size_t</u> | INCL | INCL | INCL | INCL |
| locale_t | No | No | No | No |

2.3.5.51 `<strings.h>`

This header file does not define any data types.

2.3.5.52 <stropts.h>

Table 109: <stropts.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|----------|-------------|-----------------|-----------------|
| struct bandinfo | No | No | No | No |
| struct strpeek | No | No | No | No |
| struct strbuf | No | No | No | No |
| struct strfdinsert | No | No | No | No |
| struct strioctl | No | No | No | No |
| struct strrecvfd | No | No | No | No |
| <i>uid_t</i> | No | No | No | No |
| <i>gid_t</i> | No | No | No | No |
| t_scalar_t | No | No | No | No |
| t_uscalar_t | No | No | No | No |
| struct str_list | No | No | No | No |
| struct str_mlist | No | No | No | No |

2.3.5.53 <sys/ioctl.h>

Although commonly available in operating systems that implement the POSIX standard, this header file is not part of the POSIX standard. The FACE Technical Standard requires this header file to be present but not to define any data types.

2.3.5.54 <sys/IPC.h>

The methods defined in this header file are not in any FACE Profile.

Table 110: <sys/IPC.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| struct ipc_perm | No | No | No | No |
| <i>uid_t</i> | No | No | No | No |
| <i>gid_t</i> | No | No | No | No |
| <i>key_t</i> | No | No | No | No |

2.3.5.55 <sys/mman.h>

This header file does not define any data types.

2.3.5.56 <sys/msg.h>

Table 111: <sys/msg.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| msgqnum_t | No | No | No | No |
| msglen_t | No | No | No | No |
| struct msqid_ds | No | No | No | No |
| <i>pid_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>ssize_t</i> | No | No | No | No |
| <i>time_t</i> | No | No | No | No |

2.3.5.57 <sys/resource.h>

Table 112: <sys/resource.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| rlim_t | No | No | No | No |
| <i>id_t</i> | No | No | No | No |

2.3.5.58 <sys/select.h>

Table 113: <sys/select.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|----------|-------------|-----------------|-----------------|
| struct timeval | No | INCL | INCL | INCL |
| <i>time_t</i> | No | INCL | INCL | INCL |
| <i>suseconds_t</i> | No | INCL | INCL | INCL |
| <i>sigset_t</i> | No | INCL | INCL | INCL |
| struct timespec | No | INCL | INCL | INCL |
| <i>fd_set</i> | No | INCL | INCL | INCL |

2.3.5.59 <sys/sem.h>

Table 114: <sys/sem.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| struct semid_ds | No | No | No | No |
| <i>pid_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>time_t</i> | No | No | No | No |
| struct sembuf | No | No | No | No |

2.3.5.60 <sys/shm.h>

Table 115: <sys/shm.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------|----------|-------------|-----------------|-----------------|
| shmatt_t | No | No | No | No |
| struct shmid_ds | No | No | No | No |
| <i>pid_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>time_t</i> | No | No | No | No |

2.3.5.61 <sys/socket.h>

Note: The IPv6 data types are only required when IPv6 is supported.

Table 116: <sys/socket.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|----------|-------------|-----------------|-----------------|
| socklen_t | INCL | INCL | INCL | INCL |
| sa_family_t | INCL | INCL | INCL | INCL |
| struct sockaddr | INCL | INCL | INCL | INCL |
| struct sockaddr_storage | | | | |
| struct msghdr | INCL | INCL | INCL | INCL |
| struct iovec | INCL | INCL | INCL | INCL |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| struct cmsghdr | NO | NO | NO | NO |
| struct linger | NO | NO | NO | NO |
| <i>size_t</i> | INCL | INCL | INCL | INCL |
| <i>ssize_t</i> | INCL | INCL | INCL | INCL |

Note: The *struct sockaddr_storage* is only required when IPv6 is supported.

2.3.5.62 <sys/stat.h>

Table 117: <sys/stat.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------------|----------|-------------|-----------------|-----------------|
| struct stat | No | INCL | INCL | INCL |
| <i>blkcnt_t</i> | No | INCL | INCL | INCL |
| <i>blksize_t</i> | No | INCL | INCL | INCL |
| <i>dev_t</i> | No | INCL | INCL | INCL |
| <i>ino_t</i> | No | INCL | INCL | INCL |
| <i>mode_t</i> | No | INCL | INCL | INCL |
| <i>nlink_t</i> | No | INCL | INCL | INCL |
| <i>uid_t</i> | No | INCL | INCL | INCL |
| <i>gid_t</i> | No | INCL | INCL | INCL |
| <i>off_t</i> | No | INCL | INCL | INCL |
| <i>time_t</i> | No | INCL | INCL | INCL |
| <i>struct timespec</i> | No | INCL | INCL | INCL |

2.3.5.63 <sys/statvfs.h>

Table 118: <sys/statvfs.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| struct statvfs | No | No | No | No |
| <i>fblkcnt_t</i> | No | No | No | No |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------|----------|-------------|-----------------|-----------------|
| <i>fsfilcnt_t</i> | No | No | No | No |

2.3.5.64 <sys/time.h>

Table 119: <sys/time.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|----------|-------------|-----------------|-----------------|
| struct timeval | No | INCL | INCL | INCL |
| struct itimerval | No | No | No | No |
| <i>time_t</i> | No | INCL | INCL | INCL |
| <i>suseconds_t</i> | No | No | No | No |
| <i>fd_set</i> | No | INCL | INCL | INCL |

2.3.5.65 <sys/times.h>

Table 120: <sys/times.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| struct tms | No | No | No | No |
| <i>clock_t</i> | No | No | No | No |

2.3.5.66 <sys/types.h>

Table 121: <sys/types.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------|----------|-------------|-----------------|-----------------|
| <i>blkcnt_t</i> | No | INCL | INCL | INCL |
| <i>blksize_t</i> | No | INCL | INCL | INCL |
| <i>clock_t</i> | No | No | INCL | INCL |
| <i>clockid_t</i> | INCL | INCL | INCL | INCL |
| <i>dev_t</i> | No | INCL | INCL | INCL |
| <i>fsblkcnt_t</i> | No | No | No | No |
| <i>fsfilcnt_t</i> | No | No | No | No |
| <i>gid_t</i> | No | No | No | No |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|-----------------|--------------------|------------------------|------------------------|
| id_t | No | No | No | No |
| ino_t | No | INCL | INCL | INCL |
| key_t | No | No | No | No |
| mode_t | No | INCL | INCL | INCL |
| nlink_t | No | INCL | INCL | INCL |
| off_t | No | INCL | INCL | INCL |
| pid_t | No | No | INCL | INCL |
| pthread_attr_t | INCL | INCL | INCL | INCL |
| pthread_barrier_t | No | No | No | INCL |
| pthread_barrierattr_t | No | No | No | INCL |
| pthread_cond_t | INCL | INCL | INCL | INCL |
| pthread_condattr_t | INCL | INCL | INCL | INCL |
| pthread_key_t | No | INCL | INCL | INCL |
| pthread_mutex_t | INCL | INCL | INCL | INCL |
| pthread_mutexattr_t | INCL | INCL | INCL | INCL |
| pthread_once_t | INCL | INCL | INCL | INCL |
| pthread_rwlock_t | No | No | No | INCL |
| pthread_rwlockattr_t | No | No | No | INCL |
| pthread_spinlock_t | No | No | No | No |
| pthread_t | INCL | INCL | INCL | INCL |
| size_t | INCL | INCL | INCL | INCL |
| ssize_t | INCL | INCL | INCL | INCL |
| suseconds_t | No | No | No | No |
| time_t | INCL | INCL | INCL | INCL |
| timer_t | INCL | INCL | No | No |
| trace_attr_t | No | No | No | No |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------|----------|-------------|-----------------|-----------------|
| trace_event_id_t | No | No | No | No |
| trace_event_set_t | No | No | No | No |
| trace_id_t | No | No | No | No |
| uid_t | No | No | No | No |

Note: The type *suseconds_t* is also used by SO_SNDFTIMEO and SO_RECVTIMEO.

2.3.5.67 <sys/uio.h>

Table 122: <sys/uio.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| struct iovec | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>ssize_t</i> | No | No | No | No |

2.3.5.68 <sys/un.h>

Table 123: <sys/un.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------|----------|-------------|-----------------|-----------------|
| struct sockaddr_un | No | No | No | No |
| <i>safamily_t</i> | No | No | No | No |

2.3.5.69 <sys/utsname.h>

Table 124: <sys/un.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|----------------|----------|-------------|-----------------|-----------------|
| struct utsname | No | No | No | No |

2.3.5.70 <sys/wait.h>

Table 125: <sys/wait.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------|----------|-------------|-----------------|-----------------|
| idtype_t | No | No | No | No |
| <i>id_t</i> | No | No | No | No |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------|----------|-------------|-----------------|-----------------|
| <i>pid_t</i> | No | No | INCL | INCL |
| <i>siginfo_t</i> | No | No | No | No |
| <i>union sigval</i> | No | No | No | No |

2.3.5.71 **<syslog.h>**

This header file does not define any data types.

2.3.5.72 **<tar.h>**

This header file does not define any data types.

2.3.5.73 **<termios.h>**

Table 126: <termios.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|----------|-------------|-----------------|-----------------|
| <i>cc_t</i> | No | No | No | No |
| <i>speed_t</i> | No | No | No | No |
| <i>tcflag_t</i> | No | No | No | No |
| <i>struct termios</i> | No | No | No | No |

2.3.5.74 **<tgmath.h>**

This header file does not define any data types.

2.3.5.75 **<time.h>**

Note: Only CLOCK_MONOTONIC and CLOCK_REALTIME are available for use with POSIX condition variables.

Table 127: <time.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>clock_t</i> | No | No | No | INCL |
| <i>time_t</i> | INCL | INCL | INCL | INCL |
| <i>size_t</i> | No | No | INCL | INCL |
| <i>clockid_t</i> | INCL | INCL | INCL | INCL |
| <i>timer_t</i> | INCL | INCL | INCL | INCL |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|--------------------------|----------|-------------|-----------------|-----------------|
| <i>locale_t</i> | No | No | No | No |
| <i>pid_t</i> | No | No | No | INCL |
| <i>struct sigevent</i> | INCL | INCL | INCL | INCL |
| <i>struct tm</i> | No | INCL | INCL | INCL |
| <i>struct timespec</i> | INCL | INCL | INCL | INCL |
| <i>struct itimerspec</i> | INCL | INCL | INCL | INCL |

2.3.5.76 *<trace.h>*

Table 128: <trace.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------------------------|----------|-------------|-----------------|-----------------|
| <i>struct posix_trace_event_info</i> | No | No | No | No |
| <i>struct posix_trace_status_info</i> | No | No | No | No |

2.3.5.77 *<ulimit.h>*

This header file does not define any data types.

2.3.5.78 *<unistd.h>*

This header file does not define any data types.

2.3.5.79 *<utime.h>*

Table 129: <utime.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|----------|-------------|-----------------|-----------------|
| <i>struct utimbuf</i> | No | No | No | No |

2.3.5.80 *<utmpx.h>*

Note: This header file may include all symbols from *<sys/time.h>*.

Table 130: <utmpx.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|---------------------|----------|-------------|-----------------|-----------------|
| <i>struct utmpx</i> | No | No | No | No |

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-----------------------|----------|-------------|-----------------|-----------------|
| <i>pid_t</i> | No | No | No | No |
| <i>struct timeval</i> | No | No | No | No |

2.3.5.81 <wchar.h>

Note: WEOF is per definition in <wctype.h>.

Table 131: <wchar.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>FILE</i> | No | No | No | No |
| <i>locale_t</i> | No | No | No | No |
| <i>mbstate_t</i> | No | No | No | No |
| <i>size_t</i> | No | No | No | No |
| <i>va_list</i> | No | No | No | No |
| <i>wchar_t</i> | No | No | No | No |
| <i>wctype_t</i> | No | No | No | No |
| <i>wint_t</i> | No | No | No | No |

2.3.5.82 <wctype.h>

Table 132: <wctype.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|------------------|----------|-------------|-----------------|-----------------|
| <i>wint_t</i> | No | No | No | No |
| <i>wctrans_t</i> | No | No | No | No |
| <i>wctype_t</i> | No | No | No | No |
| <i>locale_t</i> | No | No | No | No |

2.3.5.83 <wordexp.h>

Table 133: <wordexp.h> Data Types

| Data Type | Security | Safety Base | Safety Extended | General Purpose |
|-------------------------|----------|-------------|-----------------|-----------------|
| <i>struct wordexp_t</i> | No | No | No | No |

2.3.6 Advice on POSIX API Usage

This section contains advice on using specific POSIX methods in a FACE conformant manner. This advice may explain why a method is excluded from all profiles. It may explain why it is not in a lower profile and offer guidance on how to write software without using that method. In some cases, dependence on a specific feature such as TCP or multiple processes may preclude the use of lower profiles.

2.3.6.1 `<aio.h>`

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.2 `<arpa/inet.h>`

2.3.6.2.1 `inet_addr()`

This method only supports IPv4. It is recommended to use `inet_pton()` which supports both IPv4 and IPv6.

`inet_addr()` is only in the FACE General Purpose Profile.

2.3.6.2.2 `inet_pton()`

This method supports both IPv4 and IPv6 and is the recommended alternative to `inet_addr()`.

`inet_pton()` is included in all FACE Profiles.

2.3.6.3 `<assert.h>`

2.3.6.3.1 `assert()`

`assert()` is only in the General Purpose Profile.

One alternative is simply not to use `assert()`. Another is to write a wrapper such as `my_assert()` that can invoke `assert()` in the General Purpose Profile and do nothing in the other profiles. Another approach is to define your own `my_assert()` type function and use that exclusively.

This function was made optional in the General Purpose Profile as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.4 `<complex.h>`

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.5 `<cpio.h>`

The capabilities in this header file are not required by any FACE Profile. This file defines constants related to the cpio archive format and does not define any APIs. It is not to be used in a UoC.

2.3.6.6 <ctype.h>

2.3.6.6.1 *isblank()*

This function was added to the Security, Safety Base, and Safety Extended Profiles as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *isblank()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture (SCA) standard.

2.3.6.7 <devctl.h>

The <devctl.h> header file is defined in POSIX 1003.26-2003. Since it is not in any edition of the base POSIX 1003 standard, its contents are not widely implemented outside the set of operating systems aligned with the FACE Technical Standard.

2.3.6.7.1 *posix_devctl()*

This method is only for use to close sockets and to set the non-blocking option on sockets.

Traditionally, software written to the BSD socket API has used *ioctl()* to make socket *read()* operations non-blocking. Many examples found in text books and on the Internet will use *ioctl()*. Unfortunately, the *ioctl()* call is not defined in the POSIX standard. The FACE Technical Standard requires that *posix_devctl()* supports non-blocking I/O on sockets in all profiles. The *posix_devctl()* method is not part of the base POSIX standard and not commonly supported by POSIX implementations.

The FACE Technical Standard includes *fcntl()* in the Safety Extended and General Purpose Profiles, and it is expected that software using *fcntl()* to enable non-blocking socket software should work in those profiles.

Other POSIX operating systems sometimes use *fcntl()* to support O_NONBLOCK on sockets. At least one example⁵ uses conditional compilation to choose between *ioctl()* and *fcntl()* as the supported mechanism:

```
int setNonblocking(int fd)
{
    int flags;

    /* If they have O_NONBLOCK, use the POSIX way to do it */
#ifndef O_NONBLOCK
    /* Fixme: O_NONBLOCK is defined but broken on SunOS 4.1.x and AIX
     * 3.2.5. */
    if (-1 == (flags = fcntl(fd, F_GETFL, 0)))
        flags = 0;
    return fcntl(fd, F_SETFL, flags | O_NONBLOCK);
#else
    /* Otherwise, use the old way of doing it */
    flags = 1;
    return ioctl(fd, FIOBIO, &flags);
#endif
}
```

Figure 9: Use *fcntl()* to Set Non-Blocking Mode

⁵ Example code from <http://www.kegel.com/dkftpbench/nonblocking.html>.

The CTS ensures that the application does not use the *ioctl()* call but any software using it instead of *fcntl()* will have to be modified for conformance.

2.3.6.8 *<dirent.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.9 *<dlfcn.h>*

The capabilities in this header file are not required by any FACE Profile. These were left out due to the complexity and variability of the implementation as well as tool chain dependencies. This also results in larger deployed executable images due to the necessity of including symbolic information.

2.3.6.10 *<errno.h>*

2.3.6.10.1 *errno Variable*

The variable *errno* must be provided in all profiles. It is set by many POSIX methods.

The operating system implementation of *errno* must ensure it is provided in a thread-safe manner.

2.3.6.11 *<fcntl.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.12 *<fenv.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.13 *<float.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.14 *<fmtmsg.h>*

The capabilities in this header file are not required by any FACE Profile. The only method assumes the existence of a console and is an optional POSIX capability.

2.3.6.15 *<fnmatch.h>*

The capabilities in this header file are not required by any FACE Profile. The only method in this header file is an extension and is optional in the POSIX standard.

2.3.6.16 *<ftw.h>*

The capabilities in this header file are not required by any FACE Profile. The capabilities in this header are optional POSIX capabilities.

2.3.6.17 *<glob.h>*

The capabilities in this header file are not required by any FACE Profile. The capabilities in this header file are complex and require dynamic memory allocation and deallocation. These are unlikely to be used in a safety or security-focused UoC.

2.3.6.18 `<grp.h>`

The capabilities in this header file are not required by any FACE Profile. The methods in this header file is an extension and optional in the POSIX standard.

2.3.6.19 `<iconv.h>`

The capabilities in this header file are not required by any FACE Profile. Internationalization support is not included in any FACE Profile.

2.3.6.20 `<inttypes.h>`

No clarifications are currently necessary for the capabilities defined in this header file. Internationalization support is not included in any FACE Profile.

2.3.6.21 `<iso646.h>`

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.22 `<langinfo.h>`

The capabilities in this header file are not required by any FACE Profile. Internationalization support is not included in any FACE Profile.

2.3.6.23 `<libgen.h>`

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.24 `<limits.h>`

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.25 `<locale.h>`

A subset of the methods in `<locale.h>` are supported in the General Purpose Profile. The locale can only be set to the default even in this profile. Internationalization support is not included in any FACE Profile.

2.3.6.26 `<math.h>`

2.3.6.26.1 `exp2()`

This function was added to the Security, Safety Base, and Safety Extended Profiles as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the `exp2()` method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.26.2 `log2()`

This function was added to the Security, Safety Base, and Safety Extended Profiles as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the `log2()` method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.26.3 *round()*

This function was added to the Security, Safety Base, Safety Extended Profiles as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *round()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.26.4 *trunc()*

This function was added to the Security, Safety Base, Safety Extended Profiles as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *trunc()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.27 *<monetary.h>*

The capabilities in this header file are not required by any FACE Profile. Internationalization support is not included in any FACE Profile.

2.3.6.28 *<mqueue.h>*

2.3.6.28.1 *mq_close()*

This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *mq_close()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.28.2 *mq_unlink()*

This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *mq_unlink()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.29 *<ndbm.h>*

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.30 *<net/if.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.31 *<netdb.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.32 *<netinet/in.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.33 *<netinet/tcp.h>*

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.34 `<nl_types.h>`

The capabilities in this header file are not required by any FACE Profile. Internationalization support is not included in any FACE Profile.

2.3.6.35 `<poll.h>`

The capabilities in this header file are not required by any FACE Profile. The method in this header is an extension and optional in the POSIX standard.

2.3.6.36 `<pthread.h>`

2.3.6.36.1 `pthread_mutexattr_setprotocol()`

The POSIX standard defines three protocols that may be associated with a mutex instance. Two of these are Priority Inversion Avoidance Protocols, while the third specifies that no protocol is in effect. The enumerated values for these are as follows:⁶

- PTHREAD_PRIO_INHERIT – use the Priority Inheritance Protocol
- PTHREAD_PRIO_NONE – use no Priority Inversion Avoidance Protocol
- PTHREAD_PRIO_PROTECT – use the Priority Ceiling Protocol

The FACE Technical Standard does not include all of these capabilities in the Security and Safety Profiles. The implication of this is that the use of all protocols is permitted in the General Purpose Profile. The fact that software written to the General Purpose Profile may require modifications as it is adapted to the more restrictive Security and Safety Profiles should not come as a surprise to anyone familiar with the FACE Technical Standard.

This restriction may impact software not written with FACE conformance in mind.

The CTS cannot ensure that mutexes are only created with the PTHREAD_PRIO_PROTECT in the Security and Safety Profiles. This must be done by inspection.

2.3.6.36.2 `pthread_testcancel()`

This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the `pthread_testcancel()` method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.37 `<pwd.h>`

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.38 `<regex.h>`

The capabilities in this header file are not required by any FACE Profile. The capabilities in this header file were not included due to implementation complexity and use of dynamic memory allocation and deallocation.

⁶ See <http://pubs.opengroup.org/onlinepubs/9699919799/>.

2.3.6.39 <sched.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.40 <search.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard. Additionally, the capabilities in this header file were not included due to implementation complexity and use of dynamic memory allocation and deallocation.

2.3.6.41 <semaphore.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.42 <setjmp.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.43 <signal.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.44 <spawn.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.45 <stdarg.h>

2.3.6.45.1 *va_copy()*

This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the *va_copy()* method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.46 <stdbool.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.47 <stddef.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.48 <stdint.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.49 <stdio.h>

2.3.6.49.1 *fdopen()*

This method is only included in the General Purpose Profile in the initial release of FACE Technical Standard, Edition 3.0 and all prior editions. This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. The

addition of this function increases alignment with the Software Communications Architecture standard.

Its use can be replaced by a *close()* followed by an *fopen()*.

2.3.6.49.2 *perror()*

This method prints to the stderr and is only found in the General Purpose Profile.

A common alternative is to use a custom logging routine.

2.3.6.49.3 *printf()*

This method is only available in the General Purpose Profile. See Section 2.3.1.2 for more information on stdin, stdout, and stderr.

2.3.6.49.4 *putc()*

This method is only available in the General Purpose Profile. See Section 2.3.1.2 for more information on stdin, stdout, and stderr.

2.3.6.49.5 *putchar()*

This method is only available in the General Purpose Profile. See Section 2.3.1.2 for more information on stdin, stdout, and stderr.

2.3.6.49.6 *puts()*

This method is only available in the General Purpose Profile. See Section 2.3.1.2 for more information on stdin, stdout, and stderr.

2.3.6.49.7 *setbuf()*

This method is only available in the General Purpose Profile because there is no automatic mechanism to assist in recovering the memory when the file is closed.

setbuf() can be implemented as a simple wrapper for *setvbuf()* per the POSIX standard.

2.3.6.50 <*stdlib.h*>

2.3.6.50.1 *_Exit()*

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.50.2 *atexit()*

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.50.3 `exit()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.50.4 `qsort()`

This function was added to the Safety Extended Profile as the result of a CR filed against FACE Technical Standard, Edition 3.0. Prior to this addition, the `qsort()` method was only available in the General Purpose Profile. The addition of this function increases alignment with the Software Communications Architecture standard.

2.3.6.51 `<string.h>`

2.3.6.51.1 `strcmp()`

This method should only be used on bounded strings. Otherwise, it is unsafe because it has no limit on string length and access can continue past the end of either string.

Use `strncmp()` which is included in all FACE Profiles.

2.3.6.51.2 `strcpy()`

This method should only be used on strings which are bounded (e.g., known maximum length) and ensured to be NULL terminated.

Use `strncpy()` which is in all FACE Profiles.

2.3.6.51.3 `strlen()`

This method should only be used on strings which are bounded (e.g., known maximum length) and ensured to be NULL terminated (e.g., allocate an additional character to each buffer and set it to the NULL character).

2.3.6.51.4 `strncmp()`

This is the safe alternative to `strcmp()` and is included in all FACE Profiles.

2.3.6.51.5 `strncpy()`

This is the safe alternative to `strcpy()` and is included in all FACE Profiles. Its computation is always based on the provided length.

This method is included in all FACE Profiles.

2.3.6.51.6 `strnlen()`

This method is not included in any FACE Profile.

2.3.6.52 `<strings.h>`

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.53 <stropts.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard. Streams are not included in any FACE Profile.

2.3.6.54 <sys/ipc.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard. System V IPC is not included in any FACE Profile. The POSIX IPC alternatives should be used instead.

2.3.6.55 <sys/mman.h>

2.3.6.55.1 *mmap()*

In the Security and Safety Profiles, *mmap()* is restricted for use on shared memory objects.

2.3.6.56 <sys/msg.h>

The capabilities in this header file are not required by any FACE Profile. System V IPC is not included in any FACE Profile. The POSIX IPC alternatives should be used instead.

2.3.6.57 <sys/resource.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.58 <sys/select.h>

2.3.6.58.1 *select()*

The *select()* method is only allowed to be used by the TSS and on sockets. The CTS can ensure that only TSS implementations make this call. However, it cannot ensure that the file descriptors passed to invocations of this method are always sockets. The interface for the *select()* method is defined as follows:

```
int select(
    int nfd,
    fd_set *restrict readfds,
    fd_set *restrict writefds,
    fd_set *restrict errorfds,
    struct timeval *restrict timeout);
```

Figure 10: *select()* Method Interface Definition

As can be seen from the calling sequence, the *readfds*, *writefds*, and *errorfds* arguments are lists of file descriptors. The CTS cannot inspect those three lists and thus not ensure that the *select()* method only uses file descriptors for sockets.

Further, to ensure this requirement is met would require careful visual inspection of the program source code to determine where each of the file descriptors on each of the lists was set. Each of these would have to be traced to the appropriate creation calls.

2.3.6.59 <sys/sem.h>

The capabilities in this header file are not required by any FACE Profile. System V IPC is not included in any FACE Profile. The POSIX IPC alternatives should be used instead.

2.3.6.60 <sys/shm.h>

The capabilities in this header file are not required by any FACE Profile. System V IPC is not included in any FACE Profile. The POSIX IPC alternatives should be used instead.

2.3.6.61 <sys/socket.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.62 <sys/stat.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.63 <sys/statvfs.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.64 <sys/time.h>

The capabilities in this header file are not required by any FACE Profile. The time-related methods in this header file are an extension and optional in the POSIX standard. The *select()*-related methods should be accessed via <sys/select.h>.

2.3.6.65 <sys/times.h>

2.3.6.65.1 *times()*

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.66 <sys/types.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.67 <sys/uio.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.68 <sys/un.h>

The capabilities in this header file are not required by any FACE Profile. UNIX domain sockets (AF_UNIX) are not included in any FACE Profile. AF_INET and AF_INET6 are included in all FACE Profiles.

2.3.6.69 <sys/utsname.h>

No clarifications are currently necessary for the capabilities defined in this header file.

2.3.6.70 <sys/wait.h>

2.3.6.70.1 *wait()*

This function was made optional in the General Purpose Profile as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.70.2 *waitpid()*

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.71 <syslog.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.72 <tar.h>

The capabilities in this header file are not required by any FACE Profile. This file defines constants related to the tar archive format and does not define any APIs. It is not to be used in a UoC.

2.3.6.73 <termios.h>

The capabilities in this header file are not required by any FACE Profile.

These methods may be used in IOSS UoCs to access and manipulate devices.

2.3.6.74 <tgmath.h>

The capabilities in this header file are not required by any FACE Profile. The proper math method for the data type should be directly invoked rather than using the type generic alternative.

2.3.6.75 <time.h>

2.3.6.75.1 *clock()*

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.75.2 *localtime()*

This method is considered unsafe because it is not guaranteed to be thread-safe. Use *localtime_r()* instead.

This is only included in the FACE General Purpose Profile.

2.3.6.75.3 *localtime_r()*

This is the thread-safe alternative to *localtime()*.

This method is included in the Safety Base, Safety Extended, and General Purpose Profiles.

2.3.6.75.4 `nanosleep()`

This method is included all profiles. It is a suitable alternative to `sleep()`.

2.3.6.76 `<trace.h>`

The capabilities in this header file are not required by any FACE Profile. POSIX tracing is primarily for system debugging and not supported in any FACE Profile.

2.3.6.77 `<ulimit.h>`

The capabilities in this header file are not required by any FACE Profile.

2.3.6.78 `<unistd.h>`

2.3.6.78.1 `_exit()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.2 `exec()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.3 `execle()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.4 `execv()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.5 `execve()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.6 `fork()`

The FACE Technical Standard specifies that a call to `fork()` is followed immediately by a call to `exec()`. Although this is common practice for POSIX programs, it is possible that existing software may not follow this rule. The CTS detects the use of `fork()` and `exec()` and a message is generated indicating that a visual inspection is required.

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.7 `getlogin()`

`getlogin()` is only in the General Purpose Profile in FACE Technical Standard, Edition 3.0 as published and earlier. This function was removed as the result of a CR filed against FACE Technical Standard, Edition 3.0. The inclusion of this method instead of `getlogin_r()` was a violation of the Golden Rule Reentrancy and Thread Safety discussed in Section 2.3.1.4.

Use `getlogin_r()` for thread safety.

2.3.6.78.8 `getlogin_r()`

`getlogin_r()` was added as the result of a CR filed against FACE Technical Standard, Edition 3.0. The inclusion of `getlogin()` instead of this method was a violation of the Golden Rule Reentrancy and Thread Safety discussed in Section 2.3.1.4.

2.3.6.78.9 `getpgrp()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.10 `getpid()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.11 `getppid()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.12 `setsid()`

This function was made optional in the General Purpose Profile as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

2.3.6.78.13 `sleep()`

This function was made optional in the Safety Extended and General Purpose Profiles as a consequence of the CR against FACE Technical Standard, Edition 3.0 which made support for multiple POSIX processes optional.

The function `nanosleep()` is an alternative that is in all profiles and is not a conditional feature.

2.3.6.79 `<utime.h>`

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.80 <utmpx.h>

The capabilities in this header file are not required by any FACE Profile. The methods in this header file are an extension and optional in the POSIX standard.

2.3.6.81 <wchar.h>

The capabilities in this header file are not required by any FACE Profile. Wide characters are not supported in any FACE Profile.

2.3.6.82 <wctype.h>

The capabilities in this header file are not required by any FACE Profile. Wide characters are not supported in any FACE Profile.

2.3.6.83 <wordexp.h>

The capabilities in this header file are not required by any FACE Profile. The POSIX shell is not supported in any FACE Profile.

2.4 FACE ARINC 653 Guidance

The ARINC 653 Specification has a much smaller API set than the POSIX standard and includes pseudo-code for the methods required. The combination of these two factors combined with the need for the POSIX standard to include compatibility with historic implementations greatly limits the variabilities in ARINC 653 implementations compared to implementations of the POSIX standard.

Guidance for specific ARINC 653 capabilities is provided in the relevant sections in this document.

2.5 FACE OSS and SCA AEP Conformance Guidance

Both the FACE Technical Standard, Edition 3.0 (with CRs applied) and the Software Communications Architecture (SCA), Version 4.1 specification (with Errata Version 1.0 applied) specify subsets of the POSIX standard available for use by a UoP (or application, respectively) and that an OSS (or Operating Environment (OE), respectively) must provide. The SCA's three Application Environment Profiles (AEPs) are proper subsets of the larger profiles; i.e., Ultra-Lightweight (ULw) ⊂ Lightweight (LW) ⊂ Full AEP.⁷ The SCA Full AEP is a proper subset of the FACE OSS General Purpose Profile; i.e., all the functions listed as mandatory (MAN) in the SCA Full AEP are also listed as included (INCL) in the FACE OSS General Purpose Profile and the FACE OSS General Purpose Profile has some functions that are not mandatory (MAN) in the SCA Full AEP. As a result, the SCA AEP profiles are proper subsets of the FACE OSS General Purpose Profile; i.e., Ultra-Lightweight ⊂ Lightweight ⊂ Full AEP ⊂ FACE OSS General Purpose Profile.

For UoPs that must simultaneously conform to both the SCA ULw AEP and FACE OSS Security Profile calls to message queues (*mq_open()*, *mq_receive()*, *mq_send()*, *pthread_attr_setdetachstate()*, and *pthread_mutexattr_settype()*) although mandatory (MAN) in

⁷ Where the symbol ⊂ implies “is a subset of”; e.g., A ⊂ B implies A is a subset of B.

the SCA ULw are not included (INCL) in the FACE OSS Security Profile and should be avoided in SCA application components executing on a FACE OSS Security Profile (as detailed in Table 134). Additional recommendations and rationale for UoPs that must simultaneously conform to either the SCA LW or Full AEP, and FACE OSS Security Profile are available in Table 134.

Table 134: Recommendations for SCA Applications in a FACE OSS Security Profile

| Function | Recommendation | Recommendation Rationale |
|--|---|---|
| <code>mq_open()</code> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <code>mq_receive()</code> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <code>mq_send()</code> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <code>pthread_attr_setdetachstate()</code> | Avoid using this function. ⁸ | No exiting threads or removal of resources in the FACE OSS Security or Safety Base Profiles. |
| <code>pthread_mutexattr_settype()</code> | Avoid using this function. | This function is not supported in the FACE OSS Security Profile due to it being marginally defined in the POSIX standard and PTHREAD_MUTEX_NORMAL and PTHREAD_MUTEX_DEFAULT mutex types can result in deadlock. |
| <code>calloc()</code> | Use <code>malloc()</code> and <code>memset()</code> instead of this function. | This function is not supported in the FACE OSS Security Profile. |
| <code>mktime()</code> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <code>time()</code> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <code>close()</code> | Avoid using this function. | IO not supported in the FACE OSS Security Profile. |
| <code>open()</code> | Avoid using this function. | IO not supported in the FACE OSS Security Profile. |
| <code>read()</code> | Avoid using this function. | IO not supported in the FACE OSS Security Profile. |
| <code>write()</code> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <code>pthread_cond_broadcast()</code> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |

⁸ Recommendation also applies to SCA applications in a FACE OSS Safety Base Profile.

| Function | Recommendation | Recommendation Rationale |
|-------------------------------|----------------------------|---|
| <i>pthread_cond_destroy()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_cond_init()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_cond_signal()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_cond_wait()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>mq_getattr()</i> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <i>mq_notify()</i> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <i>mq_setattr()</i> | Avoid using this function. | Message queues not supported in the FACE OSS Security Profile. |
| <i>snprintf()</i> | Avoid using this function. | Not supported in the FACE OSS Security Profile due to complexity. |
| <i>asctime_r()</i> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <i>ctime_r()</i> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <i>gmtime_r()</i> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <i>localtime_r()</i> | Avoid using this function. | Calendar feature not supported in the FACE OSS Security Profile. |
| <i>strerror_r()</i> | Avoid using this function. | IO not supported in the FACE OSS Security Profile. |
| <i>clearerr()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fclose()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>feof()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>ferror()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |

| Function | Recommendation | Recommendation Rationale |
|------------------|----------------------------|---|
| <i>fflush()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fgetc()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fgets()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fileno()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fopen()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fprintf()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fread()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>freopen()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fwrite()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>select()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fseek()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fseeko()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>ftell()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>ftello()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>lseek()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>access()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>chdir()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |

| Function | Recommendation | Recommendation Rationale |
|-----------------------------------|----------------------------|---|
| <i>closedir()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>creat()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>fstat()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>getcwd()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>link()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>mkdir()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>opendir()</i> | Avoid using this function. | File system not supported in the FACE OSS Security Profile. |
| <i>readdir()</i> | Avoid using this function | File system not supported in the FACE OSS Security Profile. |
| <i>pthread_cond_timedwait()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_condattr_destroy()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_condattr_init()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_getspecific()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_key_create()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |
| <i>pthread_setspecific()</i> | Avoid using this function. | Conditional variables not supported in the FACE OSS Security Profile. |

In addition, for UoPs that must simultaneously conform to both the SCA LW AEP and FACE OSS Safety Base Profile, calls to the following should be avoided in SCA application components executing on a FACE OSS Safety Base Profile: *bsearch()*, *free()*, *qsort()*, *realloc()*, *setlocale()*, *sscanf()*, and *strftime()*. Calls to the following should instead use their respective “_r()” functions: *gmtime()*, *localtime()*, *rand()*, *srand()*, and *strtok()*. Rationale for these recommendations is available in Table 135. Additional recommendations and rationale for UoPs that must simultaneously conform to both the SCA Full AEP and FACE OSS Security Profile are available in Table 135.

Table 135: Recommendations for SCA Applications in a FACE OSS Security Profile

| Function | Recommendation | Recommendation Rationale |
|-----------------------|---|--|
| <i>bsearch()</i> | Avoid using this function. | Function is not safety certified and is not supported in the FACE OSS Safety Base Profile. |
| <i>free()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>realloc()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>sscanf()</i> | Avoid using this function. | Buffer overwrite protection not supported in the FACE OSS Safety Base Profile. |
| <i>strftime()</i> | Avoid using this function. | Buffer overwrite protection not supported in the FACE OSS Safety Base Profile. |
| <i>gmtime()</i> | Use <i>gmtime_r()</i> instead of this function. | This function is not thread-safe and suggested alternate is. |
| <i>localtime()</i> | Use <i>localtime_r()</i> ⁹ instead of this function. | This function is not thread-safe and suggested alternate is. |
| <i>rand()</i> | Use <i>rand_r()</i> ⁹ instead of this function. | This function is not thread-safe and suggested alternate is. |
| <i>srand()</i> | Use <i>srand_r()</i> ⁹ instead of this function. | This function is not thread-safe and suggested alternate is. |
| <i>strtok()</i> | Use <i>strtok_r()</i> ⁹ instead of this function. | This function is not thread-safe and suggested alternate is. |
| <i>sem_destroy()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>sem_unlink()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>timer_delete()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>va_arg()</i> | Avoid using this function. | Variable argument sizes or lists not supported in the FACE OSS Safety Base Profile. |

⁹ Recommendation also applies to SCA applications in a FACE OSS Safety Extended Profile.

| Function | Recommendation | Recommendation Rationale |
|--|----------------------------|---|
| <code>va_end()</code> | Avoid using this function. | Variable argument sizes or lists not supported in the FACE OSS Safety Base Profile. |
| <code>va_start()</code> | Avoid using this function. | Variable argument sizes or lists not supported in the FACE OSS Safety Base Profile. |
| <code>vsnprintf()</code> | Avoid using this function. | Variable argument sizes or lists not supported in the FACE OSS Safety Base Profile. |
| <code>accept()</code> | Avoid using this function. | TCP not supported in the FACE OSS Safety Base Profile. |
| <code>listen()</code> | Avoid using this function. | TCP not supported in the FACE OSS Safety Base Profile. |
| <code>abort()</code> | Avoid using this function. | Signals not supported in the FACE OSS Safety Base Profile. |
| <code>kill()</code> | Avoid using this function. | Signals not supported in the FACE OSS Safety Base Profile. |
| <code>raise()</code> | Avoid using this function. | Signals not supported in the FACE OSS Safety Base Profile. |
| <code>pthread_attr_getdetachstate()</code> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <code>pthread_cancel()</code> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) and random change in execution not supported in the FACE OSS Safety Base Profile. |
| <code>pthread_cleanup_pop()</code> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) and random change in execution not supported in the FACE OSS Safety Base Profile. |
| <code>pthread_cleanup_push()</code> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) and random change in execution not supported in the FACE OSS Safety Base Profile. |
| <code>pthread_detach()</code> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |

| Function | Recommendation | Recommendation Rationale |
|---------------------------------|---|---|
| <i>pthread_exit()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>pthread_join()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>pthread_key_delete()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>pthread_kill()</i> | Use <i>sigqueue()</i> instead of this function. | This function is not supported in the FACE OSS Safety Base Profile . |
| <i>pthread_mutex_destroy()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) in the FACE OSS Safety Base Profile. |
| <i>pthread_setcancelstate()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) and random change in execution not supported in the FACE OSS Safety Base Profile. |
| <i>pthread_setcanceltype()</i> | Avoid using this function. | Removal of resources not supported (e.g., close, destroy, unlink, or exiting threads) and random change in execution not supported in the FACE OSS Safety Base Profile. |

In addition, for UoPs that must conform to both the SCA Full AEP and FACE Safety Extended Profile calls, the following should be avoided in SCA application components executing on a FACE OSS Safety Extended Profile: *fpathconf()*, *getc()*, *getchar()*, *pathconf()*, *perror()*, *printf()*, *pthread_mutexattr_gettype()*, *putc()*, *putchar()*, *setbuf()*, *setvbuf()*, *strcoll()*, *strxfrm()*, and *ungetc()*. Calls to *strerror()* should instead use *strerror_r()*. Calls to *fputc()* and *fputs()* should instead use *fprintf()*. Calls to *rewind()* should instead use *fseek()*. Calls to *sigprocmask()* should instead use *pthread_sigmask()*. Calls to *signal()* should instead use *sigaction()*. Calls to *fscanf()* should instead use *sscanf()*. Rationale for these recommendations is available in Table 136.

Table 136: Recommendations for SCA Applications in a FACE OSS Security Profile

| Function | Recommendation | Recommendation Rationale |
|-------------------|---|---|
| <i>strcoll()</i> | Avoid using this function. | Locale not supported in the FACE OSS Safety Extended Profile. |
| <i>strerror()</i> | Use <i>strerror_r()</i> instead of this function. | This function is not thread-safe and suggested alternate is. |

| Function | Recommendation | Recommendation Rationale |
|--------------------|--|---|
| <i>strxfrm()</i> | Avoid using this function. | Locale not supported in the FACE OSS Safety Extended Profile. |
| <i>fputc()</i> | Use <i>fprintf()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>fputs()</i> | Use <i>fprintf()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>fscanf()</i> | Use <i>sscanf()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>getc()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>getchar()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>perror()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>printf()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>putc()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>putchar()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>setbuf()</i> | Avoid using this function. | This function is not supported in the FACE OSS Safety Extended Profile due to safety issue (i.e., no automatic notification that it needs to be recovered when the string is closed). |
| <i>setvbuf()</i> | Avoid using this function. | This function is not supported in the FACE OSS Safety Extended Profile due to safety issue (i.e., no automatic notification that it needs to be recovered when the string is closed). |
| <i>ungetc()</i> | Avoid using this function. | STDIN, OUT, and ERROR not supported in the FACE OSS Safety Extended Profile. |
| <i>rewind()</i> | Use <i>fseek()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>fpathconf()</i> | Avoid using this function. | Function not supported in the FACE OSS Safety Extended Profile due to complexity. |

| Function | Recommendation | Recommendation Rationale |
|------------------------------------|--|---|
| <i>pathconf()</i> | Avoid using this function. | Function not supported in the FACE OSS Safety Extended Profile due to complexity. |
| <i>signal()</i> | Use <i>sigaction()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>sigprocmask()</i> | Use <i>pthread_sigmask()</i> instead of this function. | This function is not supported in the FACE Safety Extended Profile. |
| <i>pthread_mutexattr_gettype()</i> | Avoid using this function. | Function not supported in the FACE OSS Safety Extended Profile due to it being optional in the POSIX standard and marginally defined by the POSIX standard. |

For UoPs that must simultaneously conform to either the SCA ULw, LW, or Full AEP, and the FACE General Purpose Profile there are no recommendations to note since all the functions listed as mandatory (MAN) in the SCA AEPs are included (INCL) in the FACE General Purpose Profile (assuming errata and CRs are applied to each).

3 Health Monitoring/Fault Management (HMF M) Guidance

3.1 Introduction

Health Monitoring/Fault Management (HMF M) is responsible for monitoring and reporting application faults and failures with the operating system, application software, and hardware within a computing module. Faults may occur at the system, computing platform, partition, application level, or with I/O devices the computing platform is directly responsible for controlling. Fault detection, propagation, reporting, and management require careful attention to detail and coordination between multiple layers within a system to operate correctly. Not all of these layers are defined by the FACE Technical Standard.

3.1.1 Summary of Changes from FACE Technical Standard, Edition 2.1

The following changes to the OSS have occurred since FACE Technical Standard, Edition 2.1:

- Clarification that the FACE HMF M Services are to be used in POSIX operational environments and that the ARINC 653 Health Services are to be used in ARINC 653 operational environments

3.1.2 How to Read this Section

The information contained in this portion of the Reference Implementation Guide should be used when developing software that implements the FACE HMF M Services, uses FACE HMF M Services, or uses ARINC 653 Health Services.

3.1.3 FACE HMF M Services versus ARINC 653 Health Services

The FACE Technical Standard includes both the FACE HMF M Services and the ARINC 653 Health Services. Applications in the ARINC 653 operational environment are to use the ARINC 653 Health Services. Applications using the POSIX operational environment are to use the FACE HMF M Services API.

The FACE HMF M Services API are based on the ARINC 653 Health Services and intended to be functionality equivalent as much as possible. If the operating system supports ARINC 653 Health Services, the FACE HMF M Services can be implemented by directly utilizing the underlying ARINC 653 Health Services. ARINC 653 Health Services use the terms “fault” and “process” where the FACE HMF M Services use “error” and “thread”, respectively.

Given the design intent that the FACE HMF M Services reflect the ARINC 653 Health Services, the content of this section largely applies to both.

3.1.4 HMFM Areas Addressed by the FACE Technical Standard

The following areas of responsibility are addressed by the FACE Technical Standard:

- Computing platform faults such as power failure potentially impact the execution of every application on the computing platform and thus may be reported to the application via the FACE HMFMs or the ARINC 653 Health Services
Computing platform faults may also occur during initialization or when the system is incorrectly configured. Thus, only some potential computing platform faults will be reported via the FACE HMFMs. Those not reported via the FACE HMFMs API are specific to the operating system and it is the responsibility of the System Integrator to address those with the given operating system mechanisms.
- Application-level faults may be detected by either the operating system or the application itself

Operating system-detected faults include illegal operating system requests and process execution faults (e.g., memory access errors or numeric exceptions). The application may self-detect faults and report them via the HMFMs API.

3.1.5 HMFM Areas Beyond the FACE Technical Standard

The following areas of responsibility are not addressed by the FACE Technical Standard:

- I/O device faults may be detected by device drivers, I/O Services, communications protocol stacks, or applications
If redundant I/O devices are available, there may be the option to switch to a backup device. If no redundancy, and after multiple reset attempts, the only alternative may be to turn the device off and operate with reduced capability. The means to detect these faults and manage redundant I/O devices is beyond the scope of the FACE Technical Standard.
- In both partitioned and non-partitioned systems, there may be faults in the OS itself
These faults will be managed by the OS and an OS may provide capabilities for the System Integrator to extend the actions taken in response to an OS fault.
- In partitioned environments, partition-level faults may also occur from an incorrect configuration or prior to application execution
None of these faults are reported via the FACE HMFMs API. These are also the responsibility of System Integrator to address given the operating system-provided mechanisms.
- System-level faults are beyond the scope of the FACE Technical Standard

3.2 Role of Board Support Packages

In all operating system environments, the microprocessor and bus interface controllers may detect erroneous actions by software. These erroneous actions may originate in the operating system, application, or hardware interface software. Board Support Packages (BSPs) and device drivers are key to the detection and reporting of many fault types. These hardware-detected faults, along with the operating system or application-detected faults, are reported to the operating system's HMFMs capability. The operating system's HMFMs capability is utilized by

the System Integrator and application developer to provide a well-defined set of responses to the set of potential fault conditions. Figure 11 illustrates this architecturally:

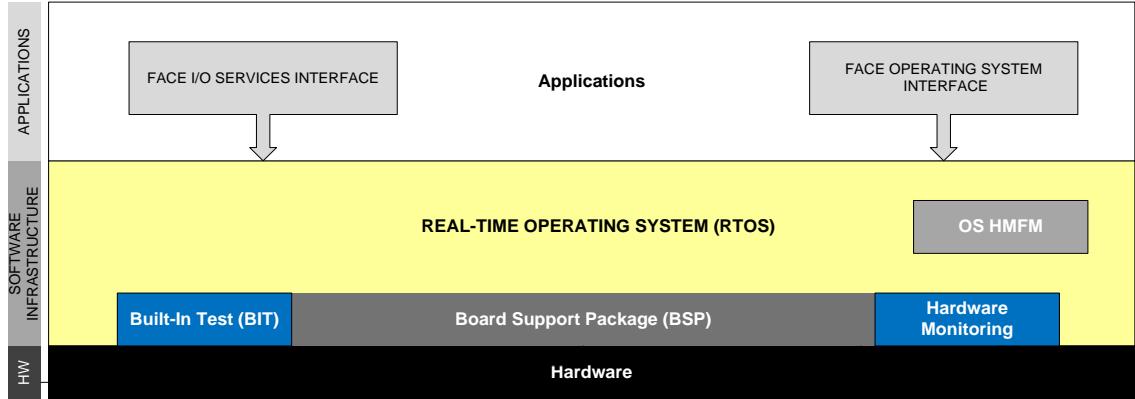


Figure 11: HMFM Board Support Package and Real-Time Operating System Relationship

At the most fundamental level, the BSP is responsible for the exception handlers that catch faults such as floating point errors and memory access violations. The BSP may use standard operating system-provided implementations of these processor-dependent exceptions. If the BSP is improperly installed, the fault handling mechanisms defined by any framework will never be used. Similarly, if device drivers do not detect and report faults in the devices they control, the faults will never enter the health framework.

The BSP may also provide additional capabilities, such as Built-In Test (BIT). BIT may execute continuously in the background or be executed on-demand. BIT executed on-demand may require the associated hardware devices be taken offline and may negatively impact operational readiness. The design of on-demand BIT may have to take into account safety considerations.

Some hardware configurations include sensors used to monitor the health-related environmental factors. Examples include thermal sensors, humidity sensors, fan speed, and indicators that the hardware case is open. The BSP may provide interfaces so the sensors can report faults.

3.3 HMFM Layering

HMFM requires a multi-layered view of the system in regard to fault detection, handling, reporting, reaction, and analysis. This view extends from addressing operating system-initiated faults to application-specific faults to entire computing platform failures.

Figure 12 illustrates the breadth of HMFM in a system.

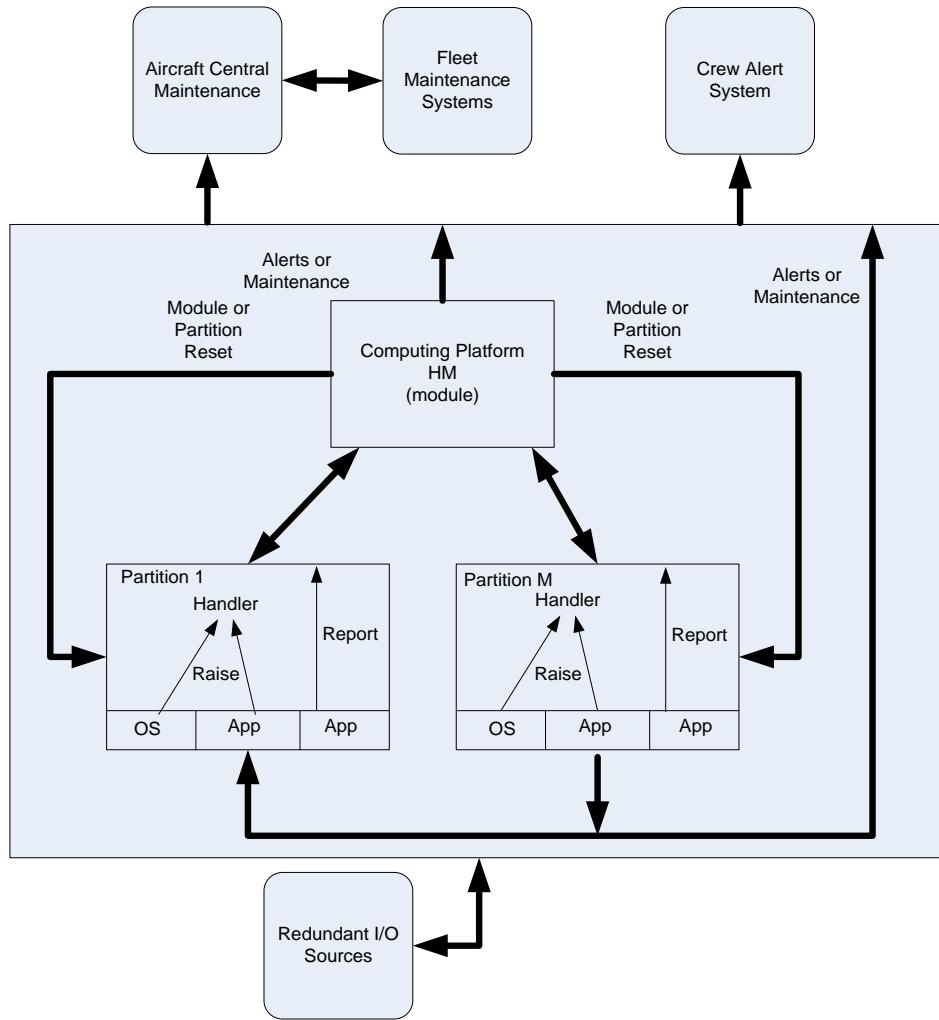


Figure 12: HMFM Layering

3.3.1 Hardware-Detected Faults

At the lowest level is the hardware. Hardware-detected application faults, such as floating point faults, divide by zero, and illegal memory accesses, can be reported to the application. ARINC 653 Health Services and POSIX signals may be used by an application to be informed of hardware faults.

The hardware may be the source of faults, but this is considered beyond the current scope of FACE HMFM. The exception is when the microprocessor detects certain erroneous application behavior and reports it via the FACE HMFM capabilities.

3.3.2 Application-Detected Faults

The application itself may detect faults and report them. These faults are processed using the same strategies as hardware-detected application faults.

The application's fault handler may report the fault to a Computing Platform Health Manager if it is unable to process the fault.

3.3.3 Computing Platform Health Manager

The Computing Platform Health Manager may be provided by the operating system or the System Integrator. It is aware of the application requirements and will perform whatever corrective or remedial action is indicated by the fault. This may involve:

- Resetting a partition
- Sending fault information to entities outside the computing platform
- Resetting the computing platform

3.3.4 External Systems

From a systems perspective, the computing platform used by a FACE Computing Environment is a single component in a complex avionics system. Some faults may be completely managed within the computing platform and need not be reported further. However, if the fault is propagated or needs to be reported outside the computing platform, then it could go to external systems such as a Crew Alert System or Aircraft Central Maintenance log. The definition of these external interfaces is beyond the scope of the FACE Technical Standard.

It is important to consider the intended audience when propagating or reporting faults outside the computing platform. For example, a Pilot or other Crew Member is likely to need a different level of detail than a Developer or Maintainer.

3.4 Operating System Configurations

The FACE Technical Standard specifies three operating system configurations and the native HMFM capabilities available:

- Pure ARINC 653 OS
- POSIX on ARINC 653 OS
- Pure POSIX OS

3.4.1 ARINC 653 OS

In an operating system based upon ARINC 653, there will be configuration tables which are used to define the desired action to take in response to specific faults (errors in ARINC 653 terminology).

An application can have an error handler process. A partition-level error mechanism is executed if an application error handler does not exist, or the handler has an error. The OS Partition Health Monitor Table specifies the action to take upon various faults at the partition level. If an error occurs at the module level, the Module Health Monitor Table specifies the system action to take based upon system mode and fault.

3.4.2 Pure ARINC 653 OS

In the pure ARINC 653 OS configuration, an application is written strictly to the ARINC 653 API. In this case, the ARINC 653 Health Services are available. From an HMFM Services perspective, this component would be FACE conformant.

3.4.3 POSIX on ARINC 653 OS

In the POSIX on ARINC 653 OS configuration, the ARINC 653 OS supports applications written to the POSIX standard. The ARINC 653 Health Services are available to the application even though they are not part of the POSIX API. If the application were evaluated as being POSIX standard-conformant, it would fail due to the use of ARINC 653 APIs. From an HMFM Services perspective, this component would be FACE conformant.

3.4.4 POSIX OS

For the purposes of this discussion, a pure POSIX OS is one with no underlying ARINC 653 services. Examples in this category include GNU/Linux, LynxOS®, and Solaris™.

There are no POSIX services for an interoperable Health Services framework. Some of the ARINC 653 fault events are similar to those defined to generate POSIX signals (e.g., SIGFPE, SIGBUS, SIGSEGV). Some faults can be detected and reported, but there are no standard Health Services in the POSIX standard to define what happens after the user signal handler is invoked.

Programming languages with built-in exception processing, such as Ada and Java®, may install handlers for the POSIX signals. For example, the Ada programming language run-time may install a signal handler for SIGFPE and translate that into raising a NUMERIC_ERROR exception as defined by the Ada standard. In these cases, the fault will likely be handled within the context of the application and not involve the HMFM Services.

3.4.5 HMFM Services per OS Configuration

The HMFM Services API provides a common HMFM API and framework available for POSIX operational environments. It is designed to not add excessive overhead or deviate from the accepted practice of using ARINC 653 Health Services.

The HMFM Services API is part of the OSS and provides compatible behavior across all of the operating system configurations. It is not an ARINC 653-compatible solution in the POSIX standard. Table 137 illustrates the feasibility of providing the FACE HMFM Services API and ARINC 653 Health Service in the various FACE OS configurations. Note that the FACE HMFM Services API is to be used by FACE UoCs implemented using POSIX services, while the ARINC 653 Health Services are to be used by FACE UoCs implemented using ARINC 653 services.

Table 137: HM Options per OS Configuration

| | ARINC 653 Health API Feasible? | HMFM Services API Feasible? |
|-----------------------|---------------------------------------|------------------------------------|
| Pure ARINC 653 OS | Yes | Yes |
| POSIX on ARINC 653 OS | Yes | Yes |
| Pure POSIX OS | No | Yes |

It is not a FACE UoC conformance requirement to use HMFM Services. The FACE Technical Standard provides the option for a FACE UoC developer to have a similar HMFM API in all of the operating system configurations examined.

3.5 HMFM Event Flow

Figure 12 illustrates the event flow of a fault being handled via the HMFM API.

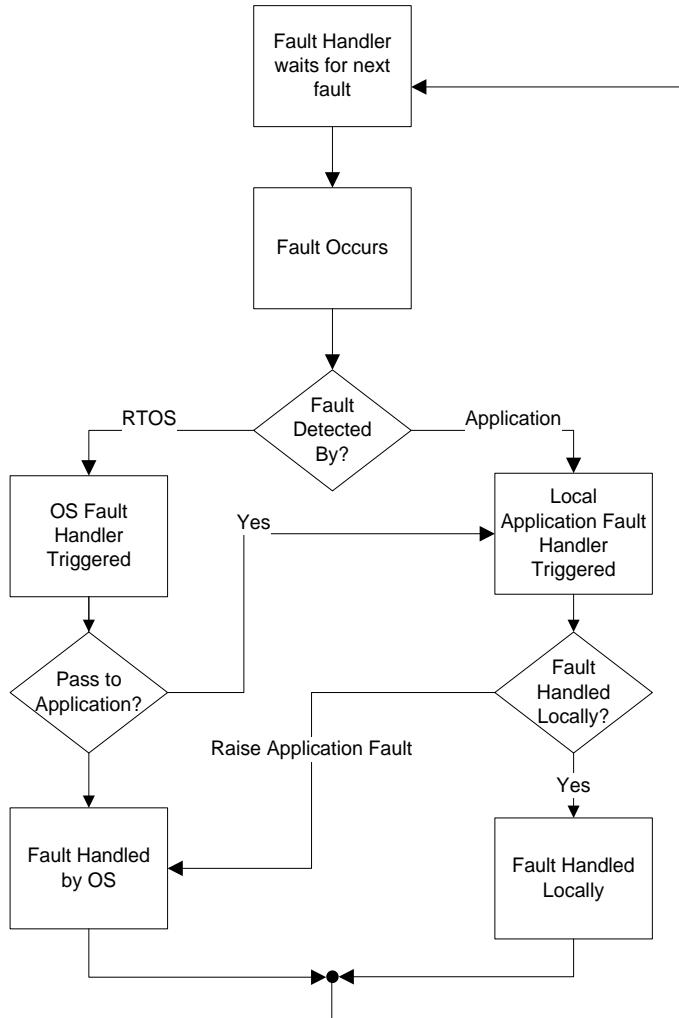


Figure 13: HMFM Event Flow

All fault handling and management software remains quiescent until a fault occurs. If the application has installed a fault handler, then that handler will be blocked waiting for a fault to occur. If the fault is detected by the operating system and can be processed by a thread-level fault handler (e.g., a numeric exception or memory access error), the operating system will trigger the execution of the application-provided fault handler if one is installed; otherwise, it will perform the default action. Similarly, if the application detects a fault and reports it via the appropriate HMFM API service, then the fault handling support is triggered. It is important to note that any operating system-triggered fault indicates that an unexpected and potentially serious error has occurred in the application.

In either case, if installed, the application-provided fault handler will execute. It will execute in the context of a special thread and there will be restrictions on the operating system services which may be invoked by the fault handler. The application-provided fault handler will take action which may include:

- Attempting to recover from the fault
- Passing fault information to a logger or partition-level health monitor
- Propagating the fault to the operating system for further action

3.6 HMFM Implementation Guidance

This section contains implementation guidance for the whether ARINC 653 Health or FACE HMFM Services API are to be used based upon the OS configurations defined in the previous section.

3.6.1 Implementation on ARINC 653 OS

In this OS configuration, applications use the ARINC 653 Health Services.

3.6.2 Implementation on ARINC 653 OS with POSIX APIs

In this OS configuration, applications in the ARINC 653 operational environment are to use the ARINC 653 Health Services. Applications using the POSIX operational environment are to use the FACE HMFM Services API.

The implementation of FACE HMFM Services API would be a simple wrapper on the ARINC 653 Health Services and it would directly utilize the underlying ARINC 653 Health Services.

3.6.3 Implementation on Pure POSIX OS

The FACE HMFM Services API is based upon ARINC 653. There is nothing that is completely comparable to these services in the POSIX standard. The HMFM Services API provides a common set of services across all FACE Profiles. Supporting HMFM in a pure POSIX environment involves technical compromises. The following is expected regarding an implementation on a pure POSIX system:

- Provide a useful functionality subset of HMFM Services API even though the POSIX standard does not provide the full complement of underlying Health Services capabilities found in ARINC 653

The following are guidelines for a POSIX standard-based FACE HMFM implementation:

- Use the *sigaction()* method to install the signal handlers; this results in the signal handler being used with the following prototype:

```
void Signal_Handler(
    int      signal,
    siginfo_t *info,
    void     *extra
);
```

- The signal parameter will contain the signal which has occurred; this can be mapped to a FACE HMFM fault type

The signal parameter indicates the type of signal which is being received. The implementation will map those to ARINC 653 faults. On some of the signal types, an operating system may provide additional information in the *si_code* field in the *siginfo_t*

structure pointed to by the *info* parameter. If available, this information may be used to further identify the fault. Table 138 provides a mapping from a subset of POSIX signals to the FACE HMFM faults which are based on those defined by ARINC 653.

Table 138: POSIX Signals Mapped to FACE Errors

| POSIX Signal | ARINC 653 Error | Comments |
|--------------|------------------|--|
| SIGFPE | NUMERIC_ERROR | |
| SIGILL | MEMORY_VIOLATION | If <i>si_code</i> is ILL_BADSTK. |
| SIGILL | STACK_OVERFLOW | Assume memory violation if <i>si_code</i> is not ILL_BADSTK. |
| SIGSEGV | MEMORY_VIOLATION | |
| SIGBUS | MEMORY_VIOLATION | |
| SIGSYS | ILLEGAL_REQUEST | |
| SIGXFSZ | ILLEGAL_REQUEST | |
| SIGQUIT | POWER_FAIL | System Integrator chooses power fail signal. |
| SIGTERM | POWER_FAIL | System Integrator chooses power fail signal. |
| SIGABRT | POWER_FAIL | System Integrator chooses power fail signal. |

The POSIX standard does not define a signal to indicate when a power failure has occurred. If the System Integrator desires to make this fault available to POSIX applications, then a POSIX signal is selected to indicate this fault. Multiple signals were identified in Table 138 to indicate a power failure. There is no FACE requirement to use a particular signal or for a hardware platform to detect that power has failed.

- The *info* parameter is a pointer to an instance of a *siginfo_t* structure
This structure contains the field *si_addr* which indicates the address of the faulting instruction. If the signal is SIGFPE or SIGILL, the structure also contains more detailed information about the fault.
- The *extra* parameter is a pointer to an implementation-defined structure; on some GNU/Linux versions, this is a pointer to a *ucontext_t* structure

The fields of the FAULT_STATUS_TYPE structure can be filled in as follows:

- The FAILED_THREAD_ID field is filled in with the ID of the thread executing the signal handler; this is returned by *pthread_self()*
- The FAILED_ADDRESS field is filled in with the *si_addr* field of structure pointed to by the *info* parameter to the signal handler

- The MESSAGE field is filled in generically with a string such as “*fault*”, the signal name, or with more detailed information based on decoding the *si_code* in the *siginfo_t* structure pointed to by the *info* parameter to the signal handler

Providing detailed information via the *si_code* may not be supported by the operating system and there is no requirement for the implementation to provide any detailed information via the MESSAGE field. It is not a FACE requirement to provide any information not defined by the HMFN API and its data structures. However, this information may be useful to System Integrators in evaluating the source of the fault and eliminating it in the future.

Some examples of more detailed information include the following using the *si_code* information:

- SIGFPE could indicate multiple conditions including integer or floating point overflow, underflow, or division by zero
- SIGILL could indicate the execution of an illegal instruction, a privileged instruction, or a coprocessor error
- SIGSEGV could indicate an access of an invalid address or insufficient privileges
- SIGBUS could indicate an access with illegal alignment or an attempt to access a non-existent physical address

The POSIX standard has no concept of ARINC 653 deadline-based scheduling. Based solely upon POSIX capabilities, the DEADLINE_MISSED fault cannot be generated.

Based upon the capabilities of the RTOS, an implementation may not be able to generate all fault types.

Propagation of a fault to a Health Manager outside of a POSIX process uses implementation-dependent mechanisms.

The goal of the HMFN Services API is to provide application portability across all OS profiles. An implementation on a pure POSIX OS provides source code-level compatibility but may have limitations on the faults which are raised by the HMFN framework.

3.7 HMFN and Non-Volatile Storage

The FACE HMFN capabilities are based on those defined in the ARINC 653 Specification. ARINC 653 intentionally does not include non-volatile storage of faults as an operating system responsibility. The rationale that went into this decision includes the following:

- Operating system-detected faults are a small subset of the overall system fault and health management

System fault management and logging was considered a System Integrator/platform provider responsibility. Different organizations that built ARINC 653-based systems had significantly different management techniques for this.
- Not all platforms support non-volatile storage
- There are some operating system-detected faults that are for conditions where the system is in a state that attempting to write to non-volatile storage is questionable

In certain fault scenarios, writing to non-volatile storage could result in data corruption or potentially prevent system recovery.

ARINC 653 does not consider non-volatile (or even volatile) storage of operating system-detected faults to be an operating system vendor implementation-specific capability. Platforms that required non-volatile storage could utilize one of the ARINC 653 Part 2 optional service categories (file system or logbooks) to store faults in non-volatile memory. The FACE OS Safety and General Purpose Profiles include support for the ARINC 653 File System services (and corresponding POSIX file system APIs) to support non-volatile requirements (when the platform provides non-volatile storage hardware). This permits non-volatile storage to be supported via a portable means.

ARINC 653 health monitoring includes interfaces/descriptions intended to provide support for communicating operating system fault messages to some compute module-level HM capability. The intent is these capabilities would be configured to forward operating system-related faults when logging of such faults was to be included as part of the platform.

3.8 Fault Handler Guidance

The reader is reminded that the FACE HMFMS Services are based on the ARINC 653 Health Services. It is intended that in a POSIX operational environment which provides the ARINC 653 Health Services, the FACE HMFMS Services be implemented directly in terms of the ARINC 653 Health Services. In that light, the FACE HMFMS Services use the terms “error” and “thread” where the ARINC 653 Services use the terms “fault” and “process”. This section is intended to be applicable to either operational environment given API and terminology mappings.

The basic structure of a user error handler is a thread body with an infinite loop around an invocation of the *FACE_HMFM_Get_Fault_Status()* method. This method blocks the thread until a fault occurs. When a fault occurs, this method will return from that blocking state with information about the fault. The user fault handler will be able to examine the fault to determine its type. The fault handler can look at the fault type and use that information to decide whether it is a fault it knows how to address or whether it should be propagated to the next level.

```
Fault_Handler:  
    loop  
        FACE_HMFM_Get_Fault_Status()  
        if the fault is one handled by this handler, the  
        application has the option to  
            a) propagate the fault to the next level,  
            b) reset the partition,  
            c) stop (e.g., idle) the partition,  
            d) reset the offending thread,  
            e) stop (e.g., idle) the offending thread, or  
            f) clean up the error source  
        In cases a - c, the fault handler will not return.  
        In cases d - f, the fault handler will attempt to return  
        to the application.  
    else the fault is not processed by this handler  
        application fault propagated to the next level via  
        FACE_HMFM_Raise_Application_Error()  
    end loop
```

Figure 14: HMFM Fault Handler Pseudo-Code

Care must be taken when writing a fault handler. The fault handler should under no circumstances perform a blocking action. It may invoke a non-blocking communications method to pass information to another partition. Hopefully, that other partition is not in a fault state and can undertake more complex actions such as logging the fault.

The fault handler can attempt thread recovery but it must be cautious in doing so. The faulting thread has both a fault condition and a logical state which must be addressed for successful recovery to occur. In some cases, correct fault recovery will require addressing both the fault condition and the logical state of the faulting thread. The specific operating system services available to address the faulting thread will vary based upon whether the partition is executing a POSIX or ARINC 653-based application.

To facilitate recovery, the fault handler must directly address the source of the fault and resolve it. The fault handler may be designed such that a single event fault allows the application to retry the operation. If the fault occurs again, then remedial action may be taken. For example, a fault caused by a memory parity error may be a single event. However, if a programming error results in a thread causing a memory violation, then the fault will occur again if the operation is retried. In this case, in order to recover, the “pointer” that the thread is using must be corrected or the thread will reuse the same invalid memory reference. Similarly, if the thread has a floating point exception due to a programming error, a fault indicator will likely need to be cleared in hardware and the faulting numeric operation will have to be corrected. In all these cases, the RTOS may provide the fault handler with the address of the faulting instruction and faulting thread’s context. Recovery is possible but requires processor and RTOS-specific information.

If the fault has occurred as the result of a soft or hard hardware fault, it is possible that the fault handler can address the fault and the application recovers. This would require knowledge of the fault, recovery actions possible, and, in some circumstances, the existence of backup devices and how to switch to them.

Assuming that the fault is one which can be recovered from, there is still the logical state of the faulting thread to address. In the case of a floating point exception, this might require knowledge of the algorithm used by the faulting thread. Knowing that there was a divide by zero fault and putting an arbitrary result into the destination register is one thing. Knowing the value to put into that destination to correct this invalid computation and allow its algorithm to produce a meaningful result is another matter entirely.

Other parts of the logical state of a thread include any locks held on resources, resources allocated, partial computations, and application state information. For example, the fault may have occurred while a thread held locks. Those locks would not automatically be released and it would be the responsibility of the recovery actions taken or initiated by the fault handler to release those. Similarly, if the thread has dynamically allocated any resources (e.g., RTOS objects, memory), then those allocated resources may have to be freed or placed into a known state.

The usual case is that an application reaches the fault handler when something has happened that it could not foresee or avoid programmatically. This implies that the application designers did not expect the condition, thus could not avoid it and are unlikely to know how to recover. For example, if a divide by zero were expected by the application designers, then it would have been trivial for them to include a test if the denominator were zero, perform an alternative computation, and avoid the faulting operation.

In addition, all software in a system may have to be reviewed and undergo rigorous coverage testing. In order to fully test complex fault recovery actions, there must be ways to generate the appropriate fault conditions reliably in a testing environment. The correctness of simple fault recovery actions is much easier to demonstrate. Thus, it is common to perform no recovery attempt and reset or idle the partition.

4 Configuration Services API

4.1 Introduction

Configuration allows modification of a FACE UoC's run-time behavior from input stored on configuration media (e.g., files, shared memory, hardware strapping) without modifying the component's executable object code. For any given capability, the selection and interconnection of components could be performed in multiple ways. Configurable components have the potential for greater portability than components designed for a single platform or mission.

4.1.1 Summary of Changes from FACE Technical Standard, Edition 2.1

The following changes to the Configuration Services API have occurred since FACE Technical Standard, Edition 2.1:

- The Configuration Services API is new in FACE Technical Standard, Edition 3.0

4.1.2 How to Read this Section

The information contained in this portion of the Reference Implementation Guide should be used when developing software that implements the FACE HMFMS Services, uses FACE HMFMS Services, or uses ARINC 653 Health Services.

4.2 FACE Configuration Categories

FACE configuration implementations fall into one of the four following categories:

1. No Software Configurability Available
2. Legacy Local Configuration
3. API-based Configuration
4. Hybrid Configuration

4.2.1 No Software Configurability Available

The No Software Configurability Available category is defined as component behavior and must be handled through software modification.

4.2.2 Legacy Local Configuration

The Legacy Local Configuration category is defined as configuring a component by direct access to configuration media. The media's properties (e.g., name, location, content, format) are tightly coupled to the configurable component. The configurable component processes the data to modify the component's behavior without modifying its executable object code. Alternatively, behavior can be altered through physical mechanisms such as hardware strapping. The validity

of the configuration data (including its format, structure, type fidelity, boundary checking, and allowable enumeration values) is typically enforced by the configuration parser within the component.

Figure 15 depicts two entities involved in this exchange: the configurable component and the configuration media. These entities are directly related by operating system calls (e.g., `open()`, `read()`, `close()`). Processing of configuration data occurs within the component. To avoid driving cost, effort, and recertification into legacy components, using the FACE Configuration Services API should be considered over the Legacy Local Configuration approach.

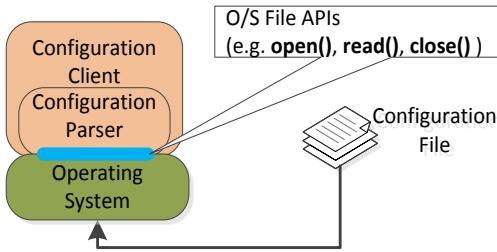


Figure 15: Legacy Local Configuration

When the Software Supplier identifies that Legacy Local Configuration is used, component characterization, such as hard-coded configuration expectations, must be documented.

4.2.3 API-Based Configuration

To support the portability of FACE UoCs, the configuration of those components must be portable as well. The capabilities provided by the Configuration Services API were designed to provide an architectural point of variability where the System Integrator may supply an implementation which supports storage of configuration information in system-specific formats on system-specific media.

This division of labor provides the System Integrator flexibility regarding where the configuration data is stored and how it is accessed. The System Integrator may select or create an implementation of a Configuration Services API for each configurable component. The intention is for the Configuration Services API to provide an abstraction of accessing configuration information. This is analogous to how the TSS abstracts communication. No effort is required by Software Suppliers when changes in configuration location, media, or access methods are made.

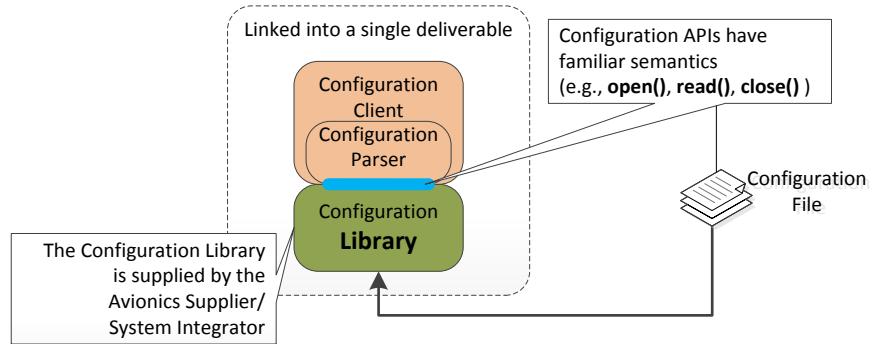


Figure 16: FACE API-Based Local Configuration

The Software Supplier is encouraged to use the Configuration Services API introduced in FACE Technical Standard, Edition 3.0. Using this API enables the System Integrator to supply an appropriate Configuration Services API for linking to configurable software components.

The benefit to the Software Supplier is that a single standardized API for configuration tasks with a defined behavior may now be used in components targeted for any FACE Segment. There is also no need to modify the component for new environments because the implementation requiring the change is now supplied by the System Integrator.

The benefit to the System Integrator is the ability to control where the configuration data is stored. It allows the creation of a library which maps the location-agnostic Configuration Services API calls to platform and transport-appropriate implementations. It does all this without the need for the Software Supplier (which could very likely be another vendor) to modify the component.

4.2.3.1 *Migration from Legacy Local Configuration to FACE API-Based Configuration*

Legacy Local Configuration has the drawback of limiting the potential for portability of a component to other systems. Two examples illustrating this point are:

- Secure operating systems having no file system to hold a configuration file
- A need to relocate the configuration file without modifying the configurable component

The Configuration Services API supports a migration path for legacy components by providing data movement and API usage behaviors similar to the operating system API used to access files; e.g., *open()*, *read()*, and *close()*. The Configuration Services API has a few additional and required formal parameters in their API calls to distinguish them from the standard OS APIs.

The migration of a FACE UoC from Legacy Local Configuration to API-based Configuration requires the Software Supplier to use the Configuration Services API rather than the OS API. This is accomplished by substituting the OS API with the Configuration Services API and incorporating the relevant OS API functions in the Configuration Services API. For example, using *Open(CONFIG)*, *Read(CONFIG)*, and *Close(CONFIG)* instead of *open()*, *read()*, and *close()* from the operating system. These operating system calls are then incorporated into the Configuration Services API. This allows the Software Supplier to test the configuration of their component and also allows the component's configuration information to be provided from

different media and underlying representations on multiple platforms implementing the FACE Reference Architecture.

4.2.4 Hybrid Configuration

Hybrid Configuration is accomplished by using a combination of Legacy Local Configuration and API-based Configuration.

4.3 Recommended Configuration Services Artifacts

The configuration of a component is commonly expressed in three (3) artifacts:

1. An XML Schema Definition (XSD) documenting the form of the configuration data, known as the Component Configurability Definition (CCD)
2. An XML file containing the configuration data for a component, called the Configuration Set Identifier List (CSIL)
3. A transformation process for converting the XML file to a format readable by the configuration library on the target hardware, also referred to as the Configuration Set Encoding Package (CSEP)

The FACE Technical Standard requires that a UoC's configuration be expressed in XML. Given that each deployment of a UoC may require the creation of a new configuration, providing an XSD will help ensure that each deployment's unique configuration is correct.

Artifact 3 may be unnecessary if the target system implementation natively supports XML. However, it is common to perform a transformation from an easy to edit format used by humans on a development computer to an easy to process format used by the application on the embedded system. This tends to reduce both the size and complexity of the configuration data and eases processing of it on the target system. This reduction also eases the burden of verifying the correctness of the software reading the configuration information on the target system.

Each of these items is specifically addressed in the following sections.

4.3.1 Component Configurability Definition

The Component Configurability Definition (CCD) is expressed in an XSD document and describes the following FACE UoC characteristics:

- Identification of configurable parameters
- Ranges of valid configurable parameter values
- Behavioral impact associated with each configuration value
- Default values for configurable parameters
- Disallowed combinations of configuration values

4.3.1.1 *Identification of Configurable Parameters*

Configurable parameters are the named attributes of configurable components that affect behavioral modification of the component without alteration of its executable object code. These

named attributes are called Configuration Parameter IDentifiers (CPIDs). The act of assigning values to configurable parameters may be accomplished with a configuration data file containing the CPIDs and their associated values.

The actual configurable parameter names, valid values, and file formats are defined and documented by the Software Supplier. For any given capability provided by a FACE UoC, various algorithms and feature sets may be selectable to configure the component. The values assigned to the configurable parameters determine which algorithm or feature set is used by the FACE UoC.

4.3.1.2 *Ranges of Valid Values for Configurable Parameters*

The range of valid values for each configurable parameter should be specified in the CCD to provide the ability to validate the component configuration files.

4.3.1.3 *Behavioral Impacts Associated with Each Configuration Value*

Each behavioral alternative must be defined for each valid value (or range of values) for each CPID associated with each configurable parameter. The range of valid values of the CPID TACAN_MODE is 0 through 4 with behavioral impacts as provided in Table 139.

Table 139: Configuration Parameter Example

| CPID | Value | Behavioral Impact |
|------------|-------------|---|
| TACAN_MODE | 0 | Power is OFF, Tactical Air Navigation (TACAN) is not operational. |
| | 1 | Receive mode, TACAN operates in Air-to-Ground Receive mode only. |
| | 2 (default) | T/R mode, TACAN operates in Air-to-Ground mode with simultaneous Transmit and Receive mode. |
| | 3 | A/A_RECEIVE mode, Air-to-Air Receive only mode. |
| | 4 | A/A_T/R mode, Air-to-Air mode with simultaneous Transmit/Receive. |

4.3.1.4 *Default Values*

Software Suppliers should identify the default value for all configurable parameters. Configurable parameters are not required to be explicitly set if the default value is the desired configuration value. In the example of Table 139, the default value is specified as 2 (T/R mode).

4.3.1.5 *Disallowed Combinations*

There are cases where selections of valid values for CPIDs are invalid as a group for the component. The example in Section 4.3.1.6 shows the case where “InPrecision” is set to “Single” and “OutPrecision” is set to “Double”. The significance of this example is that “Single” is normally a valid value for “InPrecision” and “Double” is normally a valid value for “OutPrecision”. However, these values are now an invalid combination for “InPrecision” and “OutPrecision” as indicated using the *xs:assert* tag in the XSD in Section 4.3.1.6.

4.3.1.6 CCD Example

The example CCD defines two CPIDs: “InPrecision” and “OutPrecision”. For each CPID, the attribute type=“PrecisionType” is a reference to a type definition which enumerates the range of valid values (“Single” and “Double”). The behavior of each value is described in the comments containing the word “BEHAVIOR”. The attribute specification of “default=“Double”” informs the reader that, if not otherwise specified, the value assigned to “InPrecision” will be “Double”. The assertion line disallows setting “InPrecision” to “Single” and “OutPrecision” to “Double”.

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
    xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    vc:minVersion="1.1">
    <xss:element name="ExampleComponentConfiguration">
        <xss:annotation>
            <xss:documentation>This XML Schema Definition File describes
                valid configuration sets for the Example Component
            </xss:documentation>
        </xss:annotation>
        <xss:complexType>
            <xss:sequence>
                <xss:element name="InPrecision" type="PrecisionType"
                    default="Double">
                    <xss:annotation>
                        <xss:documentation>The CPID "InPrecision" must be
                            assigned a value of the type "PrecisionType".
                            If not specified, InPrecision will take on the
                            default value of "Double".
                            BEHAVIOR: Selecting "Single" performs input
                            calculations in single precision math.
                            BEHAVIOR: Selecting "Double" performs input
                            calculations in double precision math
                        </xss:documentation>
                    </xss:annotation>
                </xss:element>
                <xss:element name="OutPrecision" type="PrecisionType">
                    <xss:annotation>
                        <xss:documentation>The CPID "OutPrecision" must be
                            assigned a value of the type "PrecisionType".
                            If not specified, OutPrecision will take on the
                            same value as "InPrecision".
                            BEHAVIOR: Selecting "Single" performs output
                            reporting in single precision math.
                            BEHAVIOR: Selecting "Double" performs output
                            reporting in double precision math
                        </xss:documentation>
                    </xss:annotation>
                </xss:element>
            </xss:sequence>
            <xss:assert test="not ((InPrecision eq 'Single') and
                (OutPrecision eq 'Double'))">
                <xss:annotation>
                    <xss:documentation>This assertion ensures that the case
                        where InPrecision = Single and OutPrecision = Double
                        does not successfully validate against this XML schema.
                </xss:annotation>
            </xss:assert>
        </xss:complexType>
    </xss:element>
</xss:schema>
```

```

        </xs:annotation>
    </xs:assert>
</xs:complexType>
</xs:element>
<xs:simpleType name="PrecisionType">
    <xs:annotation>
        <xs:documentation>This is the definition of the simpleType
            named "PrecisionType". Please notice that both
            InPrecision and OutPrecision use this type.
        </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:enumeration value="Single">
            <xs:annotation>
                <xs:documentation>This value selects Single Precision
                </xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="Double">
            <xs:annotation>
                <xs:documentation>This value selects Double Precision
                </xs:documentation>
            </xs:annotation>
        </xs:enumeration>
    </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Figure 17: Configuration Services XML Example

4.3.2 Component Configuration Set

A Component Configuration Set (CCS) is an XML document created by the System Integrator that is valid with respect to the CCD supplied by the Software Supplier. The CCS is scoped to a single component and captures a set of configuration value choices for a subset of the component's CPIDs.

4.3.2.1 Component Configuration Set Example

The following example CCS is valid with respect to the CCD as shown in Section 4.3.1.6 containing the “ExampleComponentConfiguration” element. Within that XML element, two other elements are found. The “InPrecision” element is given a value of “Double” as allowed by the CCD with a type named “PrecisionType”. The “OutPrecision” element is optional in this CCS. The “OutPrecision” parameter is given a value of “Single” by the System Integrator as is allowed by the CCD.

```

<?xml version="1.0" encoding="UTF-8"?>
<ExampleComponentConfiguration>
    <InPrecision>Double</InPrecision>
    <OutPrecision>Single</OutPrecision>
</ExampleComponentConfiguration>

```

4.3.3 Configuration Set Identifier List

The Configuration Set Identifier List (CSIL) is delivered to the System Integrator as an explicit description of the configurability of the component, listing named sets of configurable

parameters. A CSIL is an XML document that validates against the CSIL XSD. There may be more than one CSIL per component. Table 140 defines the Container Name and Container Content of the CSIL.

Table 140: Configuration Container Terminology

| Container Name | Container Content |
|----------------|--|
| ContainerID | A token supplied at run-time by the configurable component to the <i>Open(CONFIG)</i> service which is mapped to a storage mechanism of configuration data. |
| CSID | A token supplied at run-time by the configurable component to the <i>Read(CONFIG)</i> service which is mapped to a mechanism identifying a configuration entity group within the scope of the current ContainerID. |
| CPID | The Configurable Parameter IDentifier(s) detail the content of a configuration entity group, specifying the configuration content for each Configuration Set Identifier (CSID). |

4.3.3.1 Configuration Set Identifier List Schema

Figure 18 provides a pictorial representation of the XSD.

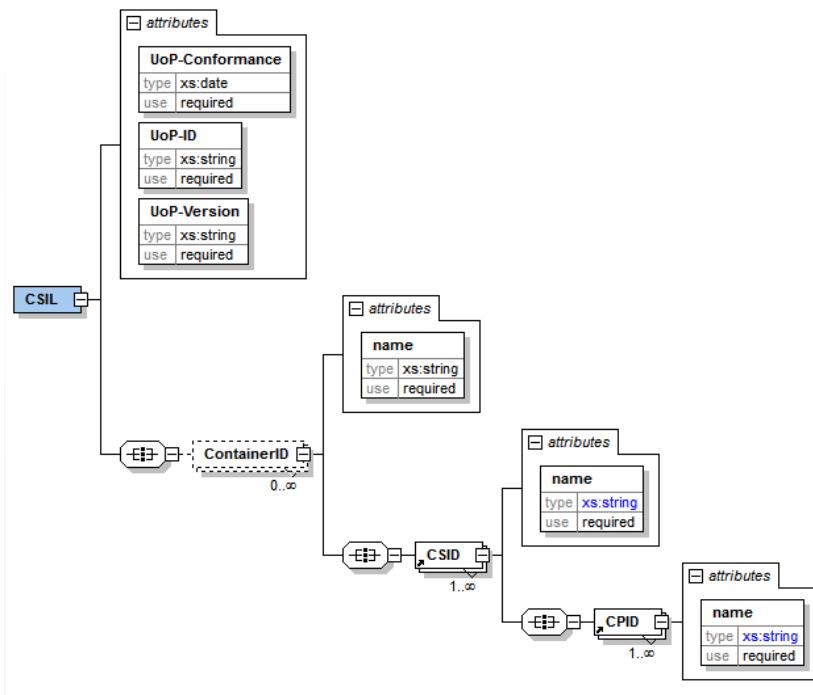


Figure 18: CSIL Schema Pictorial Representation

The following is the XSD used for validation of the CSIL.

```

<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <xss:annotation>
    
```

```

<xs:documentation>Configuration-Set Identifier List (CSIL)
    describes the "data_set" names that may be requested by
    the configurable component and the set of configurable
    parameters to be supplied when that CSIL-ID is requested
</xs:documentation>
</xs:annotation>
<xs:element name="CSID">
    <xs:complexType mixed="false">
        <xs:all>
            <xs:element ref="CPID" maxOccurs="unbounded"/>
        </xs:all>
        <xs:attribute name="name" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
<xs:minLength value="1"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
<xs:element name="CPID" nillable="false">
    <xs:complexType mixed="false">
        <xs:attribute name="name" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
<xs:minLength value="1"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
<xs:element name="CSIL">
    <xs:complexType mixed="false">
        <xs:all>
            <xs:element name="ContainerID" minOccurs="0"
                maxOccurs="unbounded">
                <xs:complexType mixed="false">
<xs:all>
                <xs:element ref="CSID" maxOccurs="unbounded"/>
            </xs:all>
            <xs:attribute name="name" type="xs:string"
                use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:all>
            <xs:attribute name="UoP-Conformance" type="xs:date"
                use="required"/>
            <xs:attribute name="UoP-ID" type="xs:string" use="required"/>
            <xs:attribute name="UoP-Version" type="xs:string"
                use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>

```

Figure 19: CSIL Schema

4.3.3.2 Configuration Set Identifier List Examples

4.3.3.2.1 Configurable Component CSIL Example

The example CSIL shown below consists of a single root element named “CSIL” containing information for two configuration data containers named “MyStuff” and “C:\FACE\NavigationComponent\Config\Mypdi.pdi”. The CSID “all” is defined for the “MyStuff” container and the CSID “compact” is defined for the “C:\FACE\NavigationComponent\Config\Mypdi.pdi” container. Each configuration data container enumerates the names of applicable CPIDs.

```
<?xml version="1.0" encoding="UTF-8"?>
<CSIL xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="FACE_CSIL_V1.xsd"
      UoP-Conformance="2014-06-05"
      UoP-ID="TrackCorrelator"
      UoP-Version="3.0a">
  <ContainerID name="MyStuff">
    <CSID name="all">
      <CPID name="InPrecision"/>
      <CPID name="OutPrecision"/>
    </CSID>
  </ContainerID>
  <ContainerID name="C:\FACE\NavigationComponent\Config\Mypdi.pdi">
    <CSID name="compact">
      <CPID name="InPrecision"/>
      <!-- OutPrecision automatically tracks InPrecision -->
    </CSID>
  </ContainerID>
</CSIL>
```

4.3.3.2.2 Non-Configurable Component (Empty) CSIL Example

In the case of a non-configurable component, the document is empty except for the CSIL open tag, required attributes, and close tag. A CSIL describing a non-configurable component is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<CSIL xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="FACE_CSIL_V1.xsd"
      UoP-Conformance="2014-06-05"
      UoP-ID="TrackCorrelator"
      UoP-Version="3.0a">
</CSIL>
```

4.3.4 Configuration Set Encoding Package

The Configuration Set Encoding Package (CSEP) is the documented approach of converting configuration parameters into a component’s encoded format for deployment. The CSEP consists of a document identifying the transformation process, any necessary tools to facilitate the process, and example input and output files.

The documented process for transforming the artifacts into the component encoded format is necessary for all components. The documented process must include sufficient detail for the System Integrator to create configuration files to be consumed by the component. If the process requires tool(s), platform-independent tool(s) must be provided. Example input (CCS and CSIL)

and corresponding output file(s) must accompany the CSEP to assist the System Integrator in validating that the process is being executed correctly.

4.4 Configuration Services API Comparison to Operating System Services

The services provided by the Configuration Services API closely mimic the operating system's file API and behaviors in an effort to mitigate the impact of replacing the operating system calls. For example, variants of the *open()* operating system call (e.g., *fopen()*, *mqopen()*) would be replaced by *Open(CONFIG)*. Additionally, they support the concept of containers which contain sets of name/value pairs. A single UoC can access a single or multiple containers and query for the value associated with a particular name or key.

- *Initialize(CONFIG)* prepares the Configuration Services for use by the component
- *Open(CONFIG)* establishes a session with the Configuration Services implementation
The component passes in the container name and a handle is returned for future access using services such as *Read(CONFIG)*, *Seek(CONFIG)*, and *Close(CONFIG)*.
- *Read(CONFIG)* retrieves configuration information
A particular data set from the container is specified through the *set_name* parameter.
- *Seek(CONFIG)* determines where data should be read from by specifying an origin and offset
It supports stream and file access in implementations using variants of the operating system call to *seek()* (e.g., *fseek()*, *lseek()*). *Seek(CONFIG)* may not be meaningful for a particular target system encoding.
- *Close(CONFIG)* is used at the conclusion of operations on a particular configuration set whose handle was returned by a previous call to *Open(CONFIG)* and can clean up resources no longer needed

The *Open(CONFIG)* API has an input parameter *container_name* used by the Configuration Services API to identify the desired configuration data resource. The API's *container_name* parameter is a unique identifier allowing the library to unambiguously resolve the configuration data resource; be it a file, database, or any other media as determined by the System Integrator. Some possible options for identifying a configuration data resource through *container_name* are:

- Using the actual name of a configuration path and file for the library to directly access the named file
- Using the supplied path and filename as an indirect mapping to a differently named or located resource
- Using a string such as "MyStuff" which the library would resolve to access the appropriate configuration resource

The *container_name* parameter to the *Open(CONFIG)* service refers to the storage of the configuration data as depicted in Figure 20. Access to the configuration data container is in contrast to accessing the configuration data itself, which is achieved by specifying the *set_name* in the *Read(CONFIG)* service. The Configuration Services API implementation can use the

container_name parameter to determine the location and access methods needed. For example, the container could be an INI file, a binary blob, or an SQL database.

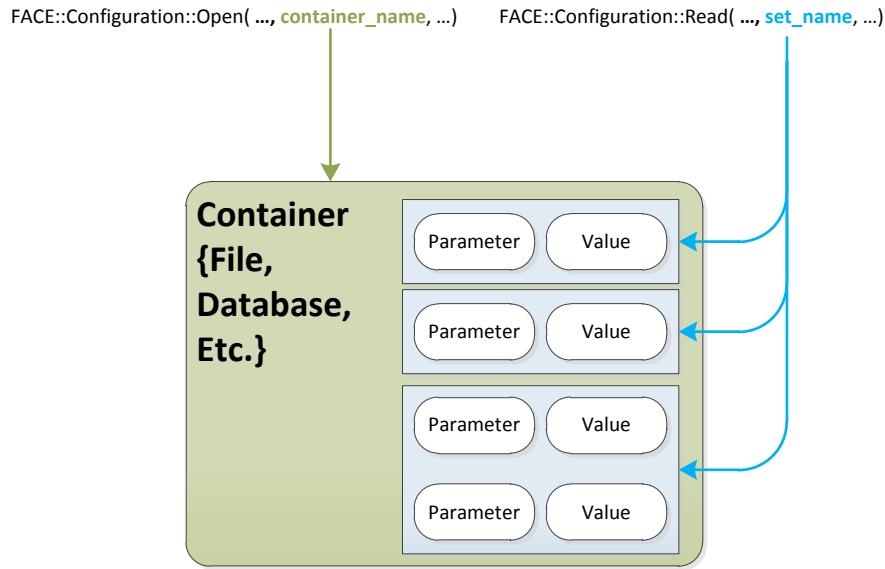


Figure 20: Configuration Data Container

The *Read(CONFIG)* service has an input parameter *set_name* specifying the desired set of encoded configuration data items and their assigned values. A configuration entity is composed of a named parameter (CPID) and an assigned value. The configuration entity could be used alone or in a group of configuration entities. The configuration entity is encoded in the format of the configurable component. The *set_name* parameter is used to identify the desired configuration entity. The *set_name* parameter may not directly address the configuration entity, which is similar to the *container_name* parameter of the *Open(CONFIG)* service. In both the *Open(CONFIG)* and *Read(CONFIG)* services, the Configuration Services API maps the input parameters to the desired container and configuration entity.

5 Programming Language Run-Time Guidance

5.1 Introduction

UoCs are designated as portable when they can be redeployed on different FACE Computing Environments without technically requiring more than a recompilation of the UoC and a re-linking of software libraries, language run-times, and/or frameworks.

A language run-time is an executable object code library used by a compiler (and linker) to implement features and capabilities of a software programming language such as C, C++, Ada, or Java.

Language run-times generally consist of two interfaces: explicit and implicit:

Explicit interfaces are defined as part of a programming language standard and invoked directly by a UoC. For example, the *sin()* and *cos()* methods from the C Standard Library are explicit interfaces. The FACE OSS Profiles define the explicit POSIX, ARINC 653, and Standard C Library methods that a conformant UoC may use.

Implicit interfaces are known and invoked by the compiler in order to implement specific language features or to abstract common differences in capabilities supported across a processor family. The implicit interfaces are usually unique to each compiler. For example, internally generated calls to compiler built-in functions simplifying code conversion or math operations not necessarily supported by the hardware such as a long floating operation.

The FACE Technical Standard is built upon the foundation that source code developed to a strict programming language definition should be able to be compiled by multiple compilers without changing the source code. Using programming language extensions (permitted by programming language standards) decreases the possibility of successful recompilation due to unique vendor extension capabilities. Using a language framework (e.g., Ada Ravenscar profile, Embedded C++) tends to make a UoC more portable across different compilers. If moving a UoC from one FACE OSS to another, the UoC should be compiled using system build tools for the target FACE OSS.

If the FACE UoP does not follow these strict definitions, then it may be able to include support methods and programming language run-times that are not defined by the FACE Technical Standard.

5.1.1 Summary of Changes from FACE Technical Standard, Edition 2.1

The following changes to the Configuration Services API have occurred since FACE Technical Standard, Edition 2.1:

- C:
 - Require support for exact width types from `<stdint.h>` which are optional in C99
 - Ada 2012 may be used when in the General Purpose Profile

- Java:
 - Java Standard Edition updated to Edition 8
 - Java Enterprise Edition updated to Edition 7
 - OSGi version updated to 6

5.1.2 How to Read this Section

The information contained in this portion of the Reference Implementation Guide should be used when developing software that implements any UoC.

5.2 Conformance Testing Background

This section is not intended to be a complete explanation of the FACE verification process. Its purpose is to highlight the points in the process that are pertinent to this discussion. In particular, the focus is on the object code under test – not the artifacts provided by the Software Supplier.

The Software Supplier provides the Verification Authority with the object code. The object code is a special build of their software which is:

- Built against the Gold Standard Library
- Built for an arbitrary target

The FACE Conformance Policy allows for conformance testing a PCS, TSS, PSSS, or IOSS UoC on only one target platform, but the conformance statement applies to all target platforms. This is predicated on the assumption that the software under test is portable and that one verification is sufficient. This should be true as long as the software is configured in the same manner on all target platforms.

The FACE Conformance Test Suite (CTS) uses a strategy of compiling and linking against Gold Standard Libraries to test conformance to FACE Interfaces for a C/C++ non-OSS UoC. This is a different target environment from a system-specific implementation of the FACE Computing Environment. It is recommended to use GNU/Linux x86 as the CTS host environment and configure CTS to use the native toolchain. This avoids the Verification Authority needing access to any operating system-specific tools which might require licensing. Using a common free platform is simple and easy. The features of the FACE Technical Standard support the necessary source code portability between the CTS target environment and a conformant implementation of a FACE Computing Environment.

This focus on object code for a single platform means that the Verification Authority is not required to see any source code and that the object code under evaluation is not that which will be deployed. This presents two “opportunities” for differences between the conformance and production builds.

5.3 Use of Programming Language Run-Times

If a non-standard run-time is selected, it must be included in a UoP. If a standard run-time allowed by the FACE Technical Standard is selected, it may be included within the UoP, or it may reside in a software library against which the UoP is linked, as shown in Figure 21.

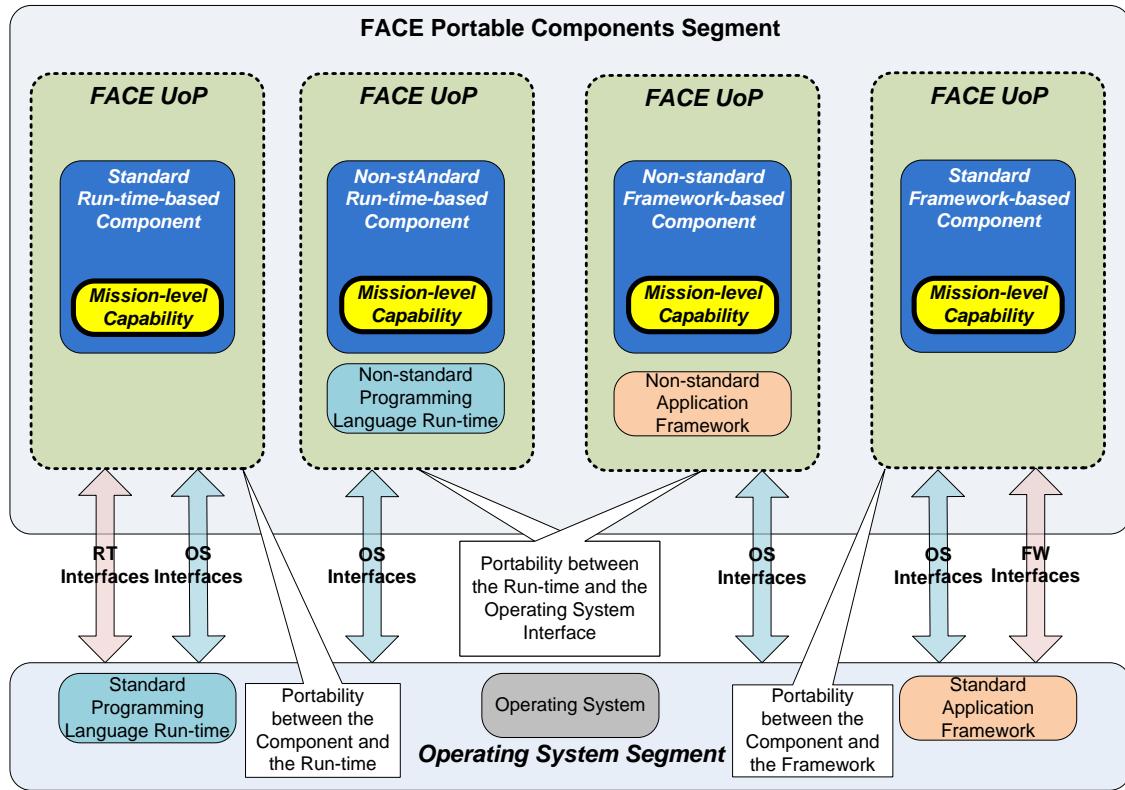


Figure 21: Language Run-Time Options

5.4

Accepted and Known Portability Issues

A goal of the FACE Consortium is to encourage software portability across multiple avionics platforms. This section is intended to capture some portability challenges facing developers of FACE UoCs. There are a few broad categories of issues addressed in this section.

One category is where software which passes the FACE CTS could still have undetected issues that result in it being non-portable software. That non-portability could manifest itself as a simple coding mistake that must be addressed as the code is migrated to a new platform or as a conformance issue on the new platform.

Another category is where standards deliberately allow for implementation-defined behavior or a choice of multiple standard behaviors. Portable programs must be carefully written with awareness of the relevant standards to avoid being impacted.

Software developers should also be aware of the impact of using common features from standards that are not part of certain FACE OSS Profiles. For example, standard in, out, and error are only available in the General Purpose Profile. Thus, a simple call to `printf()` limits the software to General Purpose. The developer must be aware of what features are really required from a profile and which are just being used out of habit.

The goal of this section is to identify some of those mechanisms so developers of FACE conformant software can avoid these pitfalls.

Issues that potentially impact C, C++, Ada, and Java programs are included.

5.4.1 Device Dependencies

PSSS and IOSS UoCs are allowed and expected to have dependencies on particular devices. If the device is not included in a system, then the associated UoCs are not applicable to that system.

5.4.2 For Conformance versus Production Build

The Verification Authority is presented with object code built against the Gold Standard Library, artifacts, and a CTS configuration. Although the compiler settings used in the CTS configuration must be compatible with those used for the Gold Standard Library build, there is no way to know if some different compiler options were used. For example, the “for conformance” build could include a *-DXXX* argument not needed in the CTS configuration file. This argument could alter the UoC in a way that is not known to the Verification Authority.

If the CTS configuration file accurately reflects the compiler and linker settings used for the production build, then the Verification Authority should be able to evaluate them. However, it is also possible that these settings differ and the Verification Authority would have no way of knowing.

5.4.3 Performance Variations

A UoC may be highly optimized for a particular microprocessor or compiler. The software itself may be completely portable to other FACE Computing Environments but not meet system performance requirements.

Similarly, the software may use an algorithm that does not scale well based on processor or memory utilization. For example, a UoC may perform adequately when used on a system with a maximum of 50 instances of something, but when moved to a system which must process 1,000 instances, performance is unacceptable.

There may also be differences in precision or accuracy of computations that makes a UoC suitable for one system but not for another.

5.4.4 Bounded or Symmetric Multiprocessing

Embedded software has historically been deployed on uniprocessor systems. This software may be written with assumptions based on that and may not scale well or operate correctly when deployed on multiprocessor systems.

5.5 Programming Language Independent Guidance

5.5.1 Endianness

A program may depend on the endianness of the underlying processor. This is usually unintentional on the part of the programmer but reflects lack of testing in a range of heterogeneous environments. This is a functional issue and likely not to be spotted until the program is ported to a processor architecture with a different endianness.

5.5.2 Word Size

Each processor has a native integer size. For most processors used by avionics, this is likely to be 32 bits. However, moving to a smaller (e.g., 16-bit) or larger (e.g., 64-bit) native integer machine could cause problems. These could manifest themselves as not writing messages or binary files that are interoperable with those written when on a processor with a different native integer size. It can also manifest itself as an increase in memory consumed.

When moving software from a larger to smaller native integer size, it is common to see mathematical overflows.

Care should be exercised to write software using properly and explicitly sized types.

The Verification Authority will not be able to spot this issue as it is a functional issue in source code.

5.5.3 Use of Non-Conformant Modes/Enumerations

The CTS checks for the use of specific methods provided by operating system, OpenGL, and segment interfaces. The operating system and OpenGL APIs include methods which take enumerated values as an argument to indicate a particular operation or mode. In some cases, the FACE Technical Standard restricts the allowed enumerated values from those defined by the upstream standard such as the POSIX standard or OpenGL. The CTS cannot test whether an enumerated value allowed by the upstream standard but not the FACE Technical Standard is used.

5.6 C and C++ Guidance

This section describes issues which are specific to UoCs implemented in C and C++.

5.6.1 Use of Conditionals

C and C++ allow for conditional compilation based on the setting of macro definitions. These expressions can be complex logical expressions or simple “if defined” or “if not defined” checks. Conditional compilation can be used to enable or disable logical program features or to adapt the program source code to a particular processor architecture or operating system. These macros can be defined by:

- The compiler based on itself, targeted operating system, and processor selection
- Arguments provided to the compiler (e.g., *-Dmyflag*)
- The program itself

The Verification Authority will be able to recognize the second case because they will be examining the CTS settings for compilation and linking if the Software Supplier provides these settings. It is possible for the software to be built for conformance with those *-Dmyflag* arguments and the CTS configuration not have them. The generated object would already have taken into account the conditional compilation and those *-Dmyflag* type arguments need not be present during conformance testing.

Ultimately, without access to the source code, the Verification Authority has no way of knowing if the first or third source of macros is being used for conditional compilation. And it is trivially simple for a Software Supplier to use compiler command line conditional directives and not include them in the CTS configuration.

5.6.1.1 *Acceptable Conditional Usage*

Conditional compilation is a commonly used feature of C and C++. Used carefully, it can increase program adaptability. This section describes a few uses of conditional compilation that do not directly negatively impact program portability.

5.6.1.1.1 *Selection of Program Features*

Complex software may have many features, not all of which are used in any particular deployment. One of the responsibilities of the System Integrator is to determine the best set of features to configure for their deployment. The most common approaches to achieving this are:

- Changing the set of files built
- Conditionally enabling/disabling sections of source code

When changing the set of files built, the software will come with instructions on how to configure it. Commonly, the user may have to edit a file for the build system or run a configure script. The mechanics are specific to the software, but the end result is building a subset of available files. An example of this is configuring an operating system for specific hardware configurations. Many device drivers and BSPs are available, but the user is configuring to only build for those appropriate to their target hardware.

When conditionally compiling, the user again may run a script or manually edit a configuration source file. Some of the “feature flags” may be specified on the command line. Again, the mechanics are specific to the software, but the end result is that based on these settings certain sections of source code within files are enabled or disabled. These feature flags are internal to the application software and do not depend on feature flags set by the operating system or compiler.

It is possible and common to mix these techniques.

A FACE UoC could be tested for conformance in a specific configuration with a specific set of features enabled. If the set of conditional compilation directives or set of files compiled to make that UoC were altered from what was actually tested for conformance, then the UoC would no longer be considered conformant. Conformance is intended to be agnostic of the FACE conformance operating environment, but not of the UoC configuration. It is possible that the configuration changes would result in non-conformant source code being enabled.

5.6.1.2 *Unacceptable Conditional Usage*

Conditional compilation can be used to adapt a program to a target platform. Since conformance is tested in a single configuration targeting a single platform, it is possible that the CTS and Verification Authority will be unable to detect that the source code contains target platform-specific code that is disabled.

Using GCC as an example, the following command shows the 239 predefined cpp symbols of a native GNU/Linux installation:

```
gcc -dM -E - </dev/null
```

Among these predefined symbols are:

- `_GNUC_` to indicate that GCC is the compiler
This can be used to distinguish between compilers from different vendors. Other macros may be provided by a vendor to distinguish specific compiler versions.
- `_x86_64_` to indicate that this is a 64-bit Intel x86 target
Other target architectures include `_PPC_` to indicate PowerPC and `_arm_` to indicate ARM.
- `_linux_` to indicate the target operating system
Other operating system predefines include `_solaris_`, `_vxworks_`, and `_rtems_...`, *etc.*
- Indicators of optional architecture features such as `_SSE2_MATH_` to indicate that the CPU has SSE2 capabilities

Since these and many more are implicitly defined by the compiler, there is no way to know if a program is evaluating them as part of conditional compilation except to analyze the source code.

Conditional compilation can be used to work around differences in different target environments or to provide more optimal solutions. These may or may not impact portability. It is important to be aware that conditional compilation can be used to disable/enable optional features and is an important way to statically configure a UoC. It should not be forbidden, but its use needs to be evaluated for portability.

5.6.2 Inline Assembly

In C and C++, it is possible to have inline assembly in a UoC submitted for conformance and it pass the CTS but not be portable to processor architectures. The code overall may or may not be portable to another target architecture. The most common way to ensure portability is to have an optimized version for one or more processor architectures and a default version in C/C++. This is the approach taken with checksum generation in the BSD TCP/IP stack.

The impact of this upon someone receiving the UoC for use would depend on the details of how this was implemented and whether a suitable implementation was available for their target.

5.7 C Specific Guidance

There are no guidance topics for UoC developers using the C Programming Language.

5.8 C++ Specific Guidance

5.8.1 Overriding `new` and `delete`

C++ includes operators to allocate and deallocate objects. These operators may be overridden either globally or as part of a specific class definition. The CPP Reference website¹⁰ gives

¹⁰ See https://en.cppreference.com/w/cpp/memory/new/operator_new.

guidance and examples on how to implement this. Overriding *new* and *delete* can be an effective way to implement application-specific memory pool management or different memory pools for different types of objects.

5.9 Ada Guidance

This section describes issues which are specific to UoCs implemented in Ada.

5.9.1 Inline Assembly

In Ada, it is possible to have inline assembly in a UoC submitted for conformance and it pass the CTS but not be portable to other processor architectures. Just as with C, it is possible to have multiple versions of the subprogram with either inline assembly targeting other microprocessors or to have a generic version in Ada. Since preprocessing is not part of the language, this is most likely accomplished by having the different implementations in separate files. Selection of the individual files would be a function of how the UoC is configured and built.

The impact of this upon someone receiving the UoC for use would depend on the details of how this was implemented and whether a suitable implementation was available for their target.

5.9.2 Accessing POSIX Functions from Ada

In general, access to POSIX services from Ada is transparent to the FACE UoC, since the relevant POSIX functions are invoked from the implementation of the run-time library. If direct invocation of a POSIX function is needed, this can be accomplished with the following approach:

- Use an Ada binding to the required POSIX functions
This may be implemented through standard Ada features (Ada-to-C interfacing; e.g., pragma Import, possibly in conjunction with unchecked conversion) or provided by the Ada run-time library vendor. A binding generator may be able to automate this process, at least in part.
- Some POSIX constructs might not be directly importable; for example, functions that take a varying number of arguments (see Section 5.9.2.1 and 5.9.2.2 for further guidance)

Note also that care must be exercised when directly invoking a POSIX function, to avoid conflicts with the run-time library.

5.9.2.1 Variadic Functions

A function with a varying number of parameters can be modeled in Ada by a set of overloaded subprograms, where each such subprogram corresponds to a specific invocation of the imported function. The following is an example using the standard variadic C function, *printf()*:

```
int printf(const char* fmt, ...);
```

Assume that the *fprint()* function is to be invoked from Ada in two ways:

- Passing a format string and also an int and two doubles
- Passing a format string and also a char and a double

This is how it can be done Ada:

```
use Interfaces.C;

package Foo is
    function printf(fmt: char_array; arg1 : int; arg2, arg3: double)
        return int;
    function printf(fmt: char_array; arg1 : char; arg2: double)
        return int;
    pragma Import(C, printf);
    Result : int;
    Result := printf( To_C("dff"), 10, 3.14, 2.718);
    Result := printf( To_C("cd"), 'z', 100.0);
end Foo;
```

Note that while this pattern does not provide access to all possible variadic invocations, it is sufficient to have access to all those used by a specific program.

5.9.2.2 *Unions*

A C union corresponds to an Ada discriminated (variant) record where there is no run-time representation for the discriminant. Pragma Unchecked_Union is the simplest way to obtain this effect in Ada. Although not in the Ada 95 standard, this pragma was one of the early features adopted in Ada 2005 and is supported by most Ada 95 compilers. Below is an example of its usage.

Consider the following example, written in C:

```
union U{
    char c;
    double d;
};
void foo(U u, int tag){.}
```

The following is an example of the Ada code that would interface with the C code defined above:

```
use Interfaces.C;

package Bar is
    type U (Discrim : int := 0) is
        record
            case Discrim is
                when 0 =>
                    C : char;
                when 1 =>
                    D : double;
                when others =>
                    null;
            end case;
        end record;
    pragma Unchecked_Union(U);
    pragma Convention(C, U);
    procedure Foo(u1 : U; Tag : int);
        pragma Import(C, Foo);
    end Bar;
```

5.10 Java Guidance

This section describes issues which are specific to UoCs implemented in Java.

5.10.1 Java Run-Time Availability

A UoC written in Java is not portable to a platform that does not have support for a Java Virtual Machine (JVM) which adheres to the FACE Technical Standard.

5.10.2 Inline Assembly

If a Java UoC uses the Java Native Interface (JNI) to invoke one or more methods that are compiled to execute natively on the platform, then the software accessed via the JNI must be FACE conformant. Logically, this is one example of a mixed language UoC. The Java portion of the UoC must be evaluated for conformance per the Java requirements and the natively compiled portion must be evaluated for conformance per the appropriate language and FACE OS Profile requirements.

5.11 Additional Considerations

Commercially available language run-times may not be FACE conformant (i.e., may have dependencies on interfaces not included in the FACE Technical Standard). Language run-times that are not FACE conformant need to be combined with a FACE conformant interface UoC in a UoP so all external interfaces are FACE conformant.

Using a FACE recognized standard run-time may increase the flexibility to manage obsolescence, but may require optimization in performance efforts to mitigate integration risks. Integration of the run-times and operating system can be beneficial as the integration risks have already been eliminated and may include optimizations that improve performance, but may limit portability.

The language run-time may include language features that compete with features in the FACE conformant operating system (e.g., threading model). These features may have different operational assumptions (e.g., when and how a thread is initially started) than the same feature in the FACE conformant operating system.

Language run-times are typically used to support a General Purpose Profile environment. For the Security and Safety Profiles, assurance activities and/or qualification concerns may restrict the language features that can be used and/or supported. Many language run-times completely implement an entire programming language standard. In these cases, the supported programming language syntax is sufficiently complete that APIs from a FACE OSS do not have to be invoked in order for the UoC to perform its intended function. These UoCs are portable to OSSs with the same level of language support.

The language run-time may require features that cannot be implemented by using the interfaces defined in a FACE OS environment.

A no-run-time solution is not always a good solution. It may provide too few language features or result in excessive object code that complicates structural coverage verification activities.

6 I/O Services Segment

6.1 Scope

This chapter provides guidance related to the I/O Services Segment (IOSS) for PSSS UoC and IOSS UoC developers. For PSSS UoC developers, the guidance relates to using the I/O Services Interface to control and communicate with I/O devices. For IOSS UoC developers, the guidance addresses topics for implementing an I/O Service. This section assumes the reader is familiar with the terms and concepts in the FACE Technical Standard, §3.4 (I/O Services Segment) and §3.5 (I/O Services Interface).

The first subsection addresses “backward compatibility” – changes in the FACE Technical Standard that would impact an IOSS UoC that is conformant to a previous edition in becoming conformant to the latest edition. The second subsection describes the design goals that motivate the characteristics of an IOSS UoC. The third subsection addresses topics on effective use of the IOS Interface from the PSSS UoC perspective. The fourth and final subsection addresses topics on implementing an IOSS UoC.

6.2 Migrating an IOSS UoC from Previous Editions

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

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

IOSS UoC requirements were added related to new features:

- LCM Services
- Injectable Interface

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

6.3 Design Goals

The design of the IOSS and the I/O Services Interface is structured to provide clarity, consistency, and extensibility.

Clarity is provided by defining a separate I/O Service for each supported I/O bus architecture. A PSSS UoC developer generally knows what kind of I/O device must be supported, and needs the I/O Service to encapsulate vendor-specific device driver details. It is unlikely to need an abstraction for every kind of I/O device, since the functional logic for processing data operations is so often tightly coupled to the I/O bus architecture. For that reason, the I/O Services Interface does not define one set of operations.

Consistency is provided by supporting multiple I/O bus architectures that follow a consistent declaration pattern for an I/O Service. The declaration pattern includes the common I/O operations that each I/O Service must provide. Some I/O Services support behavior particular to the I/O bus architecture by defining additional operations. An I/O Service also encompasses the type and variable declarations needed to configure and control an I/O device or the associated I/O bus.

Extensibility is also addressed by the declaration pattern. The declaration pattern allows the IOSS UoC developer to extend the configuration and control declarations for an I/O Service included in the FACE Technical Standard. For example, the pattern allows additional configuration parameters to be defined for a Serial I/O Service implementation. This pattern can also be used by IOSS UoC developers to support I/O bus architectures not currently included in the FACE Technical Standard in a manner that allows that IOSS UoC to be assessed for FACE conformance to a published edition.

The I/O Service provides mechanisms for configuring and querying the configuration of both the I/O connection and the I/O bus. These configuration operations are considered transactional in nature, meaning that a configuration change is intended to succeed or fail leaving the I/O Service in either a modified or an unchanged state. The configuration operations are explicitly asynchronous; i.e., they have a timeout. When invoking one of the configuration operations, the caller provides a Transaction ID, represented as a GUID_TYPE, which will be returned by the I/O Service upon subsequent calls to *Get_X_Configuration()* to communicate the last successful configuration transaction.

The choice of which parameters are bus *versus* connection parameters depends on an I/O Service implementation. Configuring a bus parameter can potentially impact multiple connections. Configuring a connection parameter impacts only the connection that is configured. Implementers of an I/O Service will select parameters that are configured as either bus or connection parameters on a case-by-case basis and document the UoC accordingly. A connection configuration change should generate a CONNECTION_CONFIG_CHANGE_EVENT and may generate a CONNECTION_STATUS_CHANGE_EVENT. A bus configuration change should generate a BUS_CONFIG_CHANGE_EVENT and may generate a BUS_STATUS_CHANGE_EVENT, CONNECTION_CONFIG_CHANGE_EVENT, and/or a CONNECTION_STATUS_CHANGE_EVENT.

6.4 Using an I/O Service

This section describes how a PSSS UoC developer uses the I/O Services Interface. A key observation for using I/O Services with this edition of the FACE Technical Standard is that there are separate interfaces for several supported I/O bus architectures. Each interface follows a consistent pattern, so a PSSS UoC developer uses the same syntax for common functions. The separation of interfaces allows parameters to be properly scoped and provides support for functions that are specifically defined for an I/O bus architecture.

An instance of an I/O Service represents a particular device bus in the system. The type of the I/O Service corresponds to the device bus architecture.

This approach encourages a PSSS UoC to be designed to abstract differences; for example, among potential serial devices using the Serial I/O Service Interface.

6.4.1 Accepting the I/O Service Assignment

A PSSS UoC needs an instance of an I/O Service in order to call operations of the I/O Services Interface. It receives this instance by implementing the Injectable Interface as described in Chapter 8. This allows the System Integrator to resolve the interface dependency. In this case, the PSSS UoC is the user and the I/O Service is the provider. It is strongly recommended that the reader review the Injectable Interface chapter before reading this section.

6.4.2 Configuring the I/O Service

In FACE Technical Standard, Edition 3.0 there are two different ways to configure functional characteristics of an IO_Service:

- Using the FACE::Configuration interface
- Using the IO Service-specific Configuration API

This section provides guidance on when to use each. Considerations when choosing include:

- Using the IO_Service-specific API is not exclusive *versus* FACE::Configuration interface
Most likely they are complimentary. The IO_Service can be pre-configured using the FACE::Configuration interface to a default set of parameters and the PSSS will change them as needed.
- If an IO_Service implements both interfaces but does not support dynamic re-configuration of parameters at run-time, it will just return the NOT_AVAILABLE error code on the IO_Service-specific API
- Using IO_Service-specific configuration is not limited to the PSSS, the System Integrator can use it too during set-up of the system; however, the usefulness of this is rather limited since the System Integrator can always rely on the FACE::Configuration API for the task

6.4.2.1 Using the FACE::Configuration interface

The FACE::Configuration interface is intended to be extended and implemented by the System Integrator in order to offer a global configuration resource for the end product. The interface abstracts the concrete implementation of the configuration service (hardcoded values by the System Integrator inside the code itself, backed by a locally available configuration file or

received remote from a configuration server) to provide an implementation-independent API to access the configuration parameters.

It is important to notice that this configuration method is in total control of the System Integrator and it is capable to configure the IO_Service independent from the user of the service (PSS). The PSSS code receives via the Injectable Interface a service already configured and ready to use. There is no way for the PSSS to change IO_Service parameters via the FACE::Configuration interface.

A second important observation is that using the FACE::Configuration interface is a “static” configuration method. This means the IO_Service is configured at start-up time, before the PSSS starts to use it and there is no way for the service to be reconfigured at run-time after that. This creates strong stability of the resulting system, since the System Integrator is in charge of building the system, has access to all the information about the IO service that is supposed to be used by the PSSS, and is most capable of configuring it the right way.

The FACE::Configuration interface should therefore be the method of choice whenever possible. The System Integrator will establish all the parameters needed by any particular IOS component using the documentation provided by the IOSS supplier, information which most likely is not available to the PSSS supplier which has to write code to be as portable as possible without coupling it with a particular IOS implementation.

This method is illustrated in the example implementation, Phytia, looking at the way the serial IO_Service is configured in the Main_one.cpp demo:

```
ConfigFile configFile;
configFile.Initialize("config.cfg", return_code);

IOSCLASS serial;
serial.Initialize("CONSOLE", return_code);
serial.Set_Reference("Configurations", configFile, unusedGuid,
return_code);

SupplierC::PSSS1 pss1;

pss1.Set_Reference("SerialPort", serial, guidserialpss1, return_code);
```

6.4.2.2 *Using the I/O Service Configuration API*

While the “static” nature of the FACE::Configuration interface guarantees the best stability of the system and portability of PSSS code, there may be cases when this solution does not satisfy a particular implementation.

One example is when a PSSS needs to open a connection with a modem at the modem standard start-up speed in order to issue the “AT” commands needed to connect to the remote equipment. The equipment may request a different communication speed needed to support the full bandwidth required to communicate at full bandwidth for the requested task.

Another example is IP Disclosure packets sent by Unmanned Air Vehicles using the STANAG-4586 NATO standard, where the Air Vehicle informs the Ground Control Station about the Port Numbers and Multicast Addresses needed by the GCS to communicate with it. Once this packet has been received, the GCS shall change the parameters of the Socket-based IO_Service implementing the communication with the Air Vehicle.

In such a situation, the PSSS shall be able to change at run-time the parameters of the IO_Service and here is where the IO_Service-specific configuration API is needed.

An example of using the IO_Service-specific configuration API can be seen below:

```
FACE::UnsignedLong numOfParams = 6;

FACE::IOSS::Serial::IO_PARAMETER aryParams[numOfParams];
FACE::IOSS::Serial::IO_PARAMETER cnfgParams[numOfParams];

aryParams[0].id = FACE_IOSS_Serial_MODE;
aryParams[0].value.discriminator =
FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT;
aryParams[0].value.values.ushort_value = FACE_IOSS_Serial_RS_232;

aryParams[5].id = FACE_IOSS_Serial_FLOW_CONTROL;
aryParams[5].value.discriminator =
FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
aryParams[5].value.values.ushort_value =FACE_IOSS_Serial_NONE;

FACE::IOSS::Serial::IO_PARAMETER_LIST paramList(aryParams,
numOfParams);
FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE myParams;
myParams.guid = 1234;
myParams.items.append(paramList);
serial.Configure_Bus_Parameters(myHandle, myTimeout, myParams,
return_code);
```

6.4.3 Basic Read/Write Example

Figure 22 shows a PSSS UoC performing basic synchronous read and write operations with one instance of an I/O Service.

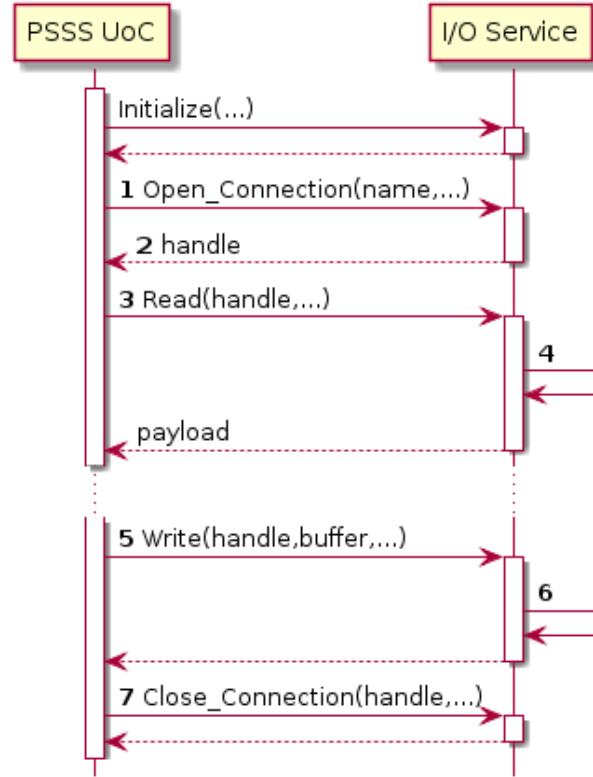


Figure 22: Basic Read/Write Sequence Diagram

1. Opens a connection to a specific device via the I/O Service instance. The specific device is identified by the name parameter. The PSSS UoC can open multiple connections with subsequent calls using different name parameters. The FACE Technical Standard §3.5, Table 6 documents the intended analogy of a connection for each of the supported I/O bus architectures.
2. A unique handle is returned for each connection. The handle is needed for subsequent I/O operations.
3. *Read()* is potentially blocking based on timeout.
4. The I/O Service returns data that may be already buffered, may come directly from a device driver, etc.
5. *Write()* is potentially blocking based on timeout.
6. *Write()* may be implemented such that NO_ERROR return code is not a guarantee the data is written to the device. For example, it may be internally buffered for later device driver interaction. The I/O Service may implement a strategy to communicate device write fail, such as CONNECTION_STATUS_CHANGE_EVENT.
7. After *Close_Connection()*, using the handle for I/O operations results in INVALID_PARAM.

6.4.4 Asynchronous Read Example

Figure 23 shows a PSSS UoC performing basic asynchronous read operations with one instance of an I/O Service. The example begins with a call to *Open_Connection()*, as shown in Section 6.5.1.2.

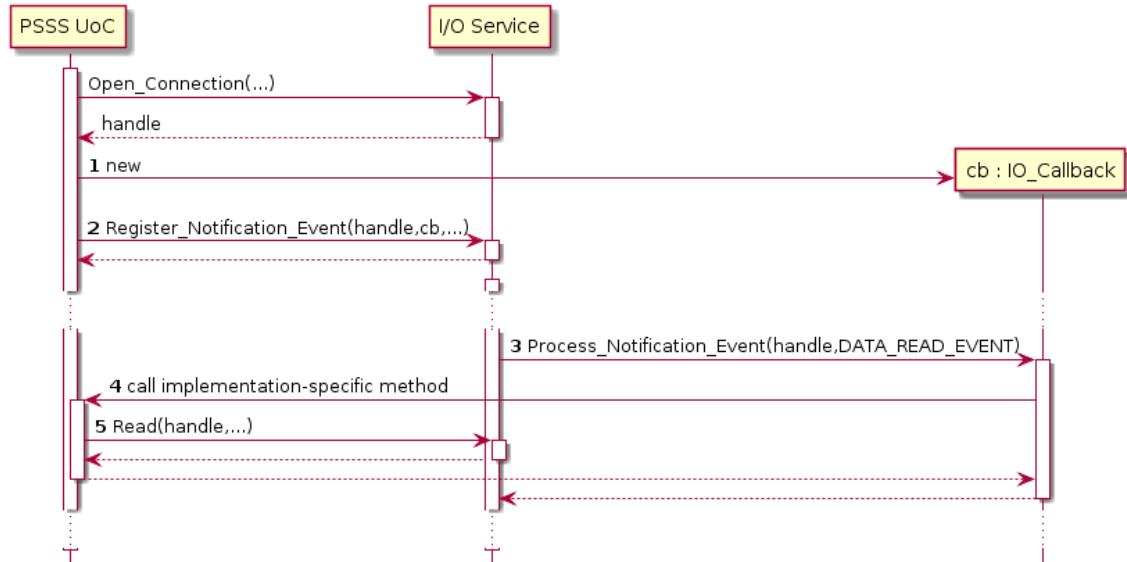


Figure 23: Basic Asynchronous Read Sequence Diagram

1. An instance of *IO_Callback* is created so it can be registered.
2. The callback instance is registered in association with an open handle. The same callback instance can be registered with multiple open handles.
3. At some point later, the I/O Service receives data from the device and invokes the registered callback for the device handle.
4. In this example, the callback has been designed to call some implementation-specific function of the PSSS UoC to read the data.
5. The PSSS UoC reads the data directly from the I/O Service.

Figure 24 shows an alternative implementation where the callback object reads the data from the I/O Service. This suggests the callback object is buffering the data for the PSSS UoC to query as it chooses.

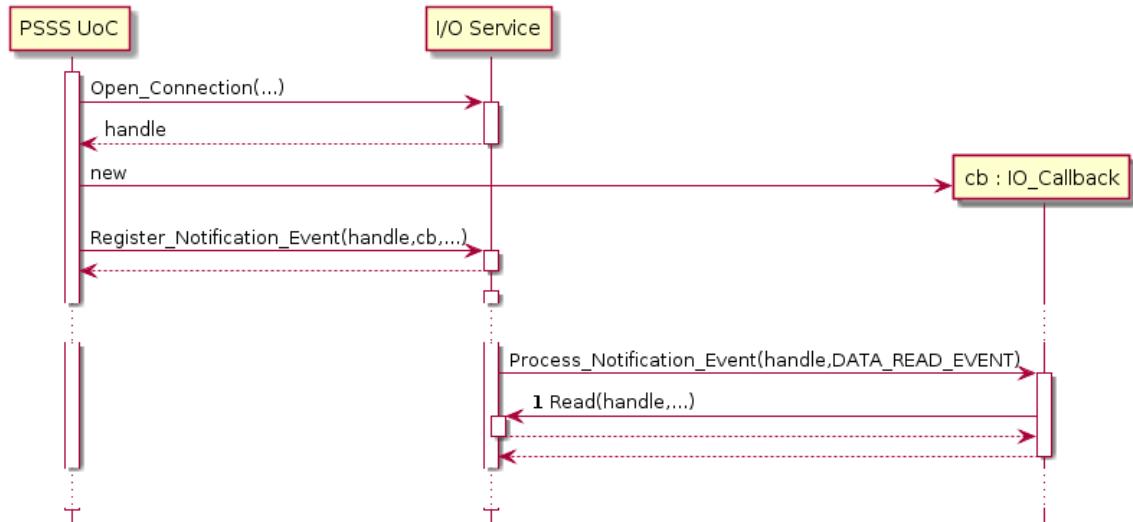


Figure 24: Asynchronous Read with Callback Buffering

The callback object reads the data each time the I/O Service invokes the registered callback for the device handle.

Figure 25 shows another alternative implementation where the callback object sets a flag in the PSSS UoC to indicate that data is available to read. The PSSS UoC reads the data later. This alternative reads the data once `Process_Notification_Event()` has returned in the I/O Service, which may be appropriate to meet I/O performance requirements. The PSSS UoC is not obligated to call `Read()` each time the flag is set. This alternative suggests the I/O Service is managing potential buffer overflow.

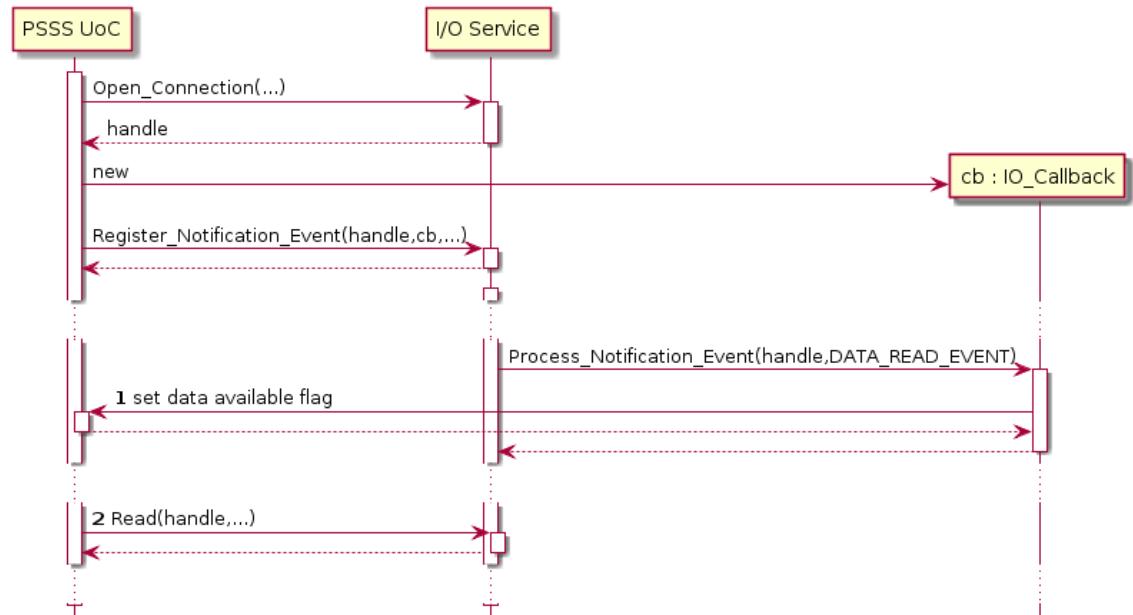


Figure 25: Asynchronous Read with Flag Assignment

In this example, the callback has been designed to call some function of the PSSS UoC to set a flag.

At some point later, the PSSS UoC reads the data directly from the I/O Service.

6.4.5 Connection Parameters Example

Figure 26 shows a PSSS UoC setting/getting connection parameters for an I/O Service. The FACE Technical Standard does not require support for any specific connection parameters, so any I/O service is free to support or not support connection parameters.

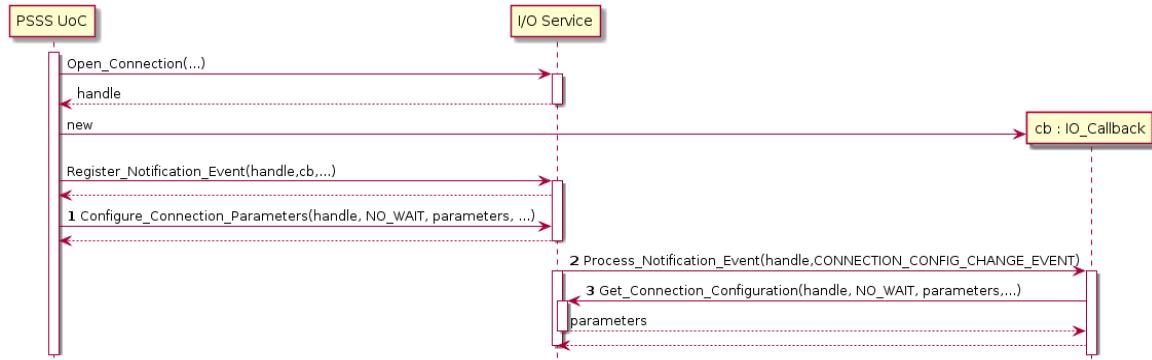


Figure 26: Connection Parameters Sequence Diagram

1. The PSSS calls `Configure_Connection_Parameters()` with a timeout value of `NO_WAIT`. The `parameters` argument contains a sequence of key/value pairs, where the key is a parameter ID and the data type for the value is based on the key. The `parameters` argument also includes an ID for the transaction of type `GUID_TYPE`. This transaction ID is cached by the I/O Service implementation, and is returned via `Get_Connection_Configuration()` so that callers are aware of the last transaction that modified the configuration parameters.
2. In this example, the `CONNECTION_CONFIG_CHANGE_EVENT` is generated, and the registered callback functions is invoked. The `Configure_Connection_Parameters()` call may have also generated `CONNECTION_STATUS_CHANGE_EVENT`, but this is not represented in the above diagram.
3. The callback instance invokes `Get_Connection_Configuration()` on the `IO_Service` with a `parameters` argument that is input/output: the parameter ID is an input; the parameter value is an output, and the transaction ID, of type `GUID_TYPE`, is an output that corresponds to the last invocation of `Configure_Connection_Parameters()`. The callback instance can use the transaction ID to determine if the configuration request was completed successfully. Note that the list of parameter items whose values are being retrieved does not need to be the same list used in the preceding `Configure_Connection_Parameters()` call.

6.4.6 Bus Parameters Example

Figure 27 shows a PSSS UoC assigning then querying configuration parameters for an I/O bus. The I/O bus is accessed via the connection ID returned from `Open_Connection()`. It also illustrates the notification events that may be triggered via a registered notification event handler.

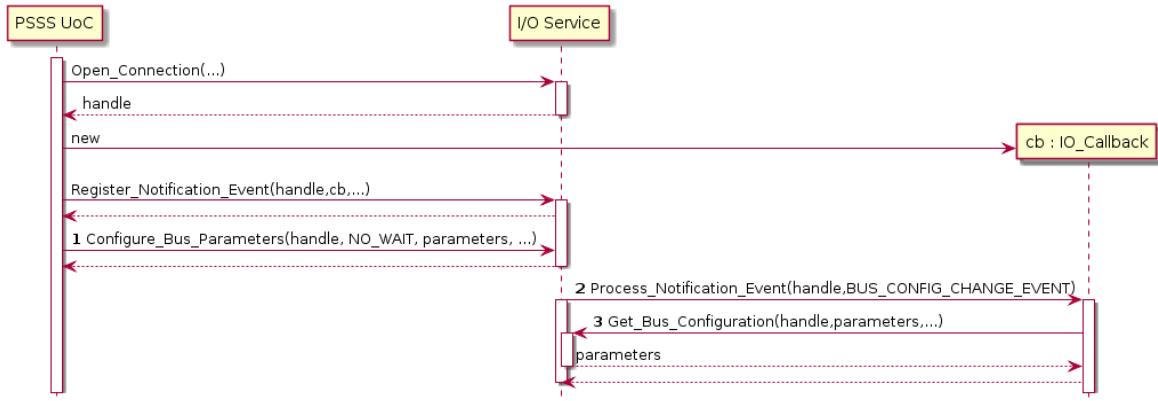


Figure 27: Bus Parameters Sequence Diagram

1. The PSSS calls *Configure_Connection_Parameters()* with a timeout value of `NO_WAIT`. The `parameters` argument contains a sequence of key/value pairs, where the key is a parameter ID and the data type for the value is based on the key. The `parameters` argument also includes an ID for the transaction of type `GUID_TYPE`. This transaction ID is cached by the I/O Service implementation, and is returned via *Get_Connection_Configuration()* so that callers are aware of the last transaction that modified the configuration parameters.
2. In this example, the `BUS_CONFIG_CHANGE_EVENT` is generated, and the registered callback function is invoked. The *Configure_Bus_Parameters()* call may have also generated `BUS_STATUS_CHANGE_EVENT`, `CONNECTION_CONFIG_CHANGE_EVENT`, or `CONNECTION_STATUS_CHANGE_EVENT`, but this is not represented in the above diagram.
3. The callback instance invokes *Get_Bus_Configuration()* on the `IO_Service` with a `parameters` argument that is input/output: the parameter ID is an input; the parameter value is an output, and the transaction ID, of type `GUID_TYPE`, is an output that corresponds to the last invocation of *Configure_Bus_Parameters()*. The callback instance can use the transaction ID to determine if the configuration request was completed successfully.

The following code reflects one option for creating and configuring the bus parameters:

```

MySerial serial;

void Connect_and_Config_MySerial()
{
    FACE::RETURN_CODE_TYPE::Value return_code;
    const FACE::IOSS::Serial::CONNECTION_NAME_TYPE myName =
        "mySerialPort";
    FACE::IOSS::CONNECTION_HANDLE_TYPE myHandle;
    FACE::TIMEOUT_TYPE myTimeout = 0;

    serial.Open_Connection(myName, myTimeout, myHandle, return_code);

    // Create an array of IO_PARAMETER's to configure mySerial
    FACE::UnsignedLong numOfParams = 6;

```

```

FACE::IOSS::Serial::IO_PARAMETER aryParams[numOfParams];

// Setup FACE::IOSS::OPERATIONAL_MODE_TYPE parameter
aryParams[0].id = FACE_IOSS_Serial_MODE;
aryParams[0].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT;
aryParams[0].value.values.ushort_value = FACE_IOSS_Serial_RS_232;

// Setup FACE::IOSS::BAUD_RATE parameter
aryParams[1].id = FACE_IOSS_Serial_BAUD_RATE;
aryParams[1].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE_LONG;
aryParams[1].value.values.ushort_value = 9600;

// Setup FACE::IOSS::DATA_BITS parameter
aryParams[2].id = FACE_IOSS_Serial_DATA_BITS;
aryParams[2].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT;
aryParams[2].value.values.ushort_value = 8;

// Setup FACE::IOSS::STOP_BITS parameter
aryParams[3].id = FACE_IOSS_Serial_STOP_BITS;
aryParams[3].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT;
aryParams[3].value.values.ushort_value = 1;

// Setup FACE::IOSS::PARITY_TYPE parameter
aryParams[4].id = FACE_IOSS_Serial_PARITY;
aryParams[4].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
aryParams[4].value.values.ushort_value =
FACE_IOSS_Serial_PARITY_ODD;

// Setup FACE::IOSS::FLOW_CONTROL parameter
aryParams[5].id = FACE_IOSS_Serial_FLOW_CONTROL;
aryParams[5].value.discriminator =

FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
aryParams[5].value.values.ushort_value =FACE_IOSS_Serial_NONE;

// Create IO_PARAMETER_LIST (Sequence) from array of parameters
FACE::IOSS::Serial::IO_PARAMETER_LIST paramList(aryParams,
numOfParams);

// Create IO_PARAMETER_TRANSACTION_TYPE
FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE myParams;
myParams.guid = 1234;
myParams.items.append(paramList);

//Call Configure
serial.Configure_Bus_Parameters(myHandle, myTimeout, myParams,
return_code);

```

```
}
```

Prior to calling the *Get_Bus_Configuration()* function, an IO_PARAMETER_TRANSACTION_TYPE data structure with the parameter IDs of the relevant configuration values needs to be defined. The definition below is an example of a serial I/O parameter list with only the parameter IDs populated in the data structure. This parameter list is then passed to the *Get_Bus_Configuration()* function where the actual value discriminator and values are populated with the configuration data specified by the parameter IDs. The parameter IDs in the list can be a subset of all the possible bus configuration parameters.

```
MySerial serial;

void Get_MySerial_Config()
{

    FACE::RETURN_CODE_TYPE::Value return_code;
    FACE::IOSS::CONNECTION_HANDLE_TYPE myHandle;
    FACE::TIMEOUT_TYPE myTimeout = 0;

    // Create an array of IO_PARAMETER's for mySerial configuration to
    get populated
    FACE::UnsignedLong numOfParams = 6;
    FACE::IOSS::Serial::IO_PARAMETER cnfgParams[numOfParams];

    //Setup the parameter IDs of the bus configuration parameters that
    you are
    //interested in obtaining
    cnfgParams[0].id = FACE_IOSS_Serial_MODE;
    cnfgParams[1].id = FACE_IOSS_Serial_BAUD_RATE;
    cnfgParams[2].id = FACE_IOSS_Serial_DATA_BITS;
    cnfgParams[3].id = FACE_IOSS_Serial_STOP_BITS;
    cnfgParams[4].id = FACE_IOSS_Serial_PARITY;
    cnfgParams[5].id = FACE_IOSS_Serial_FLOW_CONTROL;

    // Create IO_PARAMETER_LIST (Sequence) from array of parameters
    FACE::IOSS::Serial::IO_PARAMETER_LIST cnfgList(cnfgParams,
    numOfParams);

    // Create IO_PARAMETER_TRANSACTION_TYPE
    FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE myCnfgParams;
    myParams.guid = 1234; //This value needs to be a Unique transaction
    ID
    myCnfgParams.items.append(cnfgList);

    //Call Get_Bus_Configuration
    serial.Get_Bus_Configuration(myHandle, myTimeout, myCnfgParams,
    return_code);
}
```

6.4.7 Device I/O Use Case: Protocol Mediation

Figure 28 shows a PSSS UoC employing Device Protocol Mediation (DPM) to translate from a device protocol to its native protocol. DPM is a Platform-Specific Common Service of the PSSS. DPM is particularly useful when integrating an existing PSSS UoC and an existing IOSS UoC when the protocols do not match.

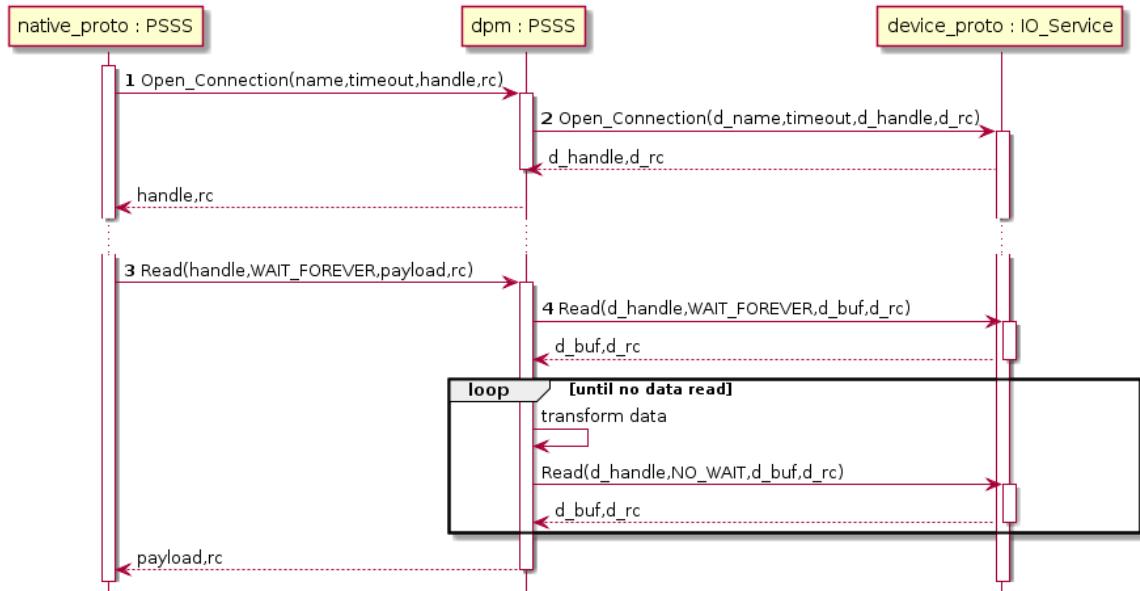


Figure 28: Device Protocol Mediation Use Case Sequence Diagram

1. The PSSS UoC expects to open a connection to an I/O device that understands its native protocol. The connection name parameter has been mapped by the System Integrator to the PSSS UoC implementing DPM.
2. The PSSS UoC implementing DPM opens a corresponding connection to the I/O Service. The connection name parameter has been mapped by the System Integrator to the I/O device. Note the timeout parameter is passed through the DPM to the I/O Service.
3. The PSSS UoC expects to read a data element per its native protocol. Note in this example the timeout parameter from the PSSS UoC is WAIT_FOREVER, so this is a blocking call until a complete protocol message is received.
4. In this example, the DPM reads data in a loop from the I/O Service, transforming as appropriate, until the expected native protocol data is populated. The first *Read()* blocks until data is available, and then the loop reads the remaining data without blocking until all available data from the I/O Service is returned.

6.5 Providing an I/O Service

The examples in this section are based on a working implementation of a Serial I/O Service written in C++. The algorithms are shown to illustrate guidance and are not normative.

The FACE Technical Standard identifies CONNECTION_CLOSED as a return code value to indicate *handle* was previously open but is now closed, distinct from INVALID_PARAM indicating *handle* is unknown. To draw the distinction, the implementation would be required to track handles after they are closed, imposing an unnecessary state management burden. It is recommended that an I/O Service return INVALID_PARAM in both cases. The CONNECTION_CLOSED return code value should be considered deprecated for the *Close_Connection()*, *Read()*, *Write()*, *Configure_Connection_Parameters()*, *Get_Connection_Status()*, and *Register_Nonification_Event()* operations.

6.5.1 Implementing an I/O Service

This section provides an example for implementing an I/O Service named MySerial in C++, but highlights topics that are relevant to implementations of other I/O Services in C, Ada, or Java.

The Serial I/O Service Interface is declared in FACE/IOSS/Serial.idl that, when mapped to C++, defines the namespace *FACE::IOSS::Serial* in the file FACE/IOSS/Serial.hpp that includes a number of structs and the following classes:

1. *FACE::IOSS::Serial::IO_Service* – the interface for the Serial I/O Service functions defined by the FACE Technical Standard.
2. *FACE::IOSS::Serial::IO_Callback* – the interface for the callback handler that supports event-based read operations.

Implementing the example Serial I/O Service involves declaring and defining a concrete class that derives from each, as shown below. The MySerial implementation uses Configuration Services to read configured values, so it supports the Injectable Interface for Configuration Services.

```
#include <FACE/IOSS/Serial.hpp>

class MySerial: public FACE::IOSS::Serial::IO_Service,
                public FACE::Injectable<FACE::Configuration> {
public:
    MySerial();
    virtual ~MySerial();

    // Method signatures will appear here and are detailed below
};

class MySerialCallback: public FACE::IOSS::Serial::IO_Callback {
public:
    MySerialCallback();
    virtual ~MySerialCallback();

    // Method signatures will appear here and are detailed below
};
```

6.5.1.1 Initialize

The *Initialize()* operation is intended to initialize the service, not necessarily the underlying device. For example, a printer service can be initialized with a number of printers available to a user, but the communication and connection to a printer may be deferred until it is time to print a document. The FACE Technical Standard does not specify when device initialization occurs, so there are multiple possible implementations. The MySerial implementation defers device initialization to *Open_Connection()*.

The *Initialize()* operation is an opportunity to read values from a configuration resource. Configured values may support behavior of the I/O Service itself as well as the parameters for one or more connections. Note the PSSS UoC user may be designed to use IOS Interface functions to explicitly configure a bus or connection with values it reads from its own configuration resource.

```

void MySerial::Initialize(
    const FACE::CONFIGURATION_RESOURCE& config_resource,
    FACE::RETURN_CODE_TYPE::Value& return_code) {

    // Assume safe values for class member variables are assigned in the
    // C++ constructor. Assign practical defaults here before reading
    // configured values.
    baudRate=115200; dataBits=8; stopBits=1; parity=0; flowControl=0;
    echo=translateLF=translateEchoCR=false;

    // Check for NO_ACTION, NOT_AVAILABLE, and IN_PROGRESS error
    // conditions.

    // Use Configuration Services to read configured values from
    // 'config_resource'. Perform validity checks on each value before
    // assignment when applicable, returning INVALID_CONFIG if check
    // fails.

    if(configFile != NULL) {
        FACE::Configuration::HANDLE_TYPE mySection;
        FACE::String fullName=configurationResource.buffer();
        fullName.append("_"); fullName.append(name);
        configFile->Open(fullName, mySection, return_code);
        if(return_code == FACE::RETURN_CODE_TYPE::NO_ERROR) {
            char buffer[129]; // assume no config line is larger than 128
            FACE::Configuration::BUFFER_SIZE_TYPE sz;

            configFile->Read(mySection,"deviceName",static_cast<void
*>(buffer),
                               sizeof(buffer), sz,return_code);
            if(return_code==FACE::RETURN_CODE_TYPE::NO_ERROR)
                strncpy(deviceName,buffer,sizeof(deviceName)-1);
            deviceName[sizeof(deviceName)-1]=0;

            configFile->Read(mySection,"baudRate",static_cast<void
*>(buffer),
                               sizeof(buffer),sz,return_code);
            int brate=atoi(buffer);
            if(brate >=50 && brate <=230400) baudRate=brate;

            configFile->Read(mySection,"dataBits",static_cast<void
*>(buffer),
                               sizeof(buffer),sz,return_code);
            int databits=atoi(buffer);
            if(databits==7 || databits==8 ) dataBits=databits;

            configFile->Read(mySection,"stopBits",static_cast<void
*>(buffer),
                               sizeof(buffer),sz,return_code);
            int stopbits=atoi(buffer);
            if(stopbits==1 || stopbits==2 ) stopBits=stopbits;

            configFile->Read(mySection,"parity",static_cast<void *>(buffer),
                               sizeof(buffer),sz,return_code);
            int parbits=atoi(buffer);
            if(parbits>=0 && parbits<=2 ) parity=parbits;
    }
}

```

```

        configFile->Read(mySection,"flowControl",static_cast<void
*>(buffer),
                     sizeof(buffer),sz,return_code);
        int flowbits=atoi(buffer);
        if(flowbits>=0 && flowbits<=2 ) flowControl=flowbits;

        configFile->Read(mySection,"echo",static_cast<void *>(buffer),
                           sizeof(buffer),sz,return_code);
        int echobits=atoi(buffer);
        if(echobits==0 || echobits==1 ) echo=(echobits==1);

        configFile->Read(mySection,"translateLF",static_cast<void
*>(buffer),
                           sizeof(buffer),sz,return_code);
        int lfbits=atoi(buffer);
        if(lfbits==0 || lfbits==1 ) translateLF=(lfbits==1);

        configFile->Read(mySection,"translateEchoCR",static_cast<void
*>(buffer),
                           sizeof(buffer),sz,return_code);
        int crbits=atoi(buffer);
        if(crbits==0 || crbits==1 ) translateEchoCR=(crbits==1);

        configFile->Close(mySection, return_code);

        return_code=FACE::RETURN_CODE_TYPE::NO_ERROR;

    } else fprintf(stderr,
                  "MySerial failed to open section: %s.
return_code=%d\n",
                  name.buffer(),return_code);

    } else return_code=FACE::RETURN_CODE_TYPE::INVALID_CONFIG;
}

```

6.5.1.2 Open_Connection

Opening a connection provides a handle to the caller that allows the caller to perform configuration, callback registration, and read/write operations on the device. Handles are unique identifiers, but no guarantee or assumptions should be made about the handle return value. For example, do not use handles as array indexes, and do not assume a call to open a connection on the same named device will return the same handle. The handle should be considered invalid for any return code besides NO_ERROR.

Just as in initialization, there are multiple options for implementation. This function may establish a connection and assert device readiness, but it is also valid to reserve system resources and defer device readiness to read, write, or callback registration operations. The MySerial implementation opens the serial device and uses a helper function to write configured values read during *Initialize()* to the device.

```

void MySerial::Open_Connection(
    const FACE::IOSS::Serial::CONNECTION_NAME_TYPE& name,
    FACE::TIMEOUT_TYPE timeout,
    FACE::IOSS::CONNECTION_HANDLE_TYPE& handle,
    FACE::RETURN_CODE_TYPE::Value& return_code) {

```

```

    if (isTimeoutOutsideRange(timeout)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    fd_m = open(deviceName, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd_m < 0) {
        fprintf(stderr,"MySerial failed to open serial device: %s.
errno=%d\n",
                deviceName,errno);
        return_code=FACE::RETURN_CODE_TYPE::INVALID_CONFIG;
    } else fprintf(stderr,"DEBUG: Serial port %s opened at speed %d\n",
                    deviceName,baudRate);

    if (! setParametersToPort()) { // Helper method to hide details
        close(fd_m);
        fd_m=-1;
        return_code=FACE::RETURN_CODE_TYPE::INVALID_CONFIG;
    }

    return_code=FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.3 Close_Connection

The MySerial implementation uses helper functions to illustrate the error-checking logic and release of internal resources for the connection, such as file descriptors or buffers.

```

void MySerial::Close_Connection(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::RETURN_CODE_TYPE::Value& return_code) {

    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    if (isHandleUnclosable(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_MODE;
        return;
    }

    if (ReleaseHandleResources(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
        return;
    } else {
        return_code = FACE::RETURN_CODE_TYPE::NO_ACTION;
        return;
    }
}

```

6.5.1.4 *Read*

The *Read()* operation is called to read data from an open connection synchronously, or upon receipt of a notification event into the payload buffer provided. The payload type for this example is unique to the Serial Bus Architecture. Other I/O Services will have their own payload type defined for the respective I/O device in the FACE Technical Standard §C.3. The *Read()* operation has a defined set of possible return values for many possible conditions.

Successful read operations should return NO_ERROR.

The timeout is provided when the method should block for a specified time interval. If a sentinel value of NO_WAIT is provided as a timeout, and no data are available to read, then return NOT_AVAILABLE. The FACE Technical Standard does not specify valid timeout ranges, besides defining the sentinel values. If the timeout specified is not valid for the specific implementation, then INVALID_PARAM is returned. If the sentinel value of INF_TIME_VALUE is provided, then the operation should block indefinitely until data is read or some other underlying concern forces a return code other than TIMED_OUT.

INVALID_PARAM should be returned if the handle provided does not refer to an existing connection.

It is possible that either the service implementation or the underlying device may have dropped data in between read operations. When that happens, return DATA_OVERFLOW, so the caller has an opportunity to respond to missing data. DATA_OVERFLOW should only be returned the first time *Read()* is called after data is dropped.

It is possible that this service has established a remote connection that is lost or closed by the remote side even though *Close()* was never called. Implementations may choose to block for the timeout and return TIMED_OUT or return NO_ACTION in this scenario.

Return CONNECTION_CLOSED if the handle refers does not reference a valid open connection. The connection may have been passed to *Close()* prior to this operation, or the handle may have become otherwise invalid within the IO_Service, and thus can no longer be used for IO_Service operations.

Connections that are not configured for reading (write-only connections) should return INVALID_MODE when *Read()* is called.

6.5.1.5 *Write*

The *Write()* operation is called to write data to an open connection synchronously. The payload type for this example is unique to the Serial Bus Architecture and is defined in FACE/IOS_Serial.idl. Other I/O Services will have their own payload type defined in the Interface Definition Language™ (IDL™) for the respective I/O device in the FACE Technical Standard §C.3. The *Write()* operation has a defined set of possible return values for many possible conditions.

Successful write operations should return NO_ERROR.

The timeout is provided when the method should block for a specified time interval. If a sentinel value of NO_WAIT is provided as a timeout, and the underlying device is currently incapable of processing a write, then return TIMED_OUT. The FACE Technical Standard does not specify valid timeout ranges, besides defining the sentinel values. If the timeout specified is not valid for

the specific implementation, then INVALID_PARAM is returned. If the sentinel value of INF_TIME_VALUE is provided, then the operation should block indefinitely until data is written or some other underlying concern forces a return code other than TIMED_OUT.

INVALID_PARAM should be returned if the handle provided does not refer to an existing connection.

It is possible that either the service implementation or the underlying device may have buffered data in between write operations. When the underlying data buffer is full due to write requests occurring faster than the underlying writes can service, return DATA_OVERFLOW, so the caller has an opportunity to handle the disproportionate write speeds. DATA_OVERFLOW should be returned each time a *Write()* fails due to an underlying buffer overflow.

It is possible that this service has established a remote connection that is lost or closed by the remote side even though the *Close()* operation was never called. Implementations may choose to block for the timeout and return TIMED_OUT or return NO_ACTION in this scenario.

Return CONNECTION_CLOSED if the handle does not reference a valid open connection. The connection may have been passed to *Close()* prior to this operation, or the handle may have become otherwise invalid within the IO_Service, and thus can no longer be used for IO_Service operations.

Connections that are not configured for writing (read-only connections) should return INVALID_MODE when the *Write()* operation is called.

6.5.1.6 Configure_Connection_Parameters

The *Configure_Connection_Parameters()* operation is called to change the current configuration of a connection. The MySerial implementation outlines iteration over the parameter set and sending the event notification.

```
void MySerial::Configure_Connection_Parameters(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::TIMEOUT_TYPE timeout,
    FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
    FACE::RETURN_CODE_TYPE::Value& return_code) {

    FACE::UnsignedLong i;
    FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
    FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue;
    FACE::UnsignedLong numberOfParametersToSet;

    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    if (isTimeoutOutsideRange(timeout)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
}
```

```

}

if ((numberOfParametersToSet = parameters.items.length()) < 1) {
    // No parameters to be set
    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
    return;
}

// Loop over each item in parameters to verify that each item has a
// recognized ID and a legal value
for (i=0; i<numberOfParametersToSet; i++) {
    ioParameterId = parameters.items[i].id;
    ioParameterValue = parameters.items[i].value;

    switch (ioParameterId) {
        case FACE_IOSS_Serial_CHANNEL_NUM:
            // If ioParameterValue is not a legal value for the channel
            // - set return_code to INVALID_PARAM
            // - return
            break;

        case FACE_IOSS_Serial_MODE:
            // If ioParameterValue is not a legal value for the serial
            mode
            // - set return_code to INVALID_PARAM
            // - return
            break;

        // . .

        default:
            // Unrecognized parameter ID
            return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
            return;
    }
}

// Loop over each item in parameters to set each parameter
for (i=0; i<numberOfParametersToSet; i++) {
    ioParameterId = parameters.items[i].id;
    ioParameterValue = parameters.items[i].value;

    switch (ioParameterId) {
        case FACE_IOSS_Serial_CHANNEL_NUM:
            // Set the channel number to ioParameterValue
            break;

        case FACE_IOSS_Serial_MODE:
            // Set the serial mode to ioParameterValue
            break;

        // . .
    }
}

MyCallback->Process_Notification_Event(handle,

```

```

FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE::CONNECTION_CONFIG_CHANGE_EVENT);

    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.7 Get_Connection_Configuration

The *Get_Connection_Configuration()* operation is called to get the current configuration of a connection. The calling function must provide a list of parameters that is populated with the specific keys for which the connection configuration values are requested. The MySerial implementation outlines populating the returned parameter set.

```

void MySerial::Get_Connection_Configuration (
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::TIMEOUT_TYPE timeout,
    FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
    FACE::RETURN_CODE_TYPE::Value& return_code) {

    FACE::UnsignedLong i;
    FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
    FACE::UnsignedLong numberParametersToGet;

    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    if (isTimeoutOutsideRange(timeout)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    if ((numberParametersToGet = parameters.items.length()) < 1) {
        // No parameters to get
        return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
        return;
    }

    // Loop over each item in parameters
    for (i=0; i<numberParametersToGet; i++) {
        ioParameterId = parameters.items[i].id;

        switch (ioParameterId) {
            case FACE_IOSS_Serial_CHANNEL_NUM:
                // Set parameters.items[i].value to the current channel number
                break;

            case FACE_IOSS_Serial_MODE:
                // Set parameters.items[i].value to the current serial mode
                break;
        }
    }
}

```

```

    // . . .

    default:
        // Unrecognized parameter ID
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
}

return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.8 Configure_Bus_Parameters

The *Configure_Bus_Parameters()* operation is called to change the current configuration of a bus.

The example code queries the current connection configuration, the connection status, and the bus status. It then applies the changes requested, and then issues notifications to registered callbacks upon change.

A BUS_CONFIG_CHANGE_EVENT is always generated upon success and issued to all open connections. A BUS_STATUS_CHANGE_EVENT is generated and issued to all open connections if the bus status changes. If the bus configuration altered a connection parameter or the connection status, then CONNECTION_CONFIG_CHANGE_EVENT or CONNECTION_STATUS_CHANGE_EVENT, respectively, are issued. A more robust implementation would include error checking and rollback parameters if any of the parameter CRs fails to maintain the transactional nature of the interface.

```

void MySerial::Configure_Bus_Parameters (
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::TIMEOUT_TYPE timeout,
    const FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{

    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    if (isTimeoutOutsideRange(timeout)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    FACE::UnsignedLong i;
    FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
    FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue;
    FACE::UnsignedLong numberOfParametersToSet =
parameters.items.length();

```

```

    unsigned int local_config_variable; // Assume that configuration
data is
                                // packed in a single integer
    FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE
original_config_params,
                                new_config_params;
    FACE::IOSS::BUS_STATUS_TYPE original_bus_status, new_bus_status;
    FACE::IOSS::Serial::CONNECTION_STATUS_TYPE original_connect,
new_connect;
    FACE::RETURN_CODE_TYPE::Value config_return_code,
                                bus_status_return_code,
                                status_return_code;

    if (numberOfParametersToSet < 1) {
        // No parameters to be set
        return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
        return;
    }

    // The I/O Service processes all I/O parameters as one transaction
that
    // succeeds or fails. If the transaction fails, the I/O parameter
values are
    // unchanged upon return.
    Get_Connection_Configuration (handle, timeout,
original_config_params,
                                config_return_code);
    Get_Bus_Status (handle, original_bus_status, bus_status_return_code);
    Get_Connection_Status (handle, original_connect, status_return_code);

    // Loop over each item in parameters
    for (i=0; i<numberOfParametersToSet; i++) {
        ioParameterId = parameters.items[i].id;
        ioParameterValue = parameters.items[i].value;

        switch (ioParameterId) {

            case FACE_IOSS_Serial_MODE:
                // If ioParameterValue is a legal value for the serial port,
set mode
                // in local_config_variable to that value. Otherwise, set
return_code
                // to INVALID_PARAM and return.
                break;

            case FACE_IOSS_Serial_BAUD_RATE:
                // If ioParameterValue is a legal value for the serial port,
set the
                // Baud Rate in local_config_variable to that value. Otherwise,
set
                // return_code to INVALID_PARAM and return.
                break;

            // . . .

            case FACE_IOSS_Serial_FLOW_CONTROL:

```

```

        // If ioParameterValue is a legal value for the serial port,
set the
            // Flow Control in local_config_variable to that value.
Otherwise, set
            // return_code to INVALID_PARAM and return.
            break;

        default:
            // Unrecognized parameter ID
            return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
            return;
    }

}

// Write local_config_variable to serial port

if (original_connect.connection_status !=
    FACE::IOSS::Serial::IO_CONNECTION_STATUS_TYPE::NOT_OPEN) {
    // Dispatch BUS_CONFIG_CHANGE_EVENT to all with a registered
notification
    // callback
    MyCallback->Process_Notification_Event (handle,
FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE::BUS_CONFIG_CHANGE_EVENT);
}

Get_Bus_Status (handle, original_bus_status, status_return_code );
if (original_bus_status != new_bus_status) {
    // Dispatch BUS_STATUS_CHANGE_EVENT to all with a registered
notification
    // callback
    MyCallback->Process_Notification_Event (handle,
FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE::BUS_STATUS_CHANGE_EVENT);
}

Get_Connection_Configuration (handle, timeout, new_config_params,
                             config_return_code);
// Compare connection parameters to determine if they changed.
if (!ParamsMatch (original_config_params, new_config_params)) {
    // Dispatch CONNECTION_CONFIG_CHANGE_EVENT to all with a registered
    // notification callback
    MyCallback->Process_Notification_Event (handle,
FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE::CONNECTION_CONFIG_CHANGE_E
VENT);
}

Get_Connection_Status (handle, new_connect, status_return_code );
// Compare connection status to determine if this changed.
if (original_connect.connection_status !=
new_connect.connection_status) {
    // Dispatch CONNECTION_STATUS_CHANGE_EVENT to all with a registered
    // notification callback
    MyCallback->Process_Notification_Event (handle,
FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE::CONNECTION_STATUS_CHANGE_E
VENT);
}

```

```

        return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
    }

```

6.5.1.9 *Get_Bus_Configuration*

The *Get_Bus_Configuration()* operation is called to get the current configuration of a bus. The calling function must provide a list of parameters that is populated with the specific keys for which the connection configuration values are requested.

```

// The Get_Bus_Configuration() function below returns the bus
configuration
// parameters based on the parameter IDs input into the routine.

void MySerial::Get_Bus_Configuration (
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::TIMEOUT_TYPE timeout,
    FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{

    FACE::UnsignedLong i;

    //Assume that configuration data is packed in a single integer
    unsigned int local_config_variable;
    short Serial_Mode;
    unsigned short Flow_Control;
    unsigned short Parity;
    FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
    FACE::UnsignedLong numberofParametersToSet =
parameters.items.length();

    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    if (isTimeoutOutsideRange(timeout)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    //Read local_config_variable from the serial port specified by the
    handle
    //If unable to read the data within the specified timeout
    // - set return_code to TIMED_OUT
    // - return

    for (i=0; i<numberofParametersToSet; i++) {
        ioParameterId = parameters.items[i].id;

        switch (ioParameterId) {

```

```

        case FACE_IOSS_Serial_MODE:
            /*Serial_Mode = Mask local_config_variable Operational Mode
bits
            if ((Serial_Mode == FACE_IOSS_Serial_RS_232) ||
                (Serial_Mode == FACE_IOSS_Serial_RS_422) ||
                (Serial_Mode == FACE_IOSS_Serial_RS_485)) {
                parameters.items[i].value.ushort_value(Serial_Mode);
            } else {
                return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
                return;
            }
            break;

        case FACE_IOSS_Serial_FLOW_CONTROL:
            /*Flow_Control = Mask local_config_variable Flow Control bits
            if ((Flow_Control == FACE_IOSS_Serial_NONE) ||
                (Flow_Control == FACE_IOSS_Serial_XON_XOFF) ||
                (Flow_Control == FACE_IOSS_Serial_RTS_CTS) ||
                (Flow_Control == FACE_IOSS_Serial_DSR_DTR)) {
                parameters.items[i].value.ushort_value(Flow_Control);
            } else {
                return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
                return;
            }
            break;

        case FACE_IOSS_Serial_PARITY:
            /*Parity = Mask local_config_variable Parity bits
            if ((Parity == FACE_IOSS_Serial_PARITY_NONE) ||
                (Parity == FACE_IOSS_Serial_PARITY_ODD) ||
                (Parity == FACE_IOSS_Serial_PARITY_EVEN) ||
                (Parity == FACE_IOSS_Serial_PARITY_MARK) ||
                (Parity == FACE_IOSS_Serial_PARITY_SPACE)) {
                parameters.items[i].value.ushort_value(Parity);
            } else {
                return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
                return;
            }
            break;

        // ...

        default:
            /* Unrecognized parameter ID
            return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
            return;
    }
}

return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.10 *Get_Connection_Status*

The MySerial implementation uses helper functions to illustrate the error-checking logic and status query for the connection represented by *handle*.

```

void IO_Service_MySerial::Get_Connection_Status(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::IOSS::Serial::CONNECTION_STATUS_TYPE& status,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{
    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }
    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */
    status = QueryConnectionStatus(handle);

    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.11 *Get_Bus_Status*

The MySerial implementation uses helper functions to illustrate the error-checking logic and status query for the bus corresponding to the connection represented by *handle*.

```

void IO_Service_MySerial::Get_Bus_Status(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::IOSS::Serial::BUS_STATUS_TYPE& bus_status_value,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{
    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    bus_status_value = QueryBusStatus(handle);

    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.12 *Register_Notification_Event*

The MySerial implementation uses helper functions to illustrate the error-checking logic and association of callback to handle. Note that since a callback can be registered per handle some type of lookup must be maintained.

```

void IO_Service_MySerial::Register_Notification_Event(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::IOSS::Serial::IO_Callback& io_callback,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{
    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

```

```

    }

    /* Remove this error trap per recommendation
    if (isHandleValidButClosed(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::CONNECTION_CLOSED;
        return;
    }
    */

    AssociateCallbackToHandle(handle, io_callback);

    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.1.13 *Unregister_Nonification_Event*

The MySerial implementation uses helper functions to illustrate the error-checking logic and disassociation of callback from handle. Note that since a callback can be registered per handle some type of lookup must be maintained.

```

void IO_Service_MySerial::Unregister_Notification_Event(
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
    FACE::RETURN_CODE_TYPE::Value& return_code)
{
    if (isHandleUnknown(handle)) {
        return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
        return;
    }

    DisassociateCallbackFromHandle(handle);

    return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
}

```

6.5.2 Developing a New I/O Service Bus Architecture

The I/O bus types specified in the FACE Technical Standard are not intended to be exhaustive. The standard takes a template-based approach to defining I/O Services that permits the introduction of new I/O bus architectures. To extend the standard, no modifications should be made to the existing IDL. Instead, a new bus-specific IDL file should be created and submitted using the FACE PR/CR process for potential inclusion in the Technical Standard as required during Conformance Verification. The following provides an example bus “XYZ” IDL that is used to read and write a set of characters and is configurable for multiple dialects. A similar IDL file for the new bus would be created defining the bus payload, bus configuration, and bus status as needed for the newly defined bus.

```

FACE_IOS_XYZ.idl
// Source file: FACE_IOS_XYZ.idl

#ifndef __FACE_IOS_XYZ
#define __FACE_IOS_XYZ

#include <FACE_IOS.idl>

module FACE {
    module IOSS {

```

```

// Declarations for the XYZ I/O Service.
module XYZ {
    // Declarations for the buffer that becomes part of the 'payload'
    // parameter for the IO_Service::Read and IO_Service::Write
    // operations.
    const unsigned short MAX_BYTE_COUNT = 10;
    typedef sequence<char, MAX_BYTE_COUNT> XYZ_BUFFER_TYPE;

    struct ReadWriteBuffer {
        XYZ_BUFFER_TYPE char_payload;
    };

    // Declarations for the defined configuration parameters of the
    // I/O Service. For each ID_PARAMETER_ID_TYPE, there is a comment
    // for the expected corresponding ID_PARAMETER_VALUE_TYPE.

    typedef short OPERATIONAL_MODE_TYPE;
    const OPERATIONAL_MODE_TYPE ENGLISH = 0;
    const OPERATIONAL_MODE_TYPE SPANISH= 1;

    // Declarations for the defined bus status types for the I/O
    // Service.
    const BUS_STATUS_TYPE HW_BUSY = 0;
    const BUS_STATUS_TYPE HW_OVERFLOW = 1;
    const BUS_STATUS_TYPE UNKNOWN_ERROR = 2;
};

// Instantiate the template module into the namespace for the I/O
// Service. This results in fully-qualified types in that namespace
// distinct to the I/O Service.
module IO_Service_Module<XYZ::ReadWriteBuffer> XYZ;
};

#endif // __FACE_IOS_XYZ

```

6.5.3 Distributed Dispatch of I/O Services Interface Functions

This use case explores the capability of I/O Services to transparently act as a proxy for I/O resources located outside the boundaries of the UoC partition or even local machine. This scenario levies the following requirements:

- The proxy I/O Service must respect the IOS Interface
- The proxy I/O Service must not transform the data read from the device in any other way than for the purpose of hiding the network-related functionality.

6.5.3.1 Simple Implementation – Server not Conformant to the FACE Technical Standard

The simplest design is to implement the I/O Service connecting to a server that is not conformant to the FACE Technical Standard. This server performs the remote operations on the hardware. Network/IPC calls are available via the OS Interface to the proxy IOSS UoC. The fact that the remote server can be a separate application, not conformant to the FACE Technical Standard, does not preclude the proxy IOSS FACE UoC from being FACE conformant, because it still uses only allowed OS Interface calls and it provides the IOS Interface.

In this scenario, seen in Figure 29 as an example, the real functionality of the I/O Service is being performed on a remote machine. The device marked in red shows that the hardware and drivers are not available on the local machine, but is instead implemented remotely. The System Integrator develops a proxy I/O Service implemented to communicate to a remote device driver accessible either over an Inter-Process (Inter-Partition) Communication mechanism or over the network with remote machines. The proxy I/O Service provides the IOS Interface, making the existence of the remote driver totally transparent to the PSSS UoC.

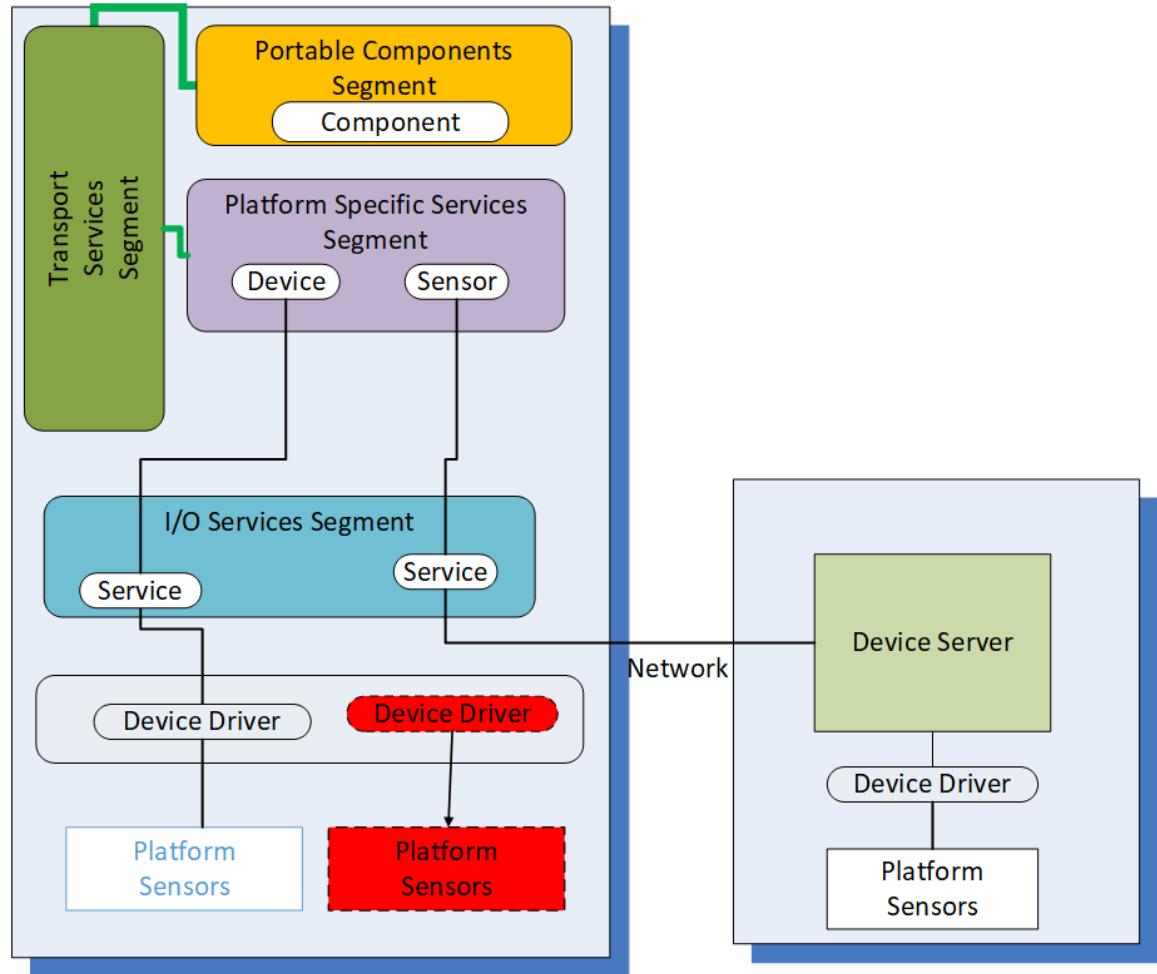


Figure 29: Distributed Dispatch to a FACE Non-Conformant Server

The proxy I/O Service is implemented as a network client for a server that in turn accesses the device on the remote machine. All methods defined by the I/O Service will be implemented remotely, packing the IN parameters and sending them to the server where the actual method implementation is being called. Any received responses are parsed out to populate the OUT parameters of the method.

The *Initialize(I/O)* method can make sure the remote service exists before returning no error. As an example, it can establish a network connection with the server but no serial port is opened at the moment. Alternatively, this method can just initialize local resources, read configuration files, and the whole connection to the server can be deferred to *Open_Connection(I/O)*. That way it may even be possible to have different connections happen on different servers, if each connection handle manages its own sockets.

Open_Connection(I/O) sends to the server the request for the port to be opened. The server may have its own mapping table into a configuration file and redirect the request to the appropriate port. For example, the request for COM1 on the client side can trigger the server to open /dev/ttyEQ7 on a multiport bus. The server replies with a message specifying the status of connection and then maps the connection to an I/O connection handle upon receiving a success reply.

Close_Connection(I/O) sends a request to the server to close the port and free any associated resources with it. Upon receiving a successful confirmation, the local class also frees any allocated resources in conjunction with that port.

Read(I/O) sends a request to the server to read data. The data received from the server is copied to the *payload* parameter.

Write(I/O) sends a request to the server to write data from the *payload* parameter and waits for a response on the write operation from the server before returning.

The behavior of these methods depends on whether the client IOSS UoC implementation is caching the relevant data; *Configure_Connection_Parameters(I/O)*, *Get_Connection_Configuration(I/O)*, *Configure_Bus_Parameters(I/O)*, *Get_Bus_Configuration(I/O)*, *Get_Connection_Status(I/O)*, or *Get_Bus_Status(I/O)*.

Register_Notification_Event(I/O) would have a distributed implementation. The client IOSS UoC implementation could start a thread able to receive notifications from the server. To achieve notifications, an implementation can use various design options:

- All I/O and socket management will happen within the context of a thread started by the *Initialize(I/O)* method
When any of the above methods are called, the data is passed to the thread and then the calling thread waits on a condition variable to be signaled when data is received. On reply, the working threads receive data from the socket and unlock the calling method via the condition variable.
- A second option may be using a separate socket for signaling; for example, all data transfer can happen on a TCP socket while a secondary UDP socket can be used to receive signals

Regardless of what method is used for implementation, the client IOSS UoC sends to the server a notification message to inform the server that a notification packet shall be sent whenever new data is available.

Unregister_Notification_Event(I/O) sends a cancelation message for notification and if no more notifications are registered locally and if the thread has been started by *Register_Notification_Event(I/O)*, then the thread is terminated.

6.5.3.2 TSS Based Implementation – FACE Server

An alternative design, shown in Figure 30, is a proxy I/O Service which interfaces with a new Device Client PSSS UoC. The Device Client in turn uses TSS to communicate with a corresponding Device Client implemented into a remote FACE Computing Environment.

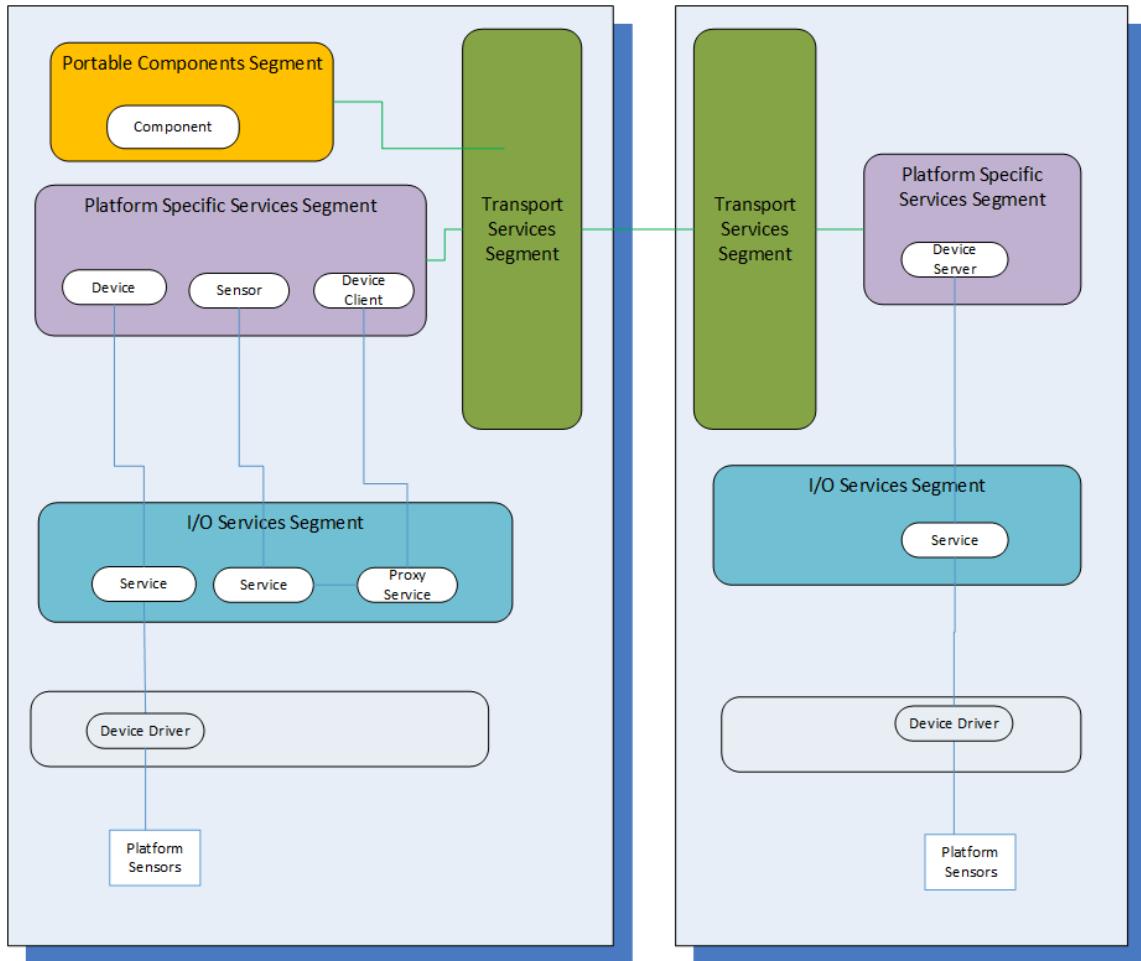


Figure 30: Distributed Dispatch to Remote PSSS UoC

The request for each of the method calls (*Read/Write/Configure/Get_Status()*, etc.) must be serialized by Proxy Service into a data-packet available by the *Read()* method of the Proxy. The Client will then pack the result of the Read into a TSS packet to be sent over TSS toward the Device Server located on the remote process where the physical device is available. The Device Server has to de-serialize the received data in order to call the corresponding methods.

While this alternative implementation appears at a superficial glance to be more aligned with FACE goals than an option that does not utilize the FACE Technical Standard, there are some caveats:

- There is a strong coupling between the implementation of Device Proxy and Device Server on the remote machine
We are not gaining any portability/interoperability over the server approach that does not utilize the FACE Technical Standard.
- Since the serialization of the commands is specific to the Proxy at the IOS layer there is little useful data modeling that can be done, so the benefit of data modeling the remote communication is marginal at best

6.5.4 IOSS Extending Parameters for an I/O Service

The I/O Services Interface supports capabilities to configure an I/O device and an I/O device bus. It is designed to support an extensible set of I/O parameters for configuration, where the I/O parameters declared by the FACE Technical Standard can be complemented with additional I/O parameters to meet the requirements of the I/O device or the I/O device bus in a conformant manner.

An I/O parameter is represented as a unique identifier and an associated value by the type *FACE::IOSS::IO_PARAMETER*. A constant of type *FACE::IOSS::IO_PARAMETER_ID_TYPE* represents the unique identifier value for a specific I/O parameter. The type *FACE::IOSS::IO_PARAMETER_VALUE_TYPE* represents the I/O parameter value as a discriminated union for any of the basic integer, floating point, or character types.

To declare an extended I/O parameter, first declare a constant of type *FACE::IOSS::IO_PARAMETER_ID_TYPE* for the unique identifier with a value that does not conflict with other I/O parameters for the same I/O bus architecture. Then, identify the corresponding union discriminator for the I/O parameter value.

The examples below illustrate this using the IOS_ARINC429 I/O interface. In the first example, the developer is building an interface for their XYZ_ARINC429 device. The ARINC429 interface itself provides three default configuration parameters: Parity, Direction, and Speed. The developer will implement these existing parameters, but they also need the ability to specify a new TimeOut parameter for their specific implementation.

A portion of the developer's new XYZ_IOS_ARINC429.hpp header file is shown below. The developer has defined the new parameter of *FACE_IOSS_ARINC429_TIMEOUT_MILLISECONDS* with an *IO_PARAMETER_ID_TYPE* value of 3. In this case, 3 is the next logically available parameter id (0, 1, and 2 are using by Parity, Direction, and Speed). Also provided in this header for the developer's convenience are definitions for the Direction and Speed values.

```
// XYZ_IOS_ARINC429.hpp

#include "IOS_ARINC429.hpp"

// Direction Parameter Definitions
#define FACE_IOSS_ARINC429_DIRECTION_RECEIVE 0x00 // Rx
#define FACE_IOSS_ARINC429_DIRECTION_TRANSMIT 0x01 // Tx

// Speed Parameter Definitions
#define FACE_IOSS_ARINC429_SPEED_LOW           0x00 // 12.5 kbits/sec
#define FACE_IOSS_ARINC429_SPEED_HIGH          0x01 // 100 kbits/sec

// Timeout Parameter Definitions
#define FACE_IOSS_ARINC429_TIMEOUT_MILLISECONDS \
((FACE::IOSS::IO_PARAMETER_ID_TYPE) 3)
```

The second example below shows how the developer would build and specify parameters for a *Configure_Connection_Parameters()* method call for a *XYZ_IOS_ARINC429* object. Note how each of the four parameters (Parity, Direction, Speed, and TimeOut) are constructed and then assembled into a *FACE::Sequence*. The *FACE::Sequence* is then set in an

IO_PARAMETER_TRANSACTION_TYPE which is then passed as a single parameter to the *Configure_Connection_Parameters()* method.

```
// PSSS_XYZ_ARINC429_Example.cpp

#include "XYZ_ARINC429.hpp"

void testXYZ_ARINC429()
{
    FACE::IOSS::ARINC429::XYZ_ARINC429 myARINC429;

    // Create an array of IO_PARAMETER's to configure myARINC429
    FACE::UnsignedLong numOfParams = 4;
    FACE::IOSS::ARINC429::IO_PARAMETER aryParams[numOfParams];

    // Setup FACE_IOSS_ARINC429_DIRECTION parameter
    aryParams[0].id = FACE_IOSS_ARINC429_DIRECTION;
    aryParams[0].value.discriminator =
FACE::IOSS::ARINC429::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
    aryParams[0].value.values.ushort_value =
FACE_IOSS_ARINC429_DIRECTION_TRANSMIT;

    // Setup FACE_IOSS_ARINC429_PARITY parameter
    aryParams[1].id = FACE_IOSS_ARINC429_PARITY;
    aryParams[1].value.discriminator =
FACE::IOSS::ARINC429::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
    aryParams[1].value.values.ushort_value =
FACE_IOSS_ARINC429_PARITY_EVEN;

    // Setup FACE_IOSS_ARINC429_CHANNEL_SPEED parameter
    aryParams[2].id = FACE_IOSS_ARINC429_CHANNEL_SPEED;
    aryParams[2].value.discriminator =
FACE::IOSS::ARINC429::IO_PARAMETER_VALUE_TYPES_TYPE::FACE USHORT;
    aryParams[2].value.values.ushort_value =
FACE_IOSS_ARINC429_SPEED_HIGH;

    // Setup FACE_IOSS_ARINC429_TIMEOUT parameter
    aryParams[3].id = FACE_IOSS_ARINC429_TIMEOUT_MILLISECONDS;
    aryParams[3].value.discriminator =
FACE::IOSS::ARINC429::IO_PARAMETER_VALUE_TYPES_TYPE::FACE LONGLONG;
    aryParams[3].value.values.longlong_value = 1000;

    // Create IO_PARAMETER_LIST (Sequence) from array of parameters
    FACE::IOSS::ARINC429::IO_PARAMETER_LIST paramList(aryParams,
numOfParams);

    // Create IO_PARAMETER_TRANSACTION_TYPE
    FACE::IOSS::ARINC429::IO_PARAMETER_TRANSACTION_TYPE myParams;
    myParams.guid = 1234;
    myParams.items.append(paramList);

    // Call Generic.Configure_Connection_Parameters
    myARINC429.Configure_Connection_Parameters(handle, timeout,
myParams, return_code);
}
```

This flexible parameter mechanism can be used to add any number of additional parameters to the I/O Service Configuration setting/getting functions.

6.5.5 Realizing an I/O Service using LCM Services

This section provides guidance for implementing an I/O Service that also provides one or more LCM Services interfaces: Initializable, Configurable, Connectable, and Stateful. LCM Services support management of software components at several specific execution points, as described in the FACE Technical Standard §3.13. General guidance for LCM Services is described in Chapter 9, so this section focuses on topics of particular concern for an I/O Service. The example in this section implements all four LCM Services interfaces for a Generic I/O Service.

6.5.5.1 Declarations for a Managed I/O Service

This example declaration extends the one shown in Section 9.5.1 for the Generic I/O Service. For simplicity, it uses the same state representation and leverages *ExampleManagedComponent* as a base class.

```
// Source file: RIG3/IOSS/ManagedGenericIO.idl

#ifndef __RIG3_IOSS_MANAGEDGENERICIO
#define __RIG3_IOSS_MANAGEDGENERICIO

#include <FACE/IOSS/Generic.idl>
#include <RIG3/LCM/ExampleManagedComponent.idl>

module RIG3 {
    module IOSS {
        // Use IDL multiple inheritance to declare the example I/O Service.
        interface ManagedGenericIO :
            FACE::IOSS::Generic::IO_Service,
            RIG3::LCM::ExampleManagedComponent {
        }; // interface ManagedGenericIO
    }; // module IOSS
}; // module RIG3

#endif //! __RIG3_IOSS_MANAGEDGENERICIO
```

Mapping the IDL for *RIG3::IOSS::ManagedGenericIO* to C++ results in the class *RIG3::IOSS::ManagedGenericIO* as an abstract base class. A concrete class that implements the pure virtual methods is needed, as shown below in the declaration of *RIG3::IOSS::MyManagedGenericIO*. Note that for the purpose of this example the pure virtual methods inherited from *FACE::IOSS::Generic::IO_Service* are not shown.

```
// Source file: RIG3/IOSS/MyManagedGenericIO.hpp

#include <RIG3/IOSS/ManagedGenericIO.hpp>

namespace RIG3 {
    namespace IOSS {
        class MyManagedGenericIO : public ManagedGenericIO {
            // Generic I/O Service member functions not shown in example

            // from FACE::LCM::Initializable::InitializableInstance
            virtual void Initialize(
                FACE::RETURN_CODE_TYPE::Value &return_code);
    };
}
```

```

        virtual void Finalize(
            FACE::RETURN_CODE_TYPE::Value &return_code);

        // from FACE::LCM::Configurable::ConfigurableInstance
        virtual void Configure(
            const FACE::CONFIGURATION_RESOURCE &configuration,
            FACE::RETURN_CODE_TYPE::Value &return_code);

        // from FACE::LCM::Connectable::ConnectableInstance
        virtual void Framework_Connect(
            const FACE::CONFIGURATION_RESOURCE &configuration,
            FACE::RETURN_CODE_TYPE::Value &return_code);
        virtual void Framework_Disconnect(
            FACE::RETURN_CODE_TYPE::Value &return_code);

        // from ExampleStateful::StatefulInstance
        virtual void Query_State(
            FACERIG::IOS::EXAMPLE_STATE::Value &current_state,
            FACE::RETURN_CODE_TYPE::Value &return_code);
        virtual void Request_State_Transition(
            const FACERIG::IOS::EXAMPLE_STATE::Value &current_state,
            FACE::RETURN_CODE_TYPE::Value &return_code);
    };
} // namespace IOSS
} // namespace RIG3

```

6.5.5.2 Definitions for a Managed I/O Service

In general, the behavior appropriate for the definitions of the virtual member functions for the LCM Services interfaces is highly dependent on the software component. Recall that LCM Services provide the opportunity for a software component to behave at particular execution points.

Note that the purpose of the *Initialize()* operation for LCM Service Initializable Interface, to initialize a software component instance, complements the purpose of the *Initialize()* operation for an I/O Service, to initialize an instance of the I/O Service. A Managed I/O Service can support deployments where integration software is responsible for software component initialization and deployments where a PSSS UoC is responsible for I/O Service initialization. Figure 31 illustrates the behavior when both approaches are used, where the Managed I/O Service reports to the PSSS UoC that it has already been initialized.

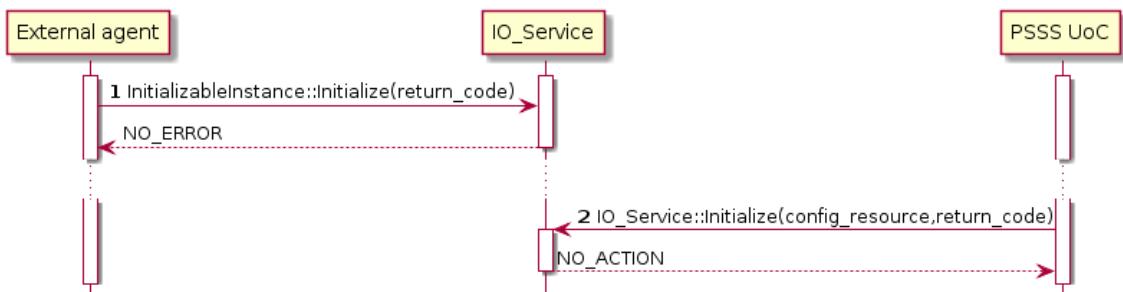


Figure 31: Initializing a Managed I/O Service

An I/O Service that provides the Initializable Interface allows integration software to handle that responsibility so that the PSSS UoC need not.

That same I/O Service can also support a PSSS UoC that does explicitly call *Initialize()* by returning NO_ACTION to indicate it has already been initialized.

7 Transport Services Segment

7.1 Purpose

The purpose of this chapter is to provide additional implementation guidance for Transport Services Segment (TSS) providers.

7.2 Introduction

The TSS abstracts data access and access to common technical functions and facilitates integration of PCS and PSSS software components into disparate architectures and platforms. TSS UoCs support data access to a variety of common technical services. The TSS UoCs provide these common technical services to a PSSS or PCS UoC through the TSS Inter-segment interfaces: Transport Services (TS) APIs (Base and TypedTS) and the Component State Persistence (CSP) API. TSS UoCs provide data transport between PCS/PSSS UoCs, as well as mediation between messages with different UoP Supplied Models (USMs). TSS UoCs support a variety of data transport protocols and messaging patterns. Additionally, TSS UoCs provide access to persistent Data Stores and the ability for UoCs to checkpoint their internal state. The TSS includes support for the capabilities identified below, and shown in Figure 32.

The Transport Services may be implemented using implementations of one to six UoC types, as described in the FACE Technical Standard §3.7.1.1, to provide the functionality required. Solutions with multiple UoCs implement subsets of TSS capabilities with each UoC. At a minimum, a TS Capability, Distribution Capability, and Configuration Capability must be part of the solution whether it be implemented by one UoC, or more than one UoC.

Examples of different TSS UoCs that may be present within a system resulting from different platform needs and business case objectives and goals include:

- TS UoC provided by a middleware supplier and a CSP UoC provided by the System Integrator
- TA UoC provided by a middleware supplier and a TS-TA Interface Adapter UoC through code generation or from a PCS supplier
- TS UoC provided by one middleware supplier and a TPM UoC provided by a transport device supplier
- Framework Services (FS) UoC that implements the SCA standard as an underlying implementation

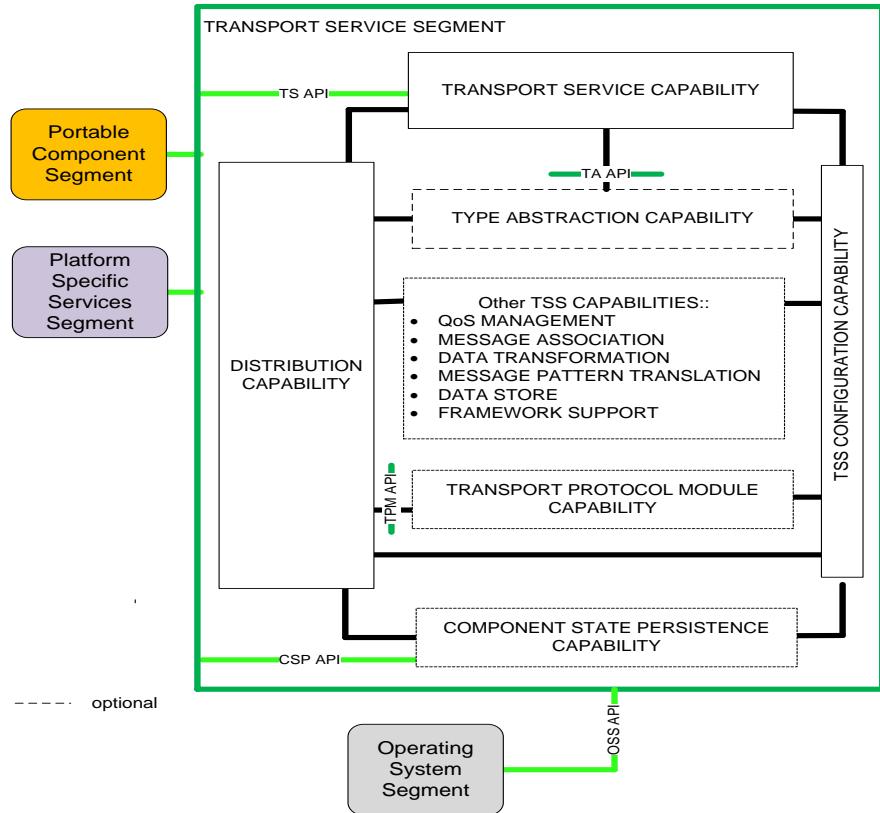


Figure 32: Transport Services Segment Capabilities

Transport Service Capability

The Transport Service Capability provides a Type-Specific Interface, and is responsible for providing the functionality of the Type-Specific Interface to UoCs. The Type-Specific Interface is the only interface that PCS and PSSS UoCs can use to interact with other PCS and PSSS UoCs. The Transport Service Capability provides the TypedTS and Read_Callback interfaces described in the FACE Technical Standard §E.3.2 and provides the Base interface described in the FACE Technical Standard §E.3.1. Note that the Base interface is supplied by either the UoC implementing the Type Abstraction or the Type-Specific Interface, not both. A new TS Type-Specific Extended Type interface has been defined, which is optional, and is described in the FACE Technical Standard §E.3.4. The TS Type-Specific Extended Type module provides different interfaces that better support request/reply message exchange patterns for PCS and PSSS users. The Transport Service Capability provides the interface for data exchange.

Distribution Capability

The Distribution Capability controls or manages the distribution of data within a TSS. It routes data to internal TSS functions or to destination endpoints. It may support functionality for data marshalling and transformations which may be used by other TSS capabilities within a TSS UoC.

Configuration Capability

The TSS Configuration Capability is responsible for managing the configuration of a TSS UoC. It uses the OSS Configuration Services to obtain the necessary TSS UoC configuration data, and it then manages configuration of TSS Capabilities.

Type Abstraction Capability

The Type Abstraction (TA) Capability provides a Type Abstraction Interface, and is responsible for providing the functionality of the Type Abstraction Interface to the Transport Service Capability. The Type Abstraction Capability provides the TypeAbstractionTS and Read_Callback interfaces described in the FACE Technical Standard §E.4.1.

QoS Management Capability

Quality of Service (QoS) may be supported by a TSS implementation and would be provided by the QoS Management Capability. The QoS use key and value attributes are defined in the FACE Technical Standard §E.2.1. The QoS attributes can be a parameter within an existing API when controlled by the UoC or the System Integrator could use configuration data.

Message Association Capability

Message Association may be supported when there are differences between source(s) of messages and destination message characteristics (e.g., structure, rate, types, etc.).

Data Transformation Capability

Data Transformation Capability performs transformations of data when the consuming software component requires its message in a different format (USM platform view) than the source(s) provides. The FACE Technical Standard does not specify or constrain when transformations are performed, such that transformations can occur as data is output or as data input is received.

Messaging Pattern Translation Capability

Messaging Pattern Translation Capabilities manage the message pattern, message communication types, transform from one message pattern to another message (e.g., Request/Reply to Publish/Subscribe), and other message processing requirements.

Transport Protocol Module Capabilities

The Transport Protocol Module (TPM) Capability allows selection between multiple protocols and transports depending on connectivity requirements. The Distribution Capability may be configured to route data to/from one or more TPMs to communicate across different protocols and transports. A TPM need not be used if the TSS natively supports the protocol or transport. A TPM allows the TSS to be extended to communicate to protocols and transports for which it was not originally designed.

Data Store Support Capability

The Data Store Support Capability provides PCS and PSSS UoCs with access to files that are not locally stored. To maintain portability of PCS and PSSS UoCs across a wide variety of system

architectures, the Data Store Support Capability allows PCS and PSSS UoCs to remain agnostic to the source of this information by using the TS Interface to access files.

Component State Persistence Capability

Component State Persistence (CSP) Capability provides a component state persistence interface, and is responsible for storing and retrieving private and checkpoint UoC data. The Component State Persistence Capability provides the CSP Interface described in the FACE Technical Standard §E.3.5. The data passing through the CSP Interface is exempt from being modeled in the FACE Data Architecture and limited to single UoC use.

Framework Support Capability

Framework Support Capability (FSC) maps framework-unique interfaces to the FACE Interfaces to make PCS and PSSS UoCs agnostic to specific frameworks. The framework provides the underlying implementation of the TSS capabilities. Framework Services, such as system time or error logging, are data modeled messages exchanged between PCS and/or PSSS UoCs and the TSS using the TS Interface. Other interfaces, such as HMFM and Configuration defined by Appendix F and G of the FACE Technical Standard respectively, may be supported as part of the Framework Support Capability. The Framework Support Capability may also use the Life Cycle interfaces defined by Appendix D of the Technical Standard to manage and control PCS and/or PSSS UoCs.

7.2.1 Summary of Changes from FACE Technical Standard, Edition 2.1

Six primary changes to TSS concepts have occurred in this edition of the Technical Standard:

1. Updated the API using the most current version of the IDL specification including IDL templates and new language mapping, which support multiple instantiations of PCS, PSSS, and TSS components in the same memory address space. The IDL specification updates result in the use of the Injectable Interface to realize the TSS UoC interfaces rather than linking directly. The API's use of in, out, and inout parameter definition aligns with standard IDL usage.
2. The redundant interfaces for initialize, create connection, destroy connection, and register callback between the Type-Specific and Type Abstraction Interfaces in FACE Technical Standard, Edition 2.1 were consolidated into one interface called Base, described in the FACE Technical Standard, Edition 3.0 §E.3.1. This removes the need for the type-specific initialize to call the type abstraction initialize, etc. Because of this, the Base interface can be implemented by the UoC providing the TypedTS interface (§E.3.2 of Edition 3.0) or the UoC providing the TypeAbstraction interface (§E.4.1 of Edition 3.0).
3. The addition of the TPM for TS interoperability, including provisions for the serialization of data messages.
4. A new CSP Interface for PCS, PSSS, and TSS UoCs to use.
5. Additional parameters in the API to include the header data and QoS parameters.
6. Additional send and receive interfaces (Type-Specific Extended Type Interface, §E.3.4) to better support request/reply paradigms.

7.2.2 How to Read this Section

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

7.3 TSS Technical Description

7.3.1 TSS Capabilities

The TSS is an architectural concept used to promote portability and reusability of application software in the PCS and PSSS segments. To achieve portability of components outside of the TSS, the TSS may need customization for a particular instance of a PCS/PSSS UoC. The internal TSS architectural elements, such as the Type Abstraction and Transport Protocol Module, were introduced to allow TSS implementations to separate application-specific and TSS general capabilities. If a TSS implementation chooses to take advantage of these architectural elements, they are considered separate UoCs which can be conformed independently.

The goal of the TSS Framework Support Capability is to map how to implement a TSS using an underlying component framework and still treat the business logic software as components that execute in a framework. The business logic aligns to the other PCS/PSSS segments, while the TSS provides the execution model and transport capabilities for those components.

The goal of the TSS Data Store Support Capability is to map a PCS/PSSS UoC's use of network storage through the TSS.

The goal of the CSP Interface is for PCS/PSSS UoCs to store and retrieve their internal state. Using the CSP Interface allows the System Integrator to implement a power-up sequence which doesn't wait for the initialization of a transport technology to complete; shortening the recovery time of a PCS/PSSS UoC after a power interrupt. Note that the TSS may also use the CSP to store and retrieve their internal state.

7.3.2 TSS Interfaces

The TSS facilitates the integration of software into disparate architectures and platforms using different transports. The TSS can be comprised of implementations of one to six UoC types, each satisfying a proper set of capabilities within the segment. The TSS has two interfaces provided for PCS or PSSS UoCs to use: Type-Specific Interface and the Component State Persistence (CSP) Interface. To enable smaller subsets of capabilities within the TSS to be conformed independently, two intra-segment interfaces have been created: Type Abstraction (TA) Interface and the Transport Protocol Module (TPM) Interface.

The Type-Specific Interface has been organized into several IDL modules to clarify the extension of the interface for each using the UoC's message types. In previous editions, a FACE_SPECIFIED_TYPE placeholder was used to indicate which interface methods are impacted by a using the UoC's message type. In this edition, the TS Interface has been organized into separate IDL modules and uses IDL templating as required to parameterize the interface into message-specific interfaces.

- Base – interface methods that do not change because of a UoC's message type

- Typed – original interface methods such as send and receive that are extended by a UoC for each message type in their USM
- Message Serialization – new helper interface methods to marshal and unmarshal UoC messages to/from its data representation used for storage or transmission
- Extended Typed – new interface methods to enhance support for request/reply communication patterns

Additionally, to support various layering approaches to the TSS, the Base interface may be provided by a single monolithic TSS UoC or, if non-homogenous, the Type Abstraction UoC to enhance separation of concerns between the connection and distribution management and message formatting.

7.4 TSS Use Cases

7.4.1 TSS Send

TSS Send is used by a PCS or PSSS UoC to make their messages available for distribution on the supported transport technologies. Messages may be sent at the time the send function is invoked or they may be transmitted at a later time based on the implementation of the TSS. The TSS is responsible for moving datagrams built from messages to their intended destinations enabling communications between FACE UoCs. The TSS implementation may need to accommodate systems that require special handling and service demands due to network faults, application deadlines, retransmission, or other communication concerns. The use of *Send_Message()* is described below and shown in Figure 33.

1. A PCS/PSSS UoC using the send message interface must get a *connection_id* prior to using the send message interface by calling *Create_Connection()*.
2. A *Send_Message()* can be called at any time. If the timeout parameter >0, the TSS returns when the message is transmitted on the transport or the timeout is reached, whichever occurs first.

Note that *Create_Connection()* is part of the *FACE::TSS::Base* interface. A *Send_Message* interface is specific to each message type and UoPModel “name” defined in the USM for the UoC. For example, a fully qualified path for a message type called “position”, in a UoPModel named myUoC, is *FACE::TSS::myUoC::position::TypedTS::Send_Message()*.

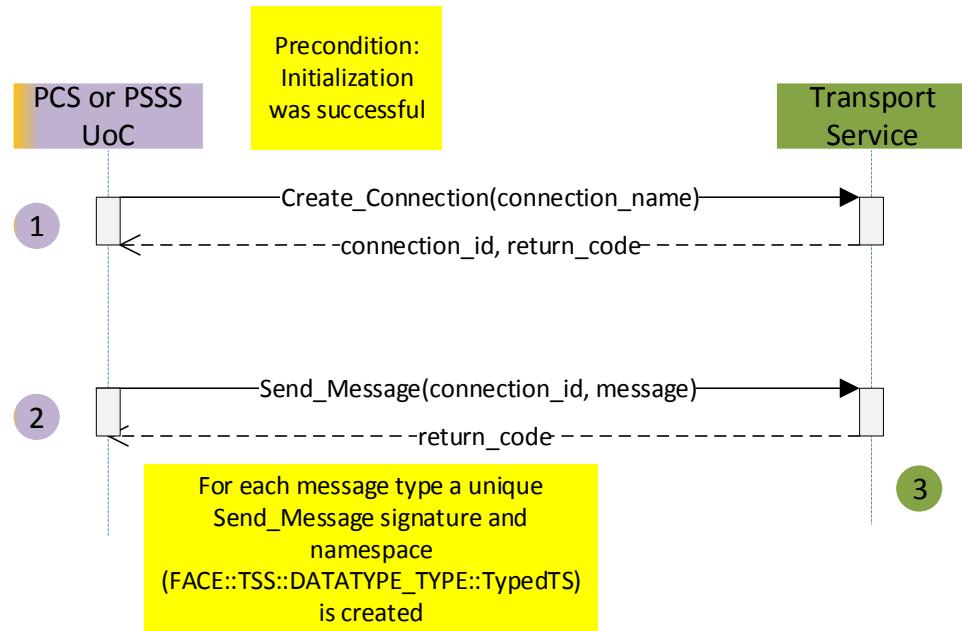


Figure 33: Send Sequence Diagram

In some TSS implementations, the TSS will be composed of two UoCs: TS-TA Adapter UoC and the Type Abstraction UoC. As shown in Figure 34, the PCS/PSSS has the same Type-Specific Interface and its sequence is described as follows:

1. The PCS/PSSS UoC must get a *connection_id* prior to using the send message interface by calling *Create_Connection()*. A separate *connection_id* is required for each different connection. There is one and only one message type defined for a publish/subscribe connection. See Sections 7.4.1 and 7.4.2 for request/reply.
2. The PCS/PSSS UoC initiates the send message using the Type-Specific Interface.
3. The TS-TA adapter then invokes the Type Abstraction send message call after assigning the MESSAGE_TYPE struct to the type-specific message.
4. If a second message is sent by the PCS/PSSS UoC, it initiates the send message call when it wants to send the data.
5. The second message uses the same Type Abstraction send message call to output the data to the transport.

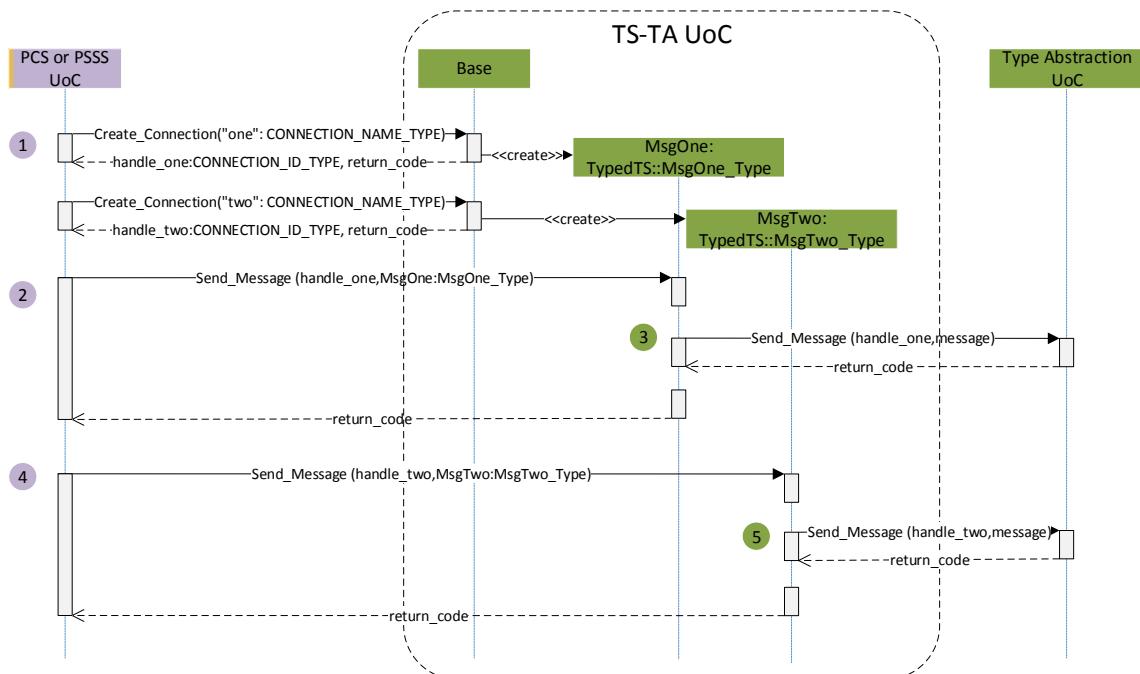


Figure 34: Send Using Type Abstraction Sequence Diagram

To promote interoperability and portability of different FACE Computing Environments, return codes for the different API calls should be consistent regardless of the underlying transport mechanism. Table 141 describes a potential mapping of different technologies to the FACE return codes for *Send_Message()*.

Table 141: Potential Transport Technology Mapping to TSS Return Codes on Send

| TSS Return Code | ARINC 653 | POSIX Socket | DDS® | CORBA® |
|-----------------|---------------|--|--|---|
| NO_ERROR | NO_ERROR | 0 | OK | 0 or no exception thrown. |
| NO_ACTION | NO_ACTION | EAGAIN EWOULDBLOCK ECONNRESET EACCES ENOBUFS | ILLEGAL_OPERATION BAD_PARAMETER ERROR UNSUPPORTED PRECONDITION_NOT_MET NOT_ENABLED NO_DATA OUT_OF_RESOURCES | INTERNAL INVALID_POLICY NO_RESOURCES BAD_INV_ORDER INV_OBJREF MARSHAL TRANSIENT |
| NOT_AVAILABLE | NOT_AVAILABLE | None. | None. | None. |
| INVALID_PARAM | INVALID_PARAM | None. | None. | None. |
| TIMED_OUT | TIMED_OUT | ETIMEDOUT | TIMEOUT | TIMEOUT |

| TSS Return Code | ARINC 653 | POSIX Socket | DDS® | CORBA® |
|-------------------|-----------|-------------------|-----------------|------------------|
| CONNECTION_CLOSED | None. | EBADF ENOTSOCK | ALREADY_DELETED | OBJECT_NOT_EXIST |
| DATA_OVERFLOW | None. | None. | None. | None. |

7.4.2 TSS Receive

Receive_Message() is used by a PCS or PSSS UoC to poll the TSS for each of their type-specific message types from the transport. Different TSS implementations will have different execution models; TSS processing may occur at different times related to the TS Interface call. For example, the receipt of incoming datagrams may be processed on its own TSS thread, interrupt-driven, or piggybacked onto a *Receive_Message()* call.

No matter how the TSS is implemented, the TSS is expected to always return the most recent message received. However, if the PCS/PSSS polls faster than datagrams are received, the TSS may not have new messages to return. Likewise, if the PCS/PSSS polls slower than datagrams are received, the TSS may have overwritten previously received datagrams. The configuration, when defined by the System Integrator, will influence how the TSS performs. If a PCS/PSSS UoC requires every instance of a message, it can use the *Read_Callback()* or *Receive_Message()* interface to receive each instance of a message. If *Read_Callback()* is used, minimize the time in the callback to avoid message loss. To ensure every message is received when polling, the System Integrator configures the message rate on the transport to be slower than the polling rate specified in the USM. The use of *Receive_Message()* is described below and shown in Figure 35.

1. A PCS/PSSS UoC must supply a *connection_id* when calling *Receive_Message()*, which is returned using the call to *Create_Connection()*.
2. If the PCS/PSSS UoC requires the same message from multiple sources, two different connections are used to receive from those different sources. Otherwise, only one connection is required to receive data and that data can be from any source for the data in the system. A single connection does not necessarily guarantee data from each source is available to the PCS/PSSS UoC.
3. The TSS may set up its own thread to read the transport technology. This may result in data being queued in between the PCS/PSSS UoC *Receive_Message()* calls.
4. A *Receive_Message* is called to poll the TSS for data. If the timeout parameter >0, TSS will return when data is ready to return or the timeout is reached, whichever occurs first. If no timeout is specified and there is no data to return, the *return_code* is “NO_ACTION”. Once data is returned by the TSS it is no longer retained by the TSS.
5. If the PCS/PSSS UoC needs to age the message appropriately, the header parameter includes a timestamp in the returned data from the *Receive_Message()* call.
6. The PCS/PSSS UoC, having set up two separate connections for the same data from two sources, polls for each message by calling the *Receive_Message()* interface with the different *connection_ids*.
7. If the PCS/PSSS UoC needs to compare the data from multiple sources, the header parameter includes a *source_uid* and *instance_uid*.

Note that *Destroy_Connection()* is part of the *FACE::TSS::Base* interface in addition to *Create_Connection()*. A *Receive_Message()* interface is specific to each message type and UoPModel “name” defined in the USM for the UoC. For example, a fully qualified path for a message type called “position”, in a UoPModel named myUoC, is *FACE::TSS::myUoC::position::TypedTS::Receive_Message()*. An incoming Type Abstraction message is passed to the TS-TA adapter in a data buffer of maximum size defined by its capacity. The TS-TA adapter uses knowledge of the IDL language mappings to then pass the fixed or variable-length message to the type required by the TypedTS *Receive_Message()* interface.

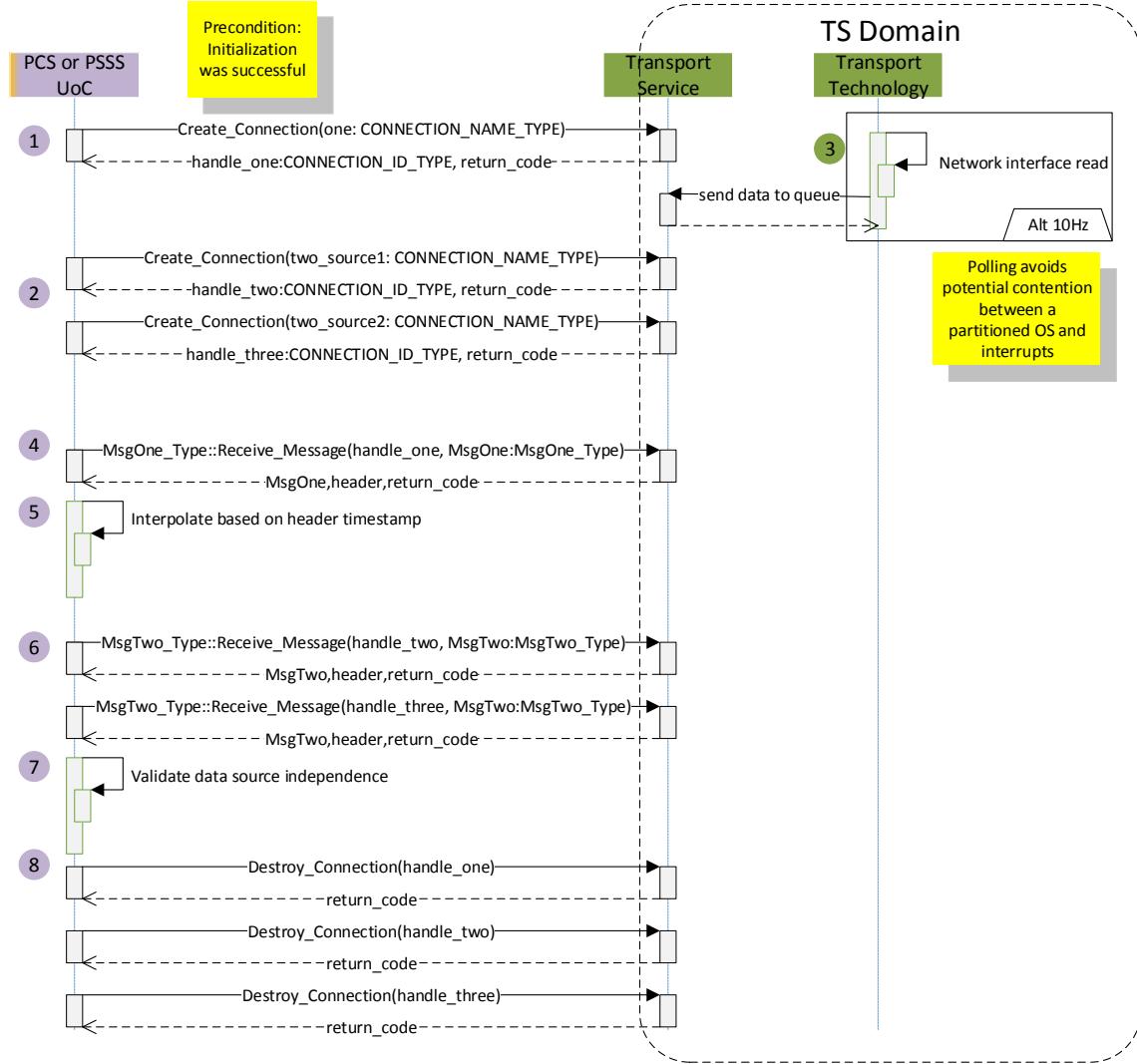


Figure 35: Receive Sequence Diagram

Sometimes PCS/PSSS UoCs will need a request/reply message pattern where the outgoing message is a request for data from a service in the system and the reply needs to be associated with the request. Figure 36 describes the Request/Reply sequence.

1. One connection is used for both the request and reply messages.
2. The PCS/PSSS UoC initiates the request message.

3. The TSS can use any underlying transport technology to distribute the message to the remote service, including the use of two physical multi-cast connections set up for one-way data traffic.
4. The PCS/PSSS UoC initiates the *Receive_Message()* (or uses *Read_Callback()*) using the same *connection_id* and *transaction_id* as in the *Send_Message()* call. The returned *transaction_id* is used to associate the reply message to the initial request.

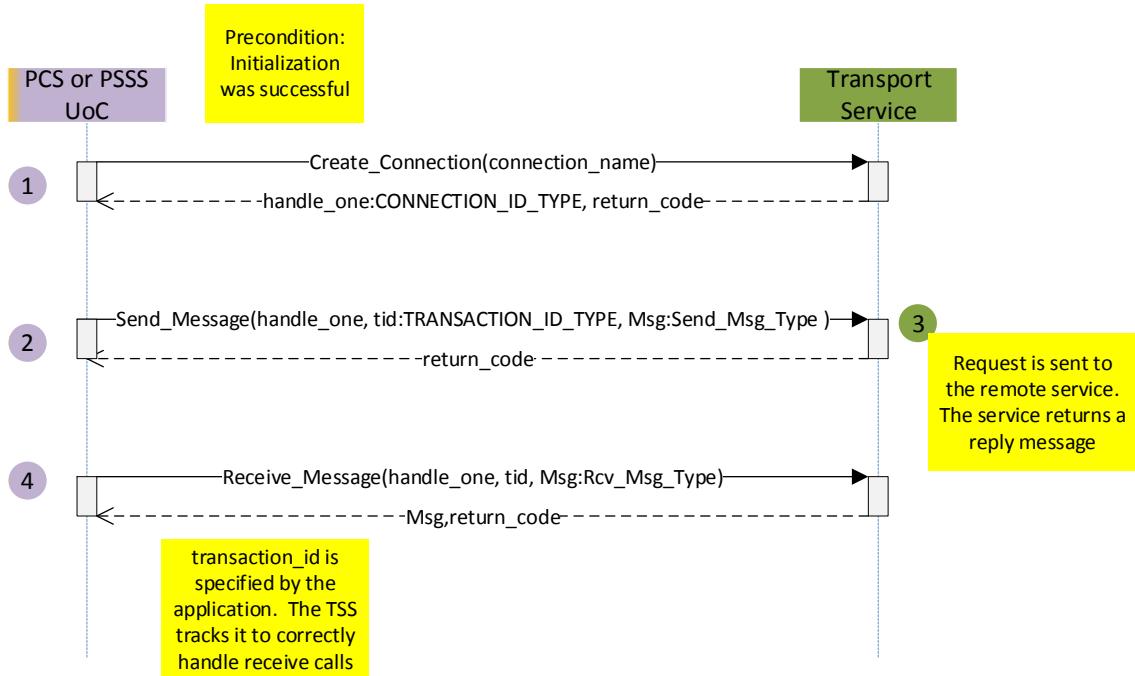


Figure 36: Request Reply Sequence Diagram

Clients and servers may be using a single connection or multiple connections to handle one or more request messages. Regardless, the TSS needs to ensure the server's reply message is correctly associated with the corresponding client's request message across the system.

Return codes for the different API calls should be consistent regardless of the underlying transport mechanism. Table 142 describes a potential mapping of different technologies to the FACE return codes for Receive.

Table 142: Potential Transport Technology Mapping to TSS Return Codes on Receive

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|-----------|--------------|-----|---------------------------|
| NO_ERROR | NO_ERROR | 0 | OK | 0 or no exception thrown. |

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------------|---------------|--|--|---|
| NO_ACTION | NO_ACTION | EAGAIN EWOULDBLOCK ECONNRESET EACCES ENOBUFS | ILLEGAL_OPERATION BAD_PARAMETER ERROR UNSUPPORTED PRECONDITION_NOT_MET NOT_ENABLED NO_DATA OUT_OF_RESOURCES | INTERNAL INVALID_POLICY NO_RESOURCES BAD_INV_ORDER INV_OBJREF MARSHAL TRANSIENT |
| NOT_AVAILABLE | NOT_AVAILABLE | None. | None. | None. |
| INVALID_PARAM | INVALID_PARAM | None. | None. | None. |
| INVALID_MODE | INVALID_MODE | None. | None. | None. |
| TIMED_OUT | TIMED_OUT | ETIMEDOUT | TIMEOUT | TIMEOUT |
| MESSAGE_STALE | None. | None. | None. | None. |
| CONNECTION_CLOSED | None. | EBADF ENOTSOCK | ALREADY_DELETED | OBJECT_NOT_EXIST |
| DATA_BUFFER_TOO_SMALL | None. | EMSGSIZE | None. | None. |
| DATA_OVERFLOW | None. | None. | None. | None. |

7.4.3 TSS Callback Handler

Callbacks are used to notify a PCS or PSSS UoC when their data has been received by the transport, as shown in Figure 37. The Transport Services defines a *Read_Callback()* interface, with the *Callback_Handler()* prototype. The use of callbacks is described below and shown in Figure 37.

1. The PCS and PSSS UoC users of the TSS implement a *Callback_Handler()* for each message type it expects to receive in this manner.
2. The PCS/PSSS UoC registers a different *Callback_Handler()* for each message type for each connection using the *Register_Callback()* interface. Registration occurs after a connection is created between the PCS/PSSS UoC and the TSS.
3. Once data is received on the transport of the message type, the TSS invokes the callback handler provided by the PCS or PSSS UoC.

Users of the TSS should expect that different TSSs will have different implementations; TSS processing may occur at different times related to the TS Interface call. For example, the receipt of incoming datagrams may be processed on its own TSS thread, interrupt-driven, or piggyback onto a *Receive_Message()* call. For a callback to be supported, the TSS would need to process

incoming datagrams out of band to TS Interface calls. Figure 37 is an example of the Callback Handler interface with one datagram and connection.

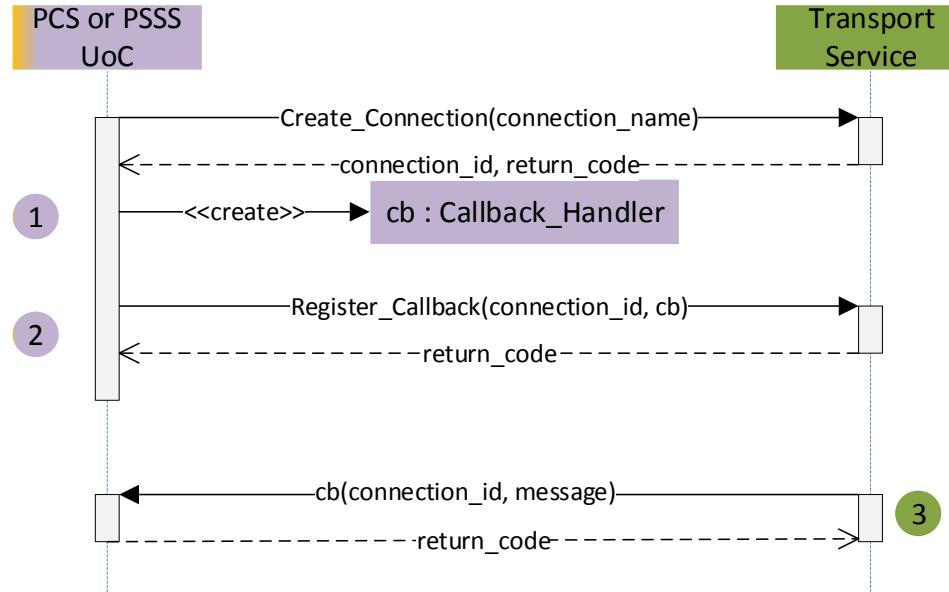


Figure 37: Callback Sequence Diagram

Return codes for the different API calls should be consistent regardless of the underlying transport mechanism. Table 143 describes a potential mapping of different technologies to the FACE return codes for Callback Handler.

Table 143: Potential Transport Technology Mapping to Return Codes on Callback Handler

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|-----------|--------------|-------|---------------------------|
| NO_ERROR | NO_ERROR | 0 | OK | 0 or no exception thrown. |
| DATA_OVERFLOW | None. | None. | None. | None. |

7.4.4 TSS Initialize

The TSS initialization is system-defined and is dependent on the complement of UoCs integrated into the same address space. When a TSS UoC provides the capabilities of the TS Interface, Distribution Capability, and any other optional TSS capabilities, it still must provide the dependency injection interface for setting a TPM reference. Otherwise, a TSS composed of multiple TSS UoCs, as shown in Figure 38, provides the dependency injection interface for each capability it uses.

The `Set_Reference()` interface provided by the Injectable Interface is specific to each interface type and has a unique namespace with a fully qualified path for the interface type. For example, the fully qualified path for the TSS Base interface is `FACE::Base::Injectable::Set_Reference()`. If there is more than one reference of the same interface type, the `interface_name` parameter in the `Set_Reference()` call is unique. Note that both the TSS `Read_Callback()` and

Message_Serialization() interfaces do not need the dependency injection interface. Callbacks and Message Serialization functions are registered or fetched, respectively, using the *Register_Callback()* and *Get_Serialization()* interfaces.

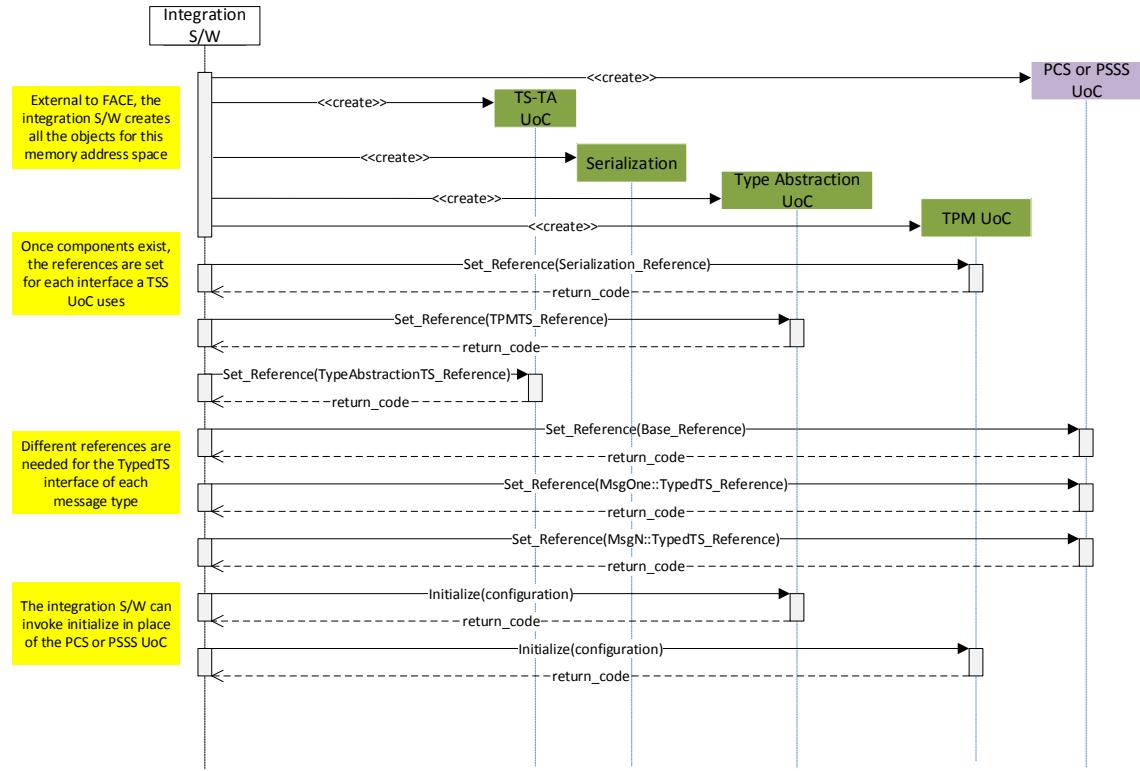


Figure 38: Initialize Sequence Diagram

Once UoC references are set, initialize can be called by the Integration software or the UoC using the TSS. A TSS that blocks on initialize returns NO_ERROR if it successfully completes its initialization. Alternatively, initialize may be non-blocking and return even though initialization hasn't completed. In this case, subsequent calls to initialize can be used to determine TSS initialize is complete, as shown in Figure 39.

1. The PCS/PSSS UoC invoking initialize is returned IN_PROGRESS. The TSS initialize completes its initialization in the time between the UoC's second and third call to initialize.
2. Any calls to initialize after it is complete returns NO_ACTION.

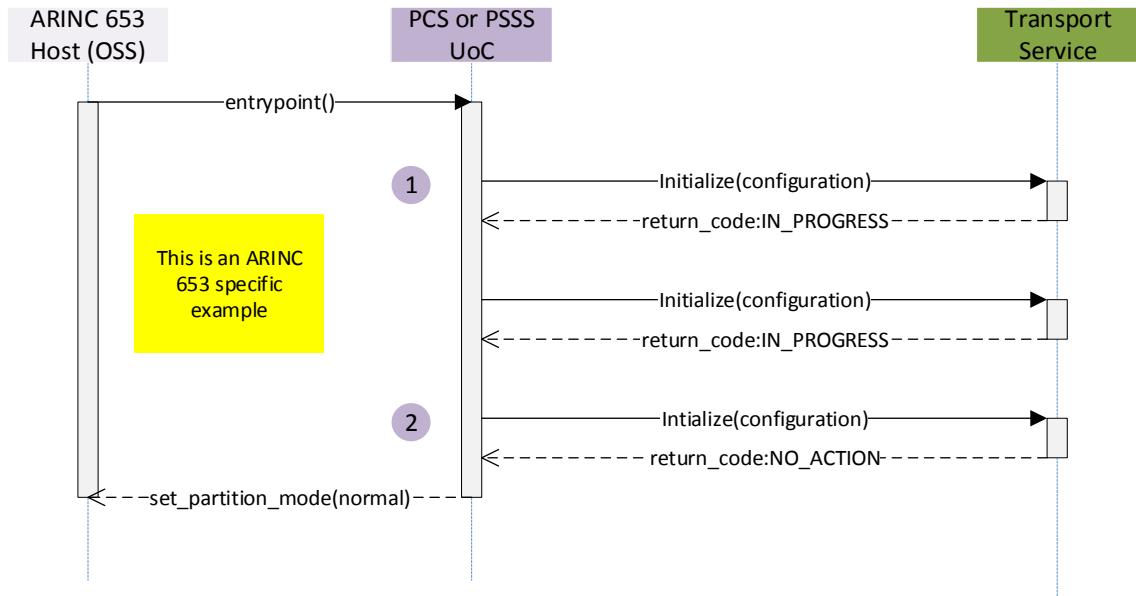


Figure 39: Example ARINC 653 Application Initializing the TSS

The format of the configuration data is specific to the TSS implementation, though implementers are encouraged to employ the integration model (see the Reference Implementation Guide for FACE Technical Standard, Edition 3.0, Volume 3: Data Architecture §3.1.3). What processing is performed during initialize is also implementation-specific. For instance, one TSS implementation may choose to open all of its configured connections to a transport during initialize. This supports transport technologies such as ARINC 653 communications which can only be configured during the initial mode of the partition. However, a different TSS implementation may choose to only initialize itself and wait to initiate a connection on the transport technology until the `Create_Connection()` call. The PCS and PSSS UoCs are built to be agnostic to those TSS differences. Note that PCS and PSSS UoCs may be designed to certain paradigms. For example, an ARINC 653 PCS UoC is likely to invoke initialize and create connection during its own initialization to align with design paradigms in ARINC 653 systems.

Return codes for the different API calls should be consistent regardless of the underlying transport mechanism. Table 144 describes a potential mapping of different technologies to the FACE return codes for Initialize.

Table 144: Potential Transport Technology Mapping to TSS Return Codes on Initialize

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|-----------|--------------|-------------------|---------------------------|
| NO_ERROR | NO_ERROR | 0 | OK | 0 or no exception thrown. |
| NO_ACTION | NO_ACTION | None. | None. | None. |
| NOT_AVAILABLE | None. | ENETDOWN | ILLEGAL_OPERATION | BAD_OPERATION |

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|----------------|--|--|---|
| INVALID_CONFIG | INVALID_CONFIG | EBADF EMFILE ENFILE EPROTOTYPE EAFNOSUPPORT EPROTONOSUPPORT EADDRINUSE ADDRNOTAVAIL EFAULT EINVAL ENOTSOCK EOPNOTSUPP | PRECONDITION_NOT_MET OUT_OF_RESOURCES INCONSISTENT_POLICY ERROR IMMUTABLE_POLICY BAD_PARAMETER UNSUPPORTED | IMP_LIMIT INITIALIZE INTERNAL INTF_REPOS INVALID_POLICY NO_RESOURCES BAD_CONTEXT BAD_INV_ORDER BAD_PARAM BAD_TYPECODE INV_FLAG INV_IDENT INV_OBJREF TRANSIENT UNKNOWN |
| IN_PROGRESS | None. | None. | None. | None. |

7.4.5 TSS Create Connection

The TSS *Create_Connection()* function is responsible for initializing a named connection. The connection created is between the UoC and the TSS and may not be a one-to-one mapping to a connection on the transport technology. A call to *Create_Connection()* is required for each publish/subscribe message defined by the calling UoC. A UoC may create multiple connections for the same type of message if it requires the same data from multiple sources. Figure 40 describes a UoC creating two connections named “one” and “two”.

1. The TSS must be successfully initialized prior to *Create_Connection()*.
2. A call to *Create_Connection()* results in the TSS associating the UoC message to a connection on the transport technology. The configuration parameters for these connections are established by the TS configuration referenced in the call to initialize. In this particular example, ARINC 653 queues, etc. have to be set up before setting the mode to normal mode, so *Create_Connection()* is done as part of the UoC’s initialization.

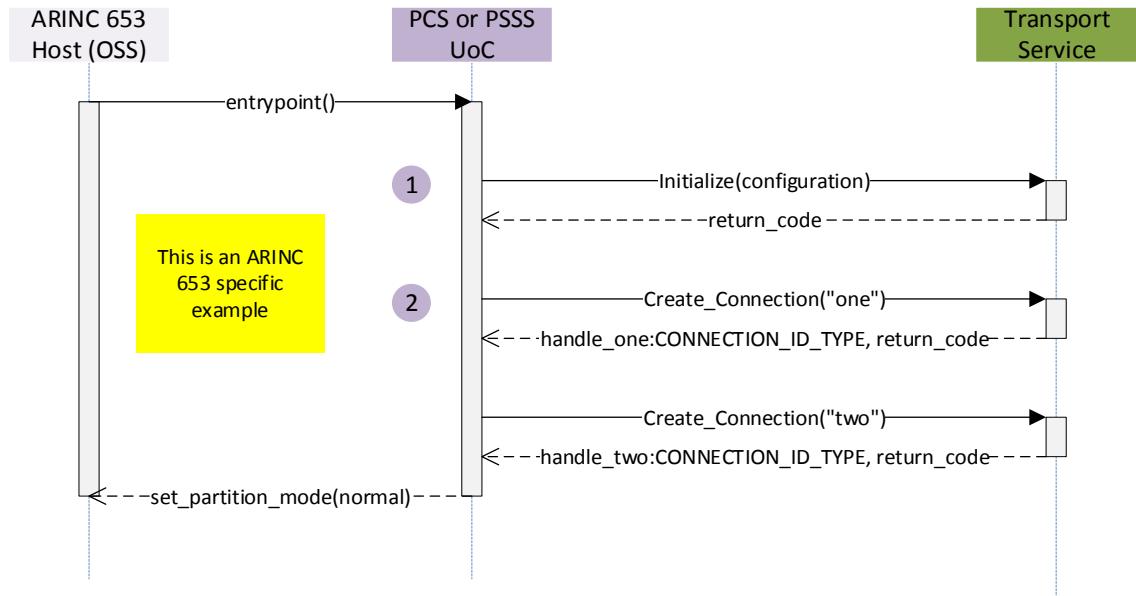


Figure 40: Create Connection Sequence Diagram

For POSIX PCS/PSSS UoCs, the TSS might have configuration stating on which port to listen. The call to `initialize` would then tell the TS which port to use, and the call to `Create_Connection()` would open a socket on the port. The point at which the socket is actually opened is implementation-dependent.

Return codes for the different API calls should be consistent regardless of the underlying transport mechanism. Table 145 describes a potential mapping of different technologies to the FACE return codes for `Create_Connection()`.

Table 145: Potential Transport Technology Mapping to TSS Return Codes on Create Connection

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|-----------|--|--|--|
| NO_ERROR | NO_ERROR | 0 | OK | 0 or no exception thrown. |
| NO_ACTION | NO_ACTION | EBADF EMFILE ENFILE EPROTOTYPE EAFNOSUPPORT EPROTONOSUPPORT EADDRINUSE ADDRNOTAVAIL EFAULT EINVAL ENOTSOCK | PRECONDITION_NOT_MET OUT_OF_RESOURCES INCONSISTENT_POLICY ERROR IMMUTABLE_POLICY BAD_PARAMETER UNSUPPORTED | IMP_LIMIT INITALIZE INTERNAL INTF_REPOS INVALID_POLICY NO_RESOURCES BAD_CONTEXT BAD_INV_ORDER BAD_PARAM BAD_TYPECODE INV_FLAG INV_IDENT |

| TSS Return Code | ARINC 653 | POSIX Socket | DDS | CORBA |
|-----------------|----------------|--------------|---------|------------------------------------|
| | | EOPNOTSUPP | | INV_OBJREF TRANSIENT UNKNOWN |
| NOT_AVAILABLE | None. | None. | None. | None. |
| INVALID_PARAM | None. | None. | None. | None. |
| INVALID_CONFIG | INVALID_CONFIG | None. | None. | None. |
| TIMED_OUT | TIMED_OUT | ETIMEDOUT | TIMEOUT | TIMEOUT |

7.4.6 Transport Protocol Module

The implementation of a TSS supports being configured for a Transport Protocol Module (TPM) even when the initial deployment of the TSS does not include a TPM component. Once a TPM becomes part of a deployed implementation, it may have dependencies on the TSS to provide it with the *FACE::TSS::Serialization::Get_Serialization()* interface to get a reference to the Message Serialization or deserialization objects based on the message's unique identification. The TSS will need to support the Serialization Interface even though the initial deployed system may not include a TPM component.

In the following example (Figure 41), an existing TSS has been implemented using sockets as its transport technology. In this example, the TSS is being re-applied into an ARINC 653 partition requiring both sockets and ARINC 653 communications (ports and queues). A TPM is used to add the ARINC 653 communications to the existing TSS.

1. A TPM instance is created as well as the TSS instance. The TSS is provided with the TPM reference using the injectable *Set_Reference()* interface.
2. During initialization, the TPM creates the ARINC 653 ports and queues using the OSS interface as per the ARINC 653 Specification.
3. Once initialization is complete, the processing platform is set to normal mode.
4. The TSS invokes the TPM read and write interfaces for messages distributed via ARINC 653 communications.

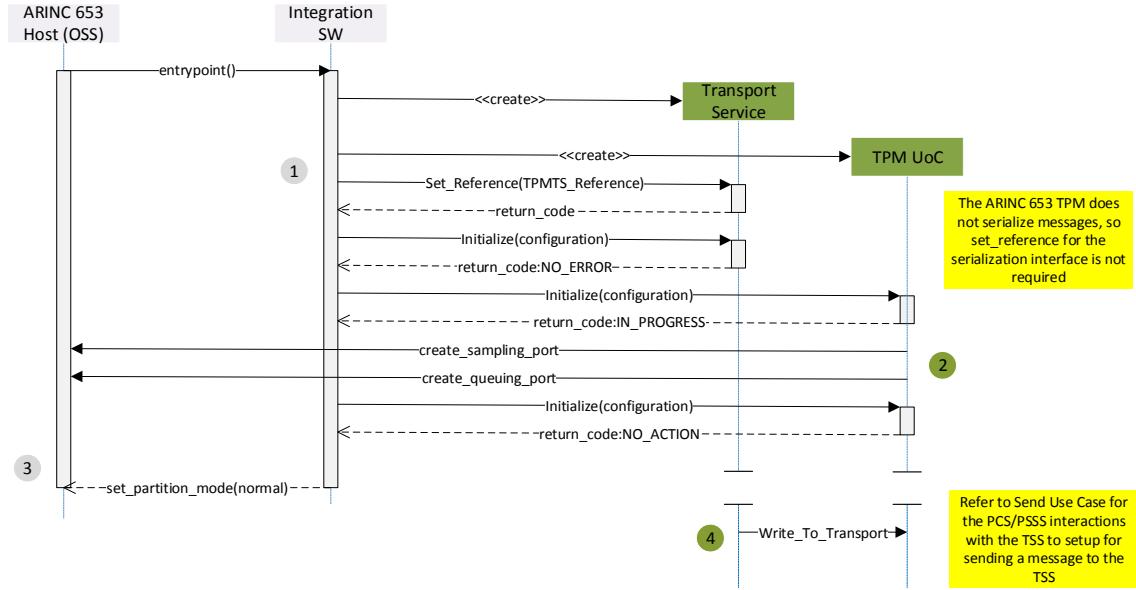


Figure 41: Adding a TPM Sequence Diagram

7.4.7 TSS Message Serialization

Message Serialization was introduced to support TPMs; although their use is not limited to only when a TPM is present in the system. Neither is it required for a TSS, Type Abstraction, nor TPM to use Message Serialization. The Message Serialization can be designed to use a TPM's primitive marshalling interface if the TPM primitive type encoding is unknown to the system or it can be designed to be self-contained to include the encoding logic.

These two examples are shown in the following diagram. They are not intended to be the only way Message Serialization can be implemented.

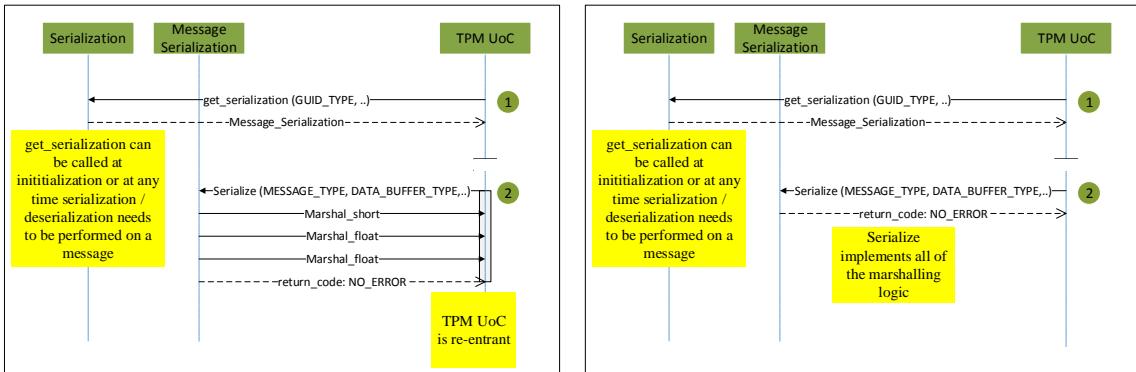


Figure 42: Example Serialization Sequence Diagram

Note that enumerations and fixed type marshalling are not standardized by the TPM's primitive marshalling interface. A best practice for Message Serialization is to encode messages as late as possible in the process of transporting data to avoid having to encode messages that would not require marshalling given its transport technology. For example, if a message is to be routed using an ARINC 653 sample port, the message itself would not need to be encoded to use this transport. The intent of Message Serialization is to provide a set of helper functions associated with the strongly typed interface of the Transport Service.

7.4.8 TSS Data Store

TSS Data Store provides the TS Interface to retrieve or store files to remote storage. The TS Interface may be built from USMs describing the file's content. The PCS or PSSS UoC uses the TSS calls (e.g., send, receive, etc.) to read and store data. Alternatively, in FACE Technical Standard, Edition 3.0 the TSS interface may reflect a well-known standard's file protocol. New data model constructs allow a USM to reference a protocol's standard and version that is used to exchange files through the TSS.

7.4.9 TSS Component State Persistence

The TSS Component State Persistence (CSP) Interface lets multiple instances of the same defined UoC share data privately; often used to exchange internal state data in support of redundancy. Another use can be for re-initializing a UoC to its previous state on power up. The CSP includes interfaces to open and close the CSP resource assigned to the using UoC. The UoCs can then create, read, update, and delete CSP data. The CSP Read having an inout parameter for the *DATA_BUFFER_TYPE* allows implementations to provide a zero-copy buffer. For zero copy, the CSP implementation changes the *DATA_BUFFER_TYPE* to point to the storage location of the data before returning. CSP storage is reclaimed when the data type is deleted by the using application or the CSP implementation subsequently moves the location of the data for wear-leveling purposes.

Using UoCs that care if their return data is located in the location they provide should communicate those needs to System Integrators and implement defensive coding if necessary. System Integrators should ensure their components are compatible when selecting PCS, PSSS, and CSP components.

7.4.10 Framework Support Capability

The FACE Technical Standard supports the use of Commercial Off-The-Shelf (COTS) component frameworks as described in the FACE Technical Standard §2.6:

1. The OSS implementation provides component framework APIs specified by the OSS.
2. A PCS or PSSS UoC uses a component framework internal to its implementation. The PCS/PSSS UoC still interfaces to the remainder of the system using the required FACE Interfaces externally.
3. Integration software uses a component framework to interface and manage FACE UoCs through the LCM Services interfaces.
4. The TSS implementation uses a component framework internal to the TSS to distribute data and manage PCS and PSSS UoCs. This TSS implementation is providing the TSS Framework Support Capability (FSC).

A TSS FSC UoC does not provide any new interfaces, but enables a system to design its solution where a framework process and control and its functional components are mapped against four layers of the FACE Segments: PCS, TSS, PSSS, and IOSS. The framework processes and control are allocated to the TSS and its interfaces are adapted to the TS Interface, LCM Services interfaces, HMFM Interface, and Configuration Management Interface to manage and communicate to PCS and PSSS UoCs. To provide I/O, the COTS framework would also be adapted to the IOSS Interfaces. Refer to Section 9.4 for a detailed discussion of component management.

7.4.11 TSS Send Message Async

Send_Message_Async() is used by a PCS or PSSS UoC to implement an asynchronous request/reply message pattern in one method. The callback is provided during the *Send_Message_Async()* call rather than being registered at startup. Figure 43 describes the Request/Reply sequence using *Send_Message_Async()*.

1. The client initiates a request message using *Send_message_Async()*. The client UoC provides both a *transaction_id* and *Read_Callback()* method.
2. The Transport Service uses its Distribution Capability to send the request to the Server PCS or PSSS UoC. The TSS is responsible for distributing the *transaction_id* along with the request message to the server.
3. Once the server receives the request and processes a response, it sends the response for the *transaction_id* provided using TSS *Send_Message()*.
4. The Transport Service uses its Distribution Capability to send the response to the Client PCS or PSSS UoC.
5. The Transport Service invokes the client's callback asynchronously.

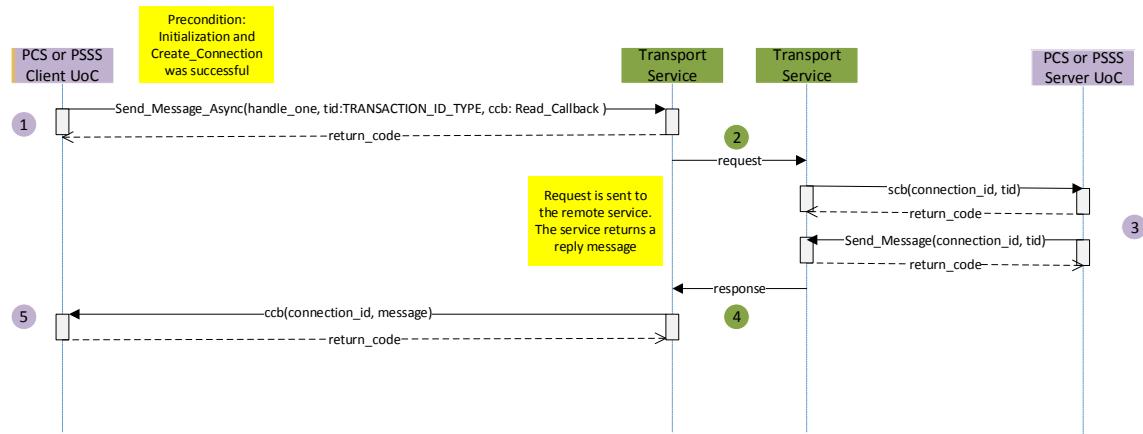


Figure 43: Send Message Async Sequence Diagram

If a UoC wants to wait for the server's reply, *Send_Message_Blocking()* can be used. Note that the *transaction_id* is not a parameter in the *Send_Message_Blocking()* because the request waits for the reply. However, a server receiving a client's request still gets a *transaction_id* parameter in its *Receive_Message()* call. In cases where the server has only one connection for all client requests, the TSS needs to associate the server's reply message to the corresponding client's request message.

7.5 Relationship to Data Architecture Artifacts

7.5.1 UoP Supplied Data Model

The UoP Supplied Data Model (USM) is used to generate the Type-Specific Interfaces of the TSS. The USM comprises data models and UoP models. The data models define a view (*aka* platform view) for each message produced and consumed by the UoC. The USMs define

connection-specific attributes of messages such as the period at which the message is produced by the UoC. The USM is used by a System Integrator to generate a compatible TSS interface for the supplied UoC.

An optional integration model also uses the USM in two ways: first, it realizes a UoC, and second it references a UoC connection. For example, Figure 44 shows a system with two different UoC instances, GPS and Autopilot. In this example, the Autopilot consumes the GPS UoC's message. Information in the integration model describes the data's path through the TSS.

FACE Technical Standard, Edition 3.0 featured a change in the FACE Modeling Language. This change from the language defined by FACE Technical Standard, Edition 2.1 involved replacing the meta-class Message Port field with a Connection field. The USM's Connection includes the periodicity, synchronization style, and the case-sensitive name used in the TSS *Create_Connection()* call. The UoC's use of the timeout parameter in the TSS interfaces will align to the synchronization style documented in the USM. Note that only some of the TSS functions have a timeout parameter and the UoC may be using interfaces which do not have a timeout parameter, such as the callback.

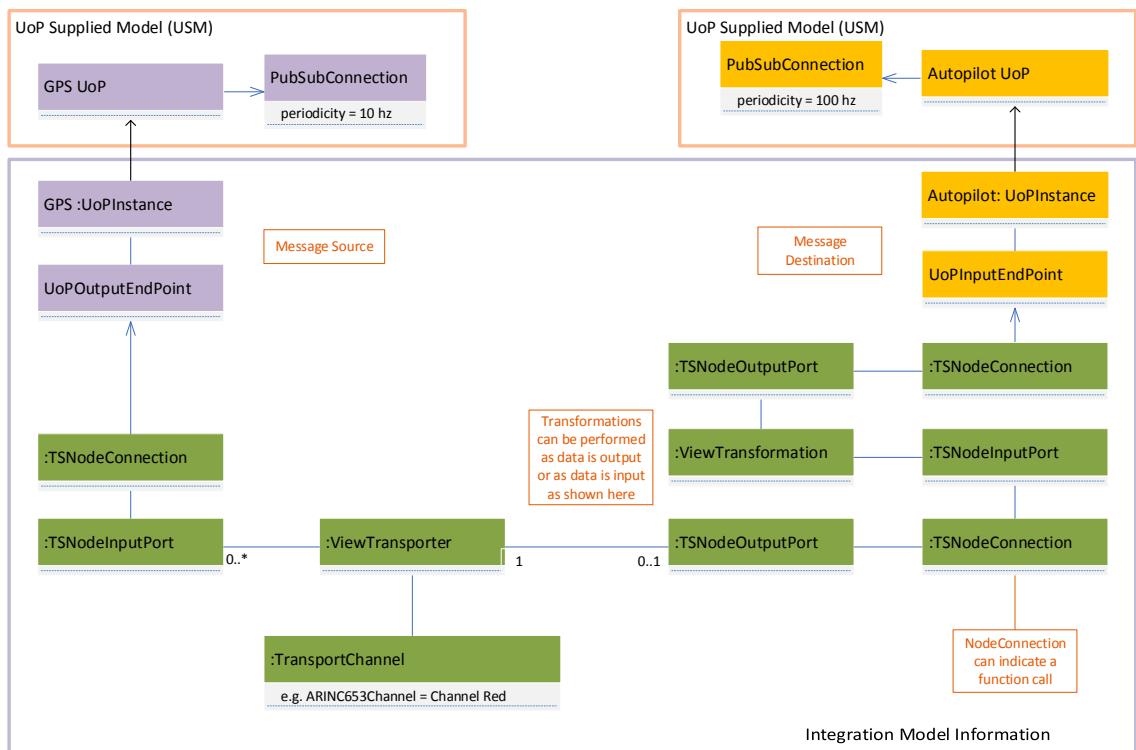


Figure 44: USM Information Related to Integration Model Information

The USM is referenced in code as the connection name's parameter. The TSS accepts the name as an argument to *Create_Connection()* and returns a connection that is now associated to the requested connection. The USM also results in a type-specific signature for the different functions. Below is an example in C++ of a *Send_Message()* call after it has been generated for the position data type of a USM for a domain limited to a single UoC with a data model name of *myUoC_DM*.

```
FACE::TIMEOUT_TYPE timeout = 1000000000;
FACE::DM::myUoC_DM::GPS_Message msg;
```

```

FACE::RETURN_CODE_TYPE::Value return_code;
FACE::TSS::CONNECTION_ID_TYPE connection_id;
FACE::TSS::MESSAGE_SIZE_TYPE max_message_size;
FACE::TSS::TRANSACTION_ID_TYPE transaction_id;

Create_Connection("GPSData", connection_id, max_message_size,
return_code);

Send_Message(connection_id, timeout, msg, transaction_id, return_code);

```

Note: The System Integrator modifies the UoC data model name to create unique namespaces when the same UoC is instantiated multiple times in the same memory address space.

7.5.2 TSS Configuration Relationship to Integration Model

A TSS must be able to distribute data at run-time and apply attributes to the data as it traverses its distribution network based on the needs of the system. The parameters used by a TSS implementation to describe this behavior could be a file read at initialization or generated code using external tools with the same parameters input during build time. Whichever implementation, a TSS would have some capabilities to accept routing information. TSS implementations may also have the ability to associate messages in order to perform transformations or build new messages from multiple messages. Or, additionally, a TSS could have the ability to provide some levels of QoS for data being transmitted along one route or another. A configuration definition can be predefined for predictable behavior; however, the approach does not preclude a TSS implementation making route decisions and QoS decisions at run-time.

The following section walks through one possible approach to defining the configuration for a TSS and relates that example to USM elements and the optional integration model. The integration model provides a standard mechanism for documenting interconnects between UoC instances in a system implementation.

The basis for the example is described in Figure 45 which shows separate structures used to define and relate TSS configuration data for input and output UoC messages and the underlying technology used to transport a message in the system. In this example, each instance of a TSS library uses the same structure, but with data specific to the UoC instances interfacing to the instance of the TSS library. The configuration data defines how to distribute and present messages to consuming UoCs.

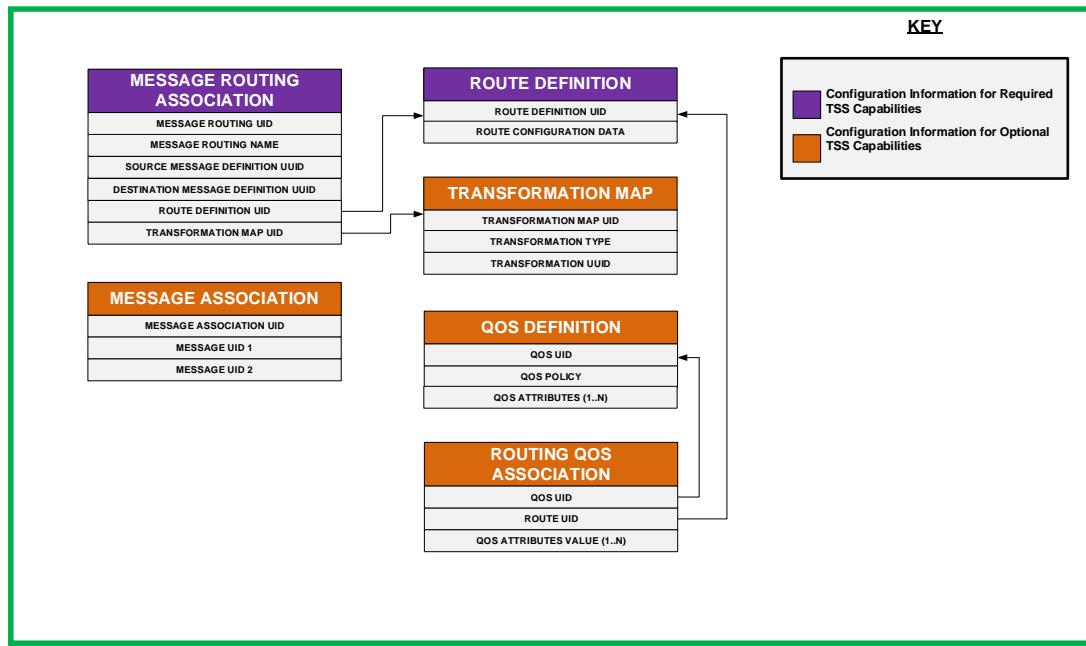


Figure 45: TSS Configuration Data Element Relationships

7.5.2.1 *Distribution and Routing*

The Distribution Capability is a key capability of the TSS and is meant to abstract the transport mechanism away from components residing in the PCS and PSSS. Without it, FACE PCS/PSSS components would be unable to communicate. The TSS allows components to be written generically, without dependencies on a specific transport mechanism or defined communication endpoint. This allows components to be used on other platforms without changes due to different transport mechanisms.

To enable PCS/PSSS portability, TSS implementations must distribute messages in accordance with a specific system design. To do so, configuration information is required by the TSS implementation either at build time or run-time. Neither Figure 45 nor the Integration Model supersede each other, but rather they are both different descriptions to derive the same TSS capabilities.

In the example below, the Message Routing Association and Route Definition provide configuration of the required Distribution Capability.

7.5.2.1.1 *Message Routing Association*

1. **Message Routing UID** – The Message Routing UID uniquely identifies each instance of the Message Routing structure for convenience to the TSS design.
2. **Message Routing Name** – The Message Routing Name is a human-readable description of the message route. Each Message Routing Name is unique. Its purpose is to provide human-readable text to support integration.
3. **Source Message Definition UUID and Destination Message Definition UUID** – Message Definition UUID is a reference derived from a specific Platform View of the USM. It is not necessary to implement the TSS with UUIDs. It is determined as follows:

- a. Each UoC USM provides a *face.datamodel.platform.View* for each message they intend to send and receive through the TSS associated with a *face.uop.Connection*. A USM connection can be a publisher, subscriber, client, server, queuing, or single instance and describes message attributes associated with the platform view such as how often the UoC expects to receive its message (aka periodicity). The name field in the *face.uop.Connection* is a case-sensitive string and represents the value passed into the *FACE::TSS::Base::Create_Connection()* interface in the *connection_name* field. When an instance of the UoC (*face.integration.UoPInstance*) is created in the integration model, the *face.integration.UoPEndPoint* (either *UoPOutputEndPoint* or *UoPInputEndPoint*) references the USM's connection and allows derivation of either the source message definition UUID (see Figure 45) or destination message definition UUID (see Figure 45) depending on whether the message is defined as an output or input, respectively.
 - b. Then, once the data flow routes are created in the integration model, the remaining source message definition UUID or destination message UUID is derived.
4. **Route Definition UID** – The Route UID references an instance of the route definition structure associated with the path to transport this source data to its destination(s).
 5. **Transformation Map UID** – The Transformation Map UID references an instance of the Transformation Map structure used to transform source data into the needed Destination Message Definition format.

7.5.2.1.2 *Route Definition*

For this example, the Route Definition structure is described below:

1. **Route Definition UID** – The Route Definition UID uniquely identifies each instance of Route Definition structures.
2. **Route Configuration Data** – The Route Configuration Data can be defined by the TSS implementation or be derived from the integration model as follows:
 - a. Within the integration model, specific transport technology configuration data is defined in the *face.integration.TransportChannel*. The instance of the transport channel and two or more *UoPEndPoints*, representing the source message(s) and destination message(s), are associated with one another through the *face.integration.ViewTransporter*. The TSS route configuration data is derived from the transport channel. The view transporter is a description of the path through the system.

7.5.2.2 *Message Transformation*

Some FACE UoCs may require data in different units and types than other FACE UoCs. The Data Transformation Capability converts data between endpoints to the appropriate format for use by the receiving FACE UoC. The USM defines messages in a manner the TSS understands and can operate on based on configuration information provided.

To perform a transformation, the TSS will need to understand the source message definition as well as the destination message definition as part of its configuration information. If transformations are supported, the TSS Distribution Capability will need to decide whether a

message requires data transformations prior to sending, upon receipt, or not at all and invoke the appropriate transformation capability within the TSS.

In our example, the Transformation Map provides configuration of the transformation capability. A Transformation Map provides the means to define conversions to be used by the TSS on a given instance of the TSS Message Parameter. TSS implementations can optionally support transformation capabilities.

7.5.2.2.1 Transformation Map

For this example, the Transformation Map structure is described below:

1. **Transformation Map UID** – The Transformation Map UID is a unique identifier associated with each instance of transformations defined.
2. **Transformation Type** – The Transformation Type identifies a type of transformation needed to generate the destination message definition. The Transformation Type can be defined by the TSS implementation or be derived from the integration model as follows:
 - a. The integration model defines several types of transformations including aggregation of multiple messages into one message type (*face.integration.ViewAggregation*), filtering out data (*face.integration.ViewFilter*), or to convert data from one view type to another view type (*face.integration.ViewTransformation*). The transformation can be associated prior to being connected to the view transporter (on output of a message from the UoC), prior to being connected to a *UoPInputEndPoint* (on input from the transport technology), or anywhere along the path defined in the system.
 - b. Each transformation type has its own unique entry and is used by itself or in combination with any other transformations to match the destination Message Parameter Instance's definition.
3. **Transformation UUID** – The Transformation UUID is reference to the integration model's definition for the transformation. It is not necessary to implement the TSS with UUIDs; however, the Transformation UUID is provided to assist in integration.

7.5.2.3 Quality of Service

Messages being transported on a route may need a Quality of Service (QoS) policy to be applied to the route. If QoS is supported, the Distribution Capability will need to associate routes to QoS policies and apply specific values to the needed attributes of the QoS policy.

Separate QoS structures allow a system to have policies which can be applied to multiple routes or one policy for each route defined. Different policies can be created such as reliable, secure, or high bandwidth, etc. to align to requirements supported by the underlying transport technologies. Once QoS policies are defined, the policy is associated to a route using the Routing QoS Association structure. This associates a Route Definition instance to a QoS policy instance. There are no associated Integration Model elements with defining QoS other than the QoS attributes will need to be modeled by the TSS implementer.

7.5.2.3.1 QoS Definition

For this example, the QoS definition is described as follows:

1. **QoS UID** – The QoS UID uniquely identifies a QoS Policy.
2. **QoS Policy** – A human-readable name of the policy represented by the QoS Attributes.
3. **QoS Attributes** – The QoS Attributes are used to define QoS Policies. Considered a key, value pair, the attributes of a policy such as “Reliable Delivery” may have attributes of “kind” and “max_blocking_time”.

7.5.2.3.2 Routing QoS Association

For this example, the Routing QoS Association structure is described as follows:

1. **QoS UID** – The QoS UID references an instance of the QoS Definition structure as part of the association of a particular QoS Policy with a Route Definition.
2. **Route Definition UID** – The Route Definition UID references an instance of the Route Definition structure that specifies a particular data transport path.
3. **QoS Attributes Values** – The QoS Attribute Values are the specific values of the QoS parameter which are defined by the QoS Definition entity as referenced by the QoS UID. Examples of QoS Values for a “Reliable Delivery” policy might be “Reliable” and “0” for the attributes “kind” and “max_blocking_time” respectively. The QoS Attribute Values are passed to the UoP in the *qos_parameter* for the *Read_Callback::Callback_Handler* and *TypedTS::Receive_Message* interfaces if QoS is supported.

7.5.2.4 Other Considerations

In addition to associating messages statically at initialization or build time, messages may need to be associated during run-time. Run-time association can assist the Distribution Capability to support TSS capabilities such as providing Message Pattern Translation, route data to a TPM, Data Store, or Framework Service.

8 Injectable Interface Guidance

8.1 Scope

This chapter provides guidance related to the Injectable Interface for System Integrators and UoC developers. For System Integrators, the guidance relates to using the Injectable Interface to resolve interface dependencies between UoCs. For UoC developers, the guidance relates to providing the Injectable Interface. This chapter assumes the reader is familiar with the terms and concepts from the FACE Technical Standard §3.11.3 and §3.11.4.1.

The reader may also be interested in Chapter 9 for related topics.

Section 8.2 addresses “backward compatibility” – changes in the FACE Technical Standard that would impact a UoC that is conformant to a previous edition in becoming conformant to the latest edition. Section 8.3 describes the design goals that motivate the characteristics of the Injectable Interface. Section 8.4 addresses topics on effective use of the Injectable Interface and on implementing the Injectable Interface from the UoC perspective.

8.2 Migrating from Previous Editions

The Injectable Interface is a new feature in FACE Technical Standard, Edition 3.0.

8.3 Design Goals

For any software component interface, there is a role for the interface user and a role for the interface provider, with an inherent dependency between the two roles. The interface user is concerned with calling the interface operations to implement its behavior. The interface provider is concerned with implementing the required behavior of the interface operations. The inherent dependency is that a software component cannot function in an integrated system as an interface user unless it is paired with a corresponding interface provider.

In the FACE Technical Standard, interfaces are declared to separate concerns among UoCs. An interface may exist at a segment boundary of the FACE Reference Architecture, where a UoC cannot cross the boundary. An interface may also be declared within a segment where it is advantageous to allow pluggable or replaceable interface provider behavior by different UoCs, such as the TSS Serialization Interface. In either case, it is generally true that an interface user is packaged in a UoC that is distinct from a UoC packaging an interface provider.

Previous editions of the FACE Technical Standard used programming language mappings where interface declarations had procedural semantics. An interface provider implemented the required behavior of an interface operation at a documented procedure name, and the interface user invoked the interface operation using that procedure name. The pairing of interface user and interface provider was accomplished via resolution of the symbol for that procedure name. This approach limits deployment alternatives in the same partition, where there could be only one interface provider for any given interface (otherwise, there would be multiple definitions of the

same procedure). This constraint might have led the System Integrator to a computing environment solution with otherwise extraneous partitions.

Now the FACE Technical Standard uses programming language mappings where interface declarations have object-oriented semantics. An interface user invokes an interface operation via an instance as an abstract classifier. This introduces a level of indirection helpful to addressing the constraint of one interface provider per partition. The inherent dependency previously described still exists, but the System Integrator can select the interface provider to resolve that dependency. This also means there can be multiple interface providers in the same partition, each with their own concrete classifier.

The pairing of interface user and interface provider is now accomplished via dependency injection, a software engineering idiom where an instance of the concrete classifier from the interface provider is assigned to (or “injected into”) the interface user externally. As this assignment is done across UoCs, the most common use case is for the System Integrator to be the external agent. The dependency injection idiom is realized by the Injectable Interface.

8.4 Injectable Examples

This section provides specific examples of the Injectable Interface being used to resolve the inherent dependency between a UoC as an interface user, one or more UoCs as interface providers, and software provided by the System Integrator as the external agent that performs the dependency injection.

The first example illustrates a Serial I/O Service packaged in an IOSS UoC that is an interface user of the Configuration Interface, and an OSS UoC that is an interface provider of the Configuration Interface. Figure 46 is a UML® class diagram of this example depicting the abstract classifiers defined by the FACE Technical Standard and the concrete classifiers that implement the interface behavior. It is worth emphasizing that when a UoC is an interface user (e.g., an IOSS UoC using the Configuration Interface), it must also be an interface provider of the Injectable Interface bound to that interface.

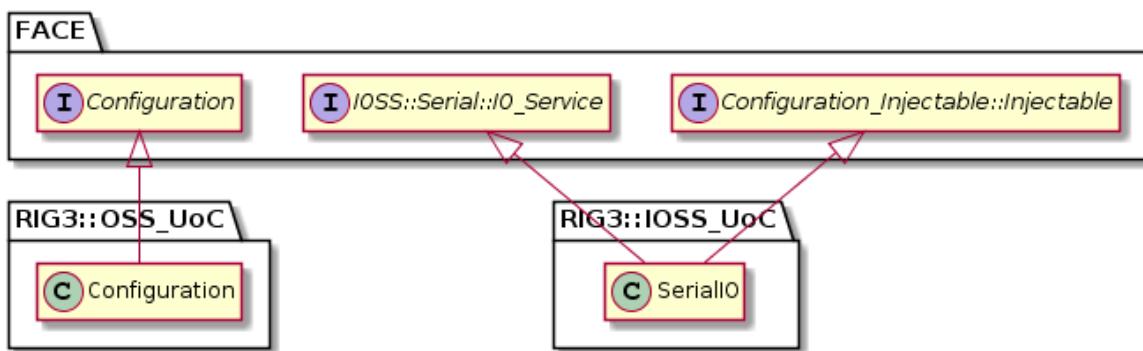


Figure 46: Class Diagram for First Injectable Example

The Injectable Interface expresses the same contract for dependency injection for every interface listed in Table 8 of the FACE Technical Standard. This is accomplished using an IDL template module named *FACE::Injectable* that is parameterized by a type classifier for the interface being assigned via dependency injection. Template modules are described in the FACE Technical Standard §3.14.2. The template module is instantiated to bind a specified interface to the type

classifier. Instantiating a template module creates a new namespace, replacing the template module parameter with the binding type for all interfaces declared in the template module. For this example, the `FACE::LCM::Injectable` template module is instantiated for `FACE::Configuration` to the namespace `FACE::Configuration_Injectable` and it contains an interface named `Injectable` that is now bound to `FACE::Configuration`. The IDL syntax for this template module instantiation is shown below.

```
// Source file: RIG3/IOSS_UoC.Injectable.idl

#include <FACE/Injectable.idl>
#include <FACE/Configuration.idl>

module FACE {
    // A template module instantiation of the Injectable Interface for
    the
    // Configuration Interface, resulting in an IDL interface named
    Injectable
    // in the new module scope.
    module FACE::Injectable<FACE::Configuration>
Configuration_Injectable;
}; // module FACE
```

The above IDL template module instantiation is syntactically equivalent to the IDL represented below.

```
#include <FACE/Injectable.idl>
#include <FACE/Configuration.idl>

module FACE {
    module Configuration_Injectable {
        interface Injectable {
            void Set_Reference (
                in STRING_TYPE interface_name,
                in FACE::Configuration interface_reference,
                in GUID_TYPE id,
                out RETURN_CODE_TYPE return_code);
        };
    };
};
```

When mapped to C++ according to the IDL language mappings, the resulting declaration is given below.

```
namespace FACE {
    namespace Configuration_Injectable {

        class Injectable {
    protected:
        Injectable() {}
    private:
        Injectable(const Injectable&);
        Injectable& operator=(const Injectable&);
    public:
        virtual ~Injectable() {}
        virtual void Set_Reference(
            const FACE::STRING_TYPE& interface_name,
```

```

        const FACE::Configuration& interface_reference,
        FACE::GUID_TYPE id,
        FACE::RETURN_CODE_TYPE::Value& return_code) = 0;
    } // interface_Injectable
}; // namespace Configuration_Injectable
}; // namespace FACE

```

This is an abstract class. Concrete UoC implementations will inherit from this class and implement the *Set_Reference()* method. The parameters in the *Set_Reference()* method are as follows:

- *Interface_name* – either for branch logic or for logging
The UoC developer would document valid and expected values for the name.
- *Interface_reference* – a reference to the actual software instance that is injected
- *id* – a value associated with the software instance that may also be used for branch or switching logic at run-time. One use case for the *id* is when translating between the TS Type-Specific Interface and the TS Type Abstraction Interface.

Below is an example concrete implementation that inherits from the above abstract *Injectable* class.

```

namespace RIG {
    namespace IOSS_UoC {
        class SerialIO : public FACE::IOSS::Serial_IO::IO_Service,
                        public FACE::Configuration_Injectable {

            /** All constructors and implementation of IO Service code
            here **/

            public:
                /**
                 * Sets the internal configuration member variable for use
                 * during configuration.
                 */
                void Set_Reference(const FACE::STRING_TYPE& interface_name,
                                  const FACE::Configuration&
interface_reference,
                                  FACE::GUID_TYPE id,
                                  FACE::RETURN_CODE_TYPE::Value& return_code)
                {

                    configuration = &interface_reference;
                    return_code.value = FACE::NO_ERROR;
                }

            private:
                FACE::Configuration *configuration;
        };
    };
}

```

In this example, the name and id are not used. UoC developers that use these parameters should explicitly document any expectations on the values so a System Integrator can call *Set_Reference()* with expected values.

Figure 47 is a UML sequence diagram depicting the collaboration between the external agent, typically software provided by the System Integrator, and the instances of the various interface users and interface providers. The instance *cfg* is of type *RIG3::OSS_UoC::Configuration* and the instance *serial* is of type *RIG3::IOSS_UoC::SerialIO*.

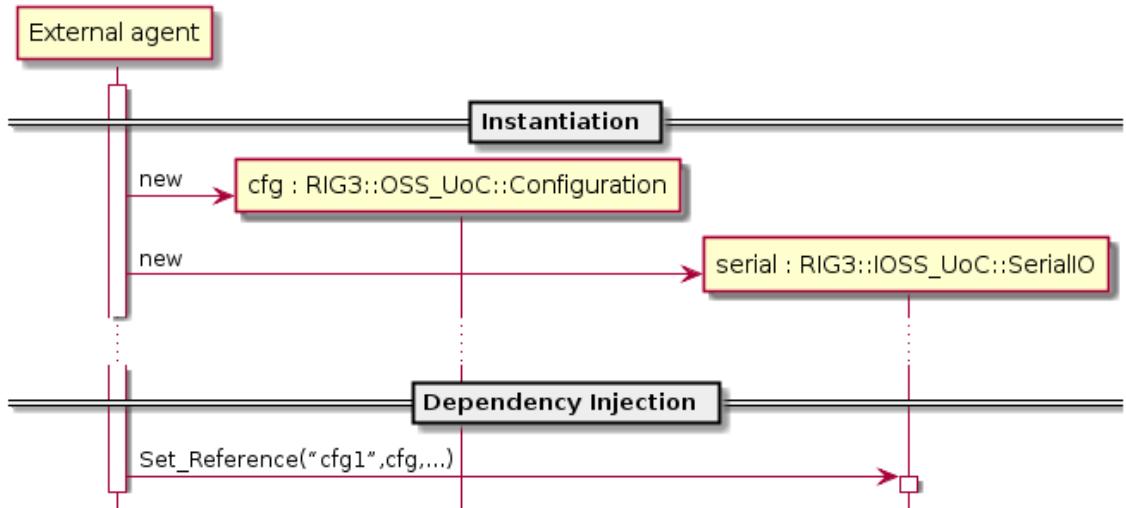


Figure 47: Sequence Diagram for First Injectable Example

To begin, the external agent creates the instances. There may be other behavior following. At a later execution point, the external agent calls the *Set_Reference()* operation to assign the *cfg* instance. This sequence is in shown from the perspective of the external agent in C++ below.

```

// Inside the external agent

// Instantiate concrete implementation of a Configuration
RIG:::OSS_UoC::Configuration cfg;

// Instantiate concrete implementation of the IO_Service
RIG:::IOSS_UoC::SerialIO serial;

// Declare the return code and the guid
FACE::RETURN_CODE_TYPE::Value return_code;
FACE::GUID_TYPE guid_is_unused;

// Inject the configuration into the IO_Service instance
serial.Set_Reference("cfg1", cfg, guid_is_unused, return_code);

// Example complete
  
```

The second example extends the first by adding a PSSS UoC that is an interface user of the Serial I/O Service packaged by the same IOSS UoC. The PSSS UoC is also an interface user of the Configuration Interface. Figure 48 is a UML class diagram depicting only the additional PSSS UoC. The Injectable Interface is instantiated for both the Configuration and the Serial I/O Service Interfaces as described in the first example.

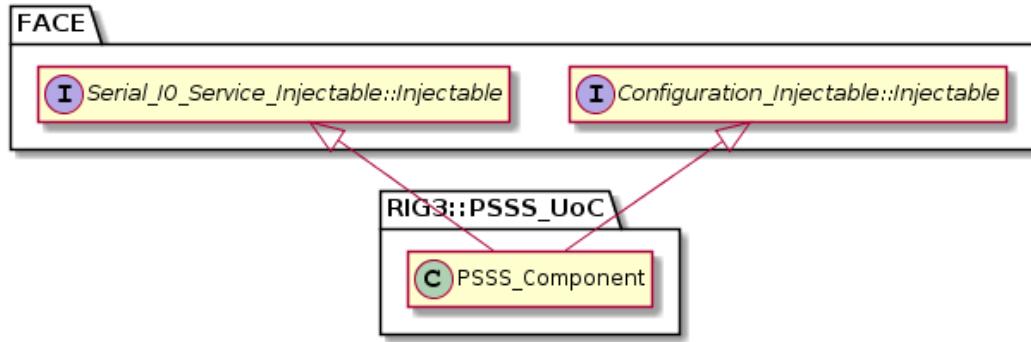


Figure 48: Class Diagram for Second Injectable Example

Figure 49 is a UML sequence diagram depicting the collaboration among the external agent, interface users, and interface providers extended from the first example to include the PSSS UoC. A new instance *psss* of type *RIG3::UoC::PSSS_Component* has been added.

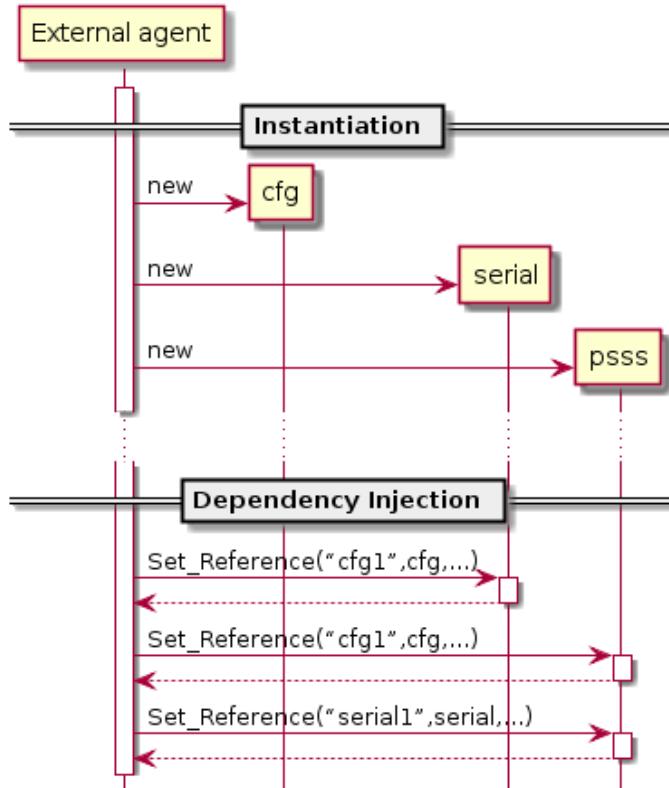


Figure 49: Sequence Diagram for Second Injectable Example

Note the external agent calls *Set_Reference()* to assign the same *cfg* instance to both the *serial* and *psss* instances, showing the System Integrator realizing a deployment with one Configuration Interface provider. The external agent also calls *Set_Reference()* to assign the *serial* instance, illustrating the System Integrator has the flexibility to select the appropriate Serial I/O Service provider for their deployment solution. This sequence is shown from the perspective of the external agent in C++ below.

```

// Inside the external agent

// Instantiate concrete implementation of a Configuration
RIG::OSS_UoC::Configuration cfg;

// Instantiate concrete implementation of the IO_Service
RIG::IOSS_UoC::SerialIO serial;

// Instantiate concrete implementation of the PSSS
RIG::PSSS_UoC::PSSS_Component psss;

// Declare the return code and the guid
FACE::RETURN_CODE_TYPE::Value return_code;
FACE::GUID_TYPE guid_is_unused;

// Inject the configuration instance into the IO_Service instance
serial.Set_Reference("cfg1", cfg, guid_is_unused, return_code);

// Inject the configuration instance into the PSSS instance
psss.Set_Reference("cfg1", cfg, guid_is_unused, return_code);

// Inject the IO_Service instance into the PSSS instance
psss.Set_Reference("serial1", serial, guid_is_unused, return_code);

// Example complete

```

9 Life Cycle Management Services Guidance

9.1 Scope

This chapter provides implementation guidance related to the Life Cycle Management (LCM) Services for System Integrators and Software Suppliers. For System Integrators, the guidance relates to using the interfaces of LCM Services to integrate UoCs that provide those interfaces. For Software Suppliers, the guidance relates to providing each of the interfaces of LCM Services. This section assumes the reader is familiar with the terms and concepts from the FACE Technical Standard §3.13.

Section 9.2 addresses “backward compatibility” – changes in the FACE Technical Standard that would impact a UoC that is conformant to a previous edition in becoming conformant to the latest edition. Section 9.3 describes the design goals that motivate the characteristics of LCM Services. Section 9.4 addresses topics on effective use of LCM Services. Section 9.5 addresses topics on implementing LCM Services from the UoC perspective.

9.2 Migrating from Previous Editions

LCM Services are a new feature in FACE Technical Standard, Edition 3.0.

9.3 Design Goals

LCM Services are used to manage software instances at specific execution points in a program. The management behavior is implemented by an external agent distinct from the managed software instances. The typical use case is the external agent being provided by the System Integrator to realize the system deployment design based on a set of UoCs from Software Suppliers. Figure 50 depicts a sequence of execution points that are addressed by three topics from the FACE Technical Standard working in conjunction: LCM Services, the Injectable Interface, and the programming language mappings.

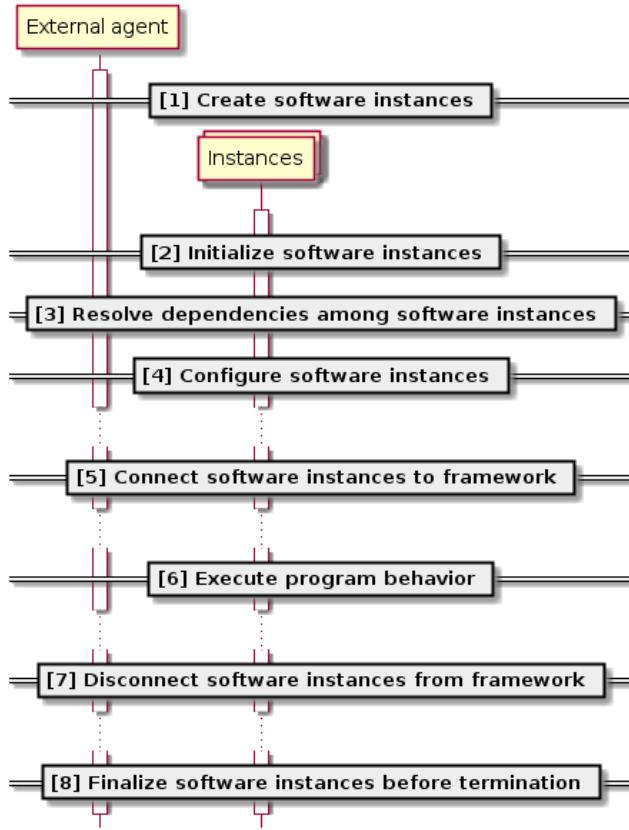


Figure 50: Program Execution Points

Each program execution point represents the preconditions and postconditions that can be assumed by a software instance. A precondition is assumed to be true when a software instance enters the corresponding execution point, while a postcondition is assumed to be true when a software instance enters every following execution point.

1. The creation execution point postcondition is every software instance in the program exists but none have been initialized.
2. The initialization execution point postcondition is every software instance has had opportunity to acquire resources that do not rely upon knowledge of other software instances. The **Initializable Interface** corresponds to this execution point.
3. The dependency resolution execution point postcondition is every software instance has been paired with other software instances and is able to call all **FACE Interface** operations. The **Injectable Interface** corresponds to this execution point. More information on the **Injectable Interface** is available in Chapter 8.
4. The configuration execution point postcondition is every software instance has had opportunity to acquire resources that rely upon knowledge of other software instances, including calls to **FACE Interface** operations. The **Configurable Interface** corresponds to this execution point.
5. The framework connection execution point precondition is that the program has bootstrapped a component framework to the point of connecting a software instance. The

postcondition is that every software instance that supports framework behavior has been connected. The Connectable Interface corresponds to this execution point.

6. The execute execution point precondition is every software instance is ready to transition to its nominal operating state. A software instance may transition among its supported states while at this execution point. The Stateful Interface is available for these transitions. The postcondition is every software instance has transitioned to its terminal operating state.
7. The framework disconnection execution point postcondition is every software instance that supports framework behavior has been disconnected and the program is free to tear down its component framework. The Connectable Interface also supports this execution point.
8. The finalization execution point preconditions are the software instance has transitioned to its terminal operating state and it cannot access Framework Services. The software instance can perform terminal behavior including calling FACE Interface operations. The postcondition is every software instance has had opportunity to complete its nominal finalization behavior and can be destroyed. The Initializable Interface also supports this execution point.

The FACE Technical Standard §3.14 describes how software is represented as classifiers for each of the supported programming languages, and then relies upon programming language syntax for creating and destroying an instance of that classifier.

The Software Supplier documents which software instances support one or more of the LCM Services interfaces in the UoC's Software Design Description or similar design artifacts available to the System Integrator. From the System Integrator's perspective, the ideal circumstance is that each software instance in the system deployment design supports all of the LCM Services. This circumstance allows the external agent to interact with the set of software instances collectively as often as possible.

The Stateful Interface is designed to support state representations that vary among UoCs integrated into the same system. This characteristic allows a Software Supplier to design a state machine of appropriate complexity for a UoC and avoids the alternative of a universal state representation for software components. The state representations are reflected in the UoC's UoPModel to benefit the System Integrator.

The Stateful Interface supports a distinction between the set of states that the external agent can use to request state transitions and the set of states that can be reported by the software instance when queried. This flexibility does not imply that the software instance maintains two separate state machines. Rather, it allows the software instance to report states that are not logical transitions from the external agent, such as degraded or error states. When there is no need make such a distinction, the Stateful Interface can be supported with the same state representation for both.

9.4 Examples for Using LCM Services

This section provides examples showing how integrating a system comprised of software components that provide LCM Services simplifies basic life-cycle activities.

9.4.1 Multi-Component Startup and Shutdown

In this example, the System Integrator is working with a PSSS UoC, an IOSS UoC that packages a Serial I/O Service Interface provider, and an OSS UoC that packages a Configuration Interface provider. Figure 51 is a UML class diagram that depicts the relationships between the classifiers from the UoCs and the classifiers from the Initializable, Configurable, and Connectable Interfaces from LCM Services.

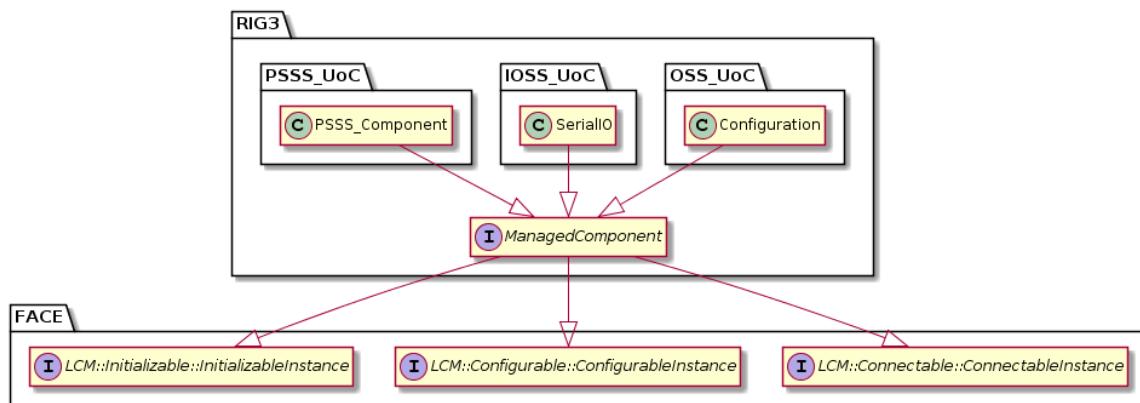


Figure 51: Class Diagram for LCM Services Example of Startup and Shutdown

Note in this example the abstract classifier *ManagedComponent* uses multiple inheritance to support the Initializable, Configurable, and Connectable Interfaces. The classifiers from each UoC inherit from this common abstract classifier. This design strategy is beneficial because the System Integrator can implement the management behavior of the external agent as loops over a collection of *ManagedComponent* instances, but it requires coordination among the System Integrator and UoC Software Suppliers. When the UoC classifiers do not share an abstract classifier, the same management behavior requires more lines of code that unroll the loops to support the varied types.

Figure 52 is a UML sequence diagram depicting the behavior through the first three program execution points.

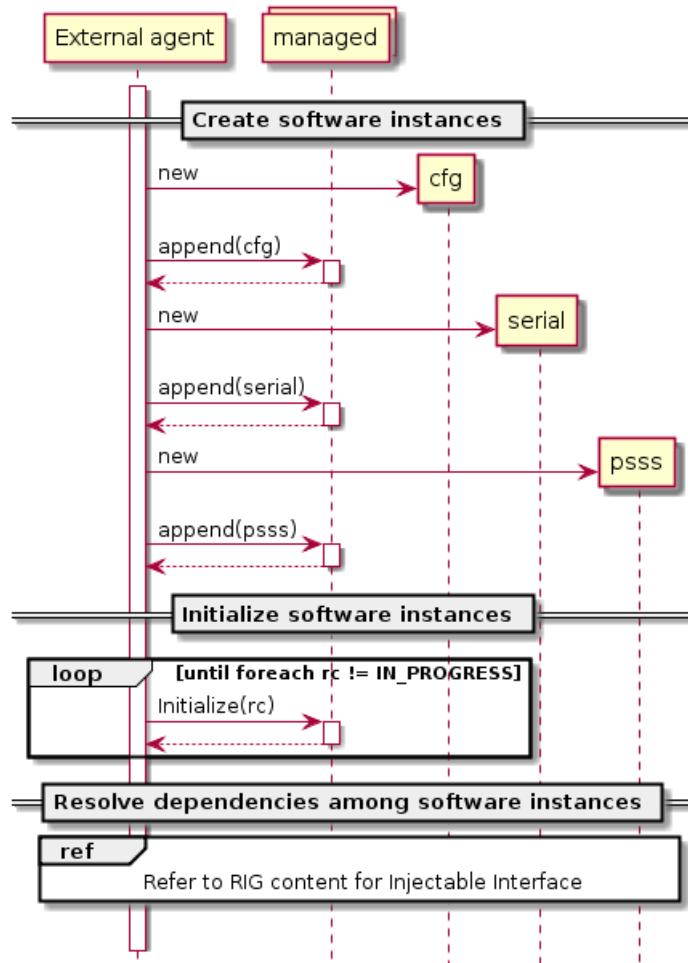


Figure 52: Sequence Diagram for Beginning Program Execution Points

The software instances are created in accordance with the syntax for the target programming language. Note that each instance is appended to a collection named *managed* that is used by the external agent for the loop-based management behavior described earlier, as illustrated for the *Initialize()* operation. The loop condition is to repeat until each call to *Initialize()* returns something other than *IN_PROGRESS*, meaning each software instance returned either a nominal value or an error value. The return codes *NO_ERROR* and *NO_ACTION* are nominal values, meaning the postcondition for this execution point is valid. The return codes *NOT_AVAILABLE* and *INVALID_CONFIG* are error values that the external agent must address with the appropriate fault management behavior to preserve the postcondition. Once the software instances have been initialized, the dependency relationships among them are resolved by the external agent using the Injectable Interface.

Figure 53 is a UML sequence diagram depicting the same loop-based management behavior performed by the external agent at other program execution points. Refer to Section 9.4.2 for a detailed discussion of state management at the execute execution point.

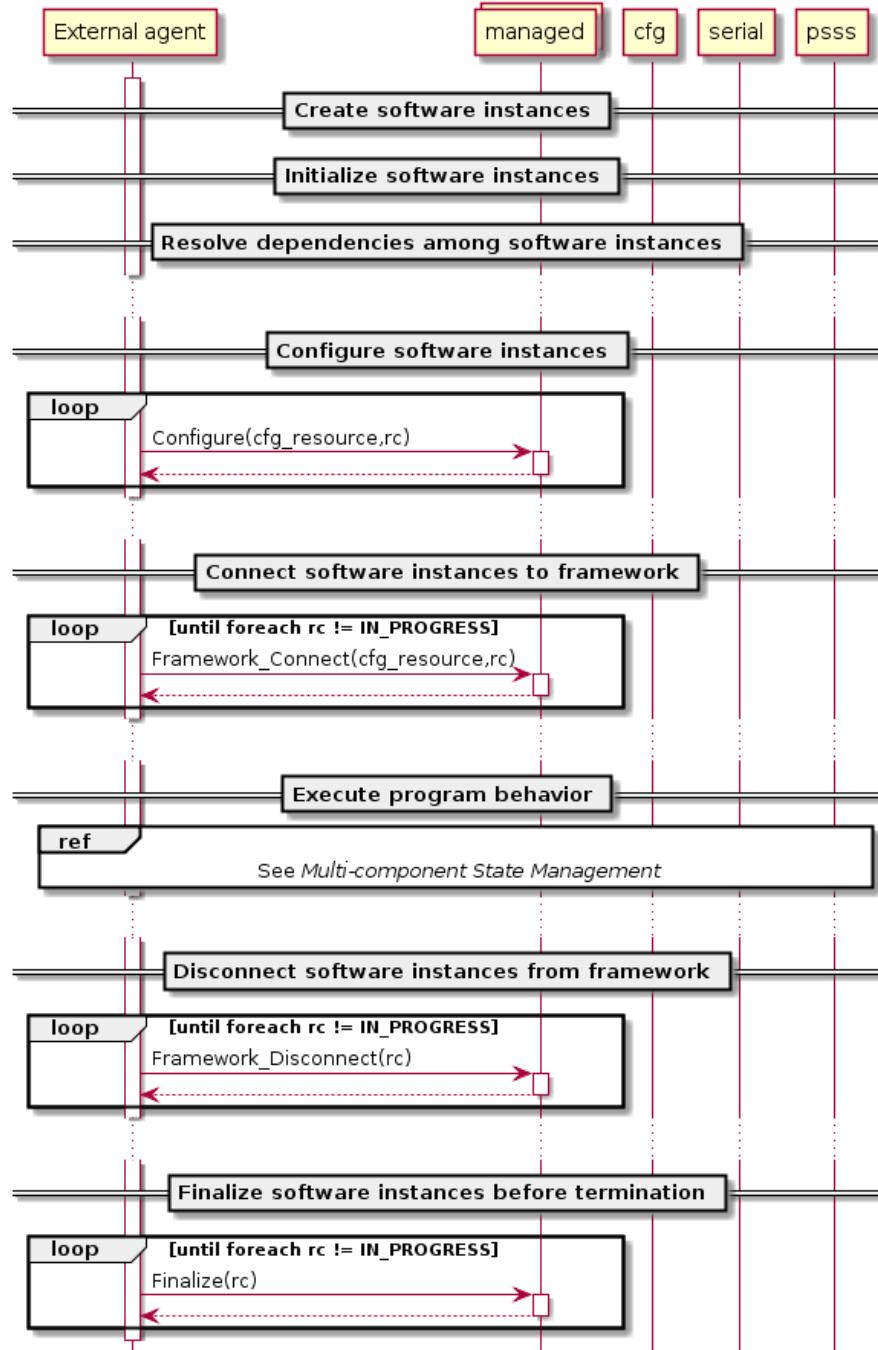


Figure 53: Sequence Diagram for Remaining Program Execution Points

Note that using the loop-based management behavior means the external agent does not deal directly with the UoC software instances, although they remain instantiated. Within each loop is logic to deal with nominal and error return values to preserve the postcondition associated with the corresponding execution point. Note also that the loop for *Configure()* does not have an IN_PROGRESS loop condition because it is a blocking operation.

9.4.2 Multi-Component State Management

This section extends the prior example to illustrate state management at the execute execution. Figure 54 is a UML class diagram that depicts the relationships between the classifiers from the UoCs and the classifiers from the Stateful Interface from LCM Services.

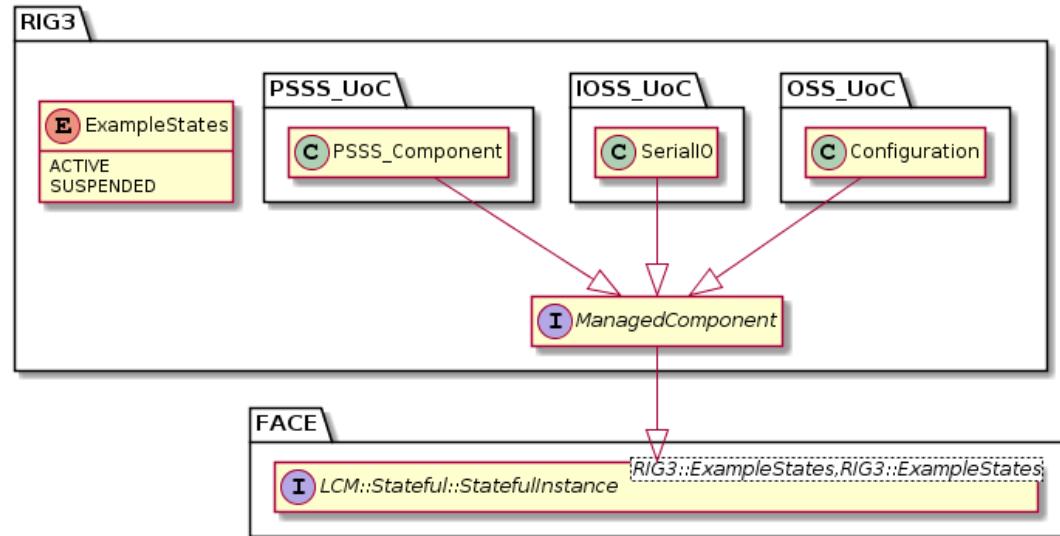


Figure 54: Class Diagram for LCM Services Example of State Management

This example declares a simple two-state representation named *RIG3::ExampleStates*. This state representation is used for both the requested and queried states of the Stateful Interface, and is shared among the UoCs. The Stateful Interface supports a system deployment design with UoCs that have differing state representations, even those that have different state representation between requested and queried states, where the System Integrator develops algorithms to map among the differing state representations and transitions.

Figure 55 is a UML sequence diagram showing the straightforward state management behavior that this example supports. Note that using the loop-based management behavior and the common state representation means the external agent does not deal directly with the UoC software instances, although they remain instantiated.

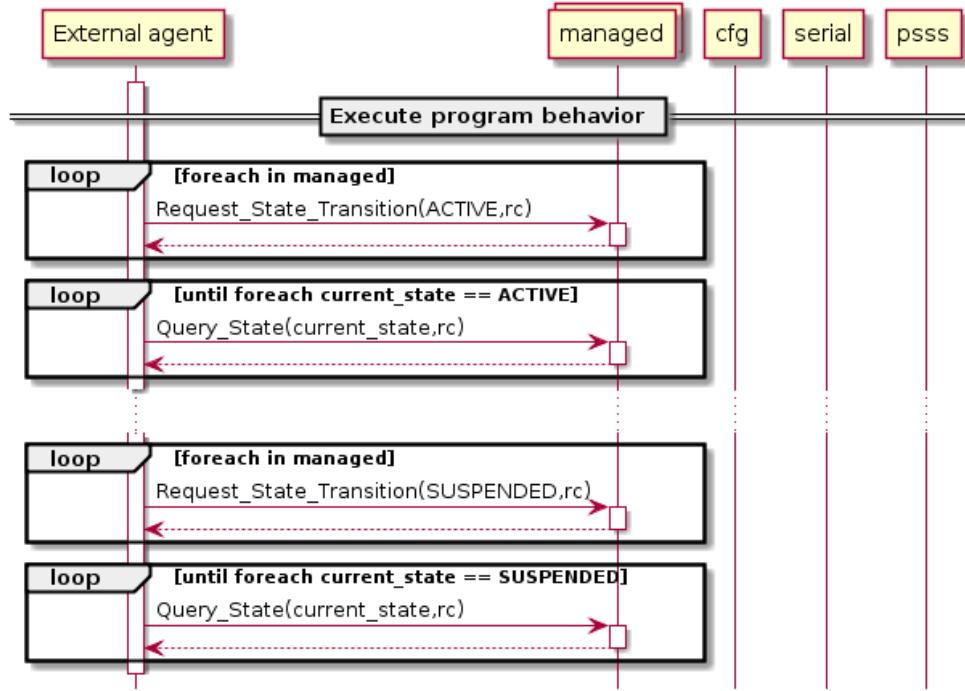


Figure 55: Sequence Diagram for LCM Services Example of State Management

The external agent asks each software instance to transition to its ACTIVE state by calling *Request_State_Transition()*, which may be an asynchronous operation. To ensure that all software instances have transitioned successfully, the external agent ensures all software instances are ACTIVE by looping until each software instance has returned ACTIVE from *Query_State()*. At this point, the external agent begins scheduling execution of software instances. Some condition triggers the external agent to transition all software instances to SUSPENDED, where the same algorithm ensures the execute execution point postcondition that all software instances have transitioned to their terminal operating states.

The Software Supplier describes the complete state machine in the UoC's Software Design Description or similar design artifacts available to the System Integrator. The intent is there is no indeterminate state; the instance reports that it is in one state until it is completely in a new state. When the behavior of the software component depends upon proper support for an indeterminate state, that state should be explicitly represented in the state model.

9.5 Examples for Providing LCM Services

This section addresses topics specific to a Software Supplier when developing software components that provide one or more of the interfaces of LCM Services.

9.5.1 Declarations for a Managed Component

This section illustrates the declarations necessary to provide LCM Services interfaces. IDL is used here as one alternative to illustrate the concept clearly. A Software Supplier can declare their software component directly in its programming language using multiple interface inheritance as described in the FACE Technical Standard §3.14. Note that each of the LCM Services interfaces is optional, while this example software component provides all.

```

// Source file: RIG3/LCM/ExampleManagedComponent.idl

#ifndef __RIG3_LCM_EXAMPLEMANAGEDCOMPONENT
#define __RIG3_LCM_EXAMPLEMANAGEDCOMPONENT

#include <FACE/LCM/Initializable.idl>
#include <FACE/LCM/Configurable.idl>
#include <FACE/LCM/Connectable.idl>
#include <FACE/LCM/Stateful.idl>

module FACE {
    module DM {
        module RIG3 {
            module LCM {
                // A simple state representation for the example.
                enum EXAMPLE_STATES { ACTIVE, SUSPENDED };
            }; // module LCM
        }; // module RIG3
    }; // module DM
}; // module FACE

module RIG3 {
    module LCM {
        // A simple state representation for the example.
        enum EXAMPLE_STATES { ACTIVE, SUSPENDED };

        // A template module instantiation for the LCM Stateful Interface
        // using the example state representation.
        module FACE::LCM::Stateful<FACE::DM::RIG3::LCM::EXAMPLE_STATES,
                                         FACE::DM::RIG3::LCM::EXAMPLE_STATES>
            ExampleStateful;

        // Use IDL multiple inheritance to declare the example software
        component.
        interface ExampleManagedComponent :
            FACE::LCM::Initializable::InitializableInstance,
            FACE::LCM::Configurable::ConfigurableInstance,
            FACE::LCM::Connectable::ConnectableInstance,
            ExampleStateful::StatefulInstance {
        }; // interface ExampleManagedComponent
    }; // module LCM
}; // module RIG3

#endif // ! __RIG3_LCM_EXAMPLEMANAGEDCOMPONENT

```

9.5.2 Definitions for a Managed Component

This section describes guidance for the operation definitions of the declarations in the previous section. Refer back to Section 9.3 for the list of supported execution points and the precondition/postcondition criteria that each operation should preserve.

The software component *ExampleManagedComponent* inherits two operations from the *InitializableInstance* interface: *Initialize()* and *Finalize()*. *Initialize()* corresponds to execution point #2 and *Finalize()* corresponds to execution point #8.

The *Configure()* operation, corresponding to execution point #4, is inherited from the *ConfigurableInstance* interface. It is recommended to use Configuration Services to read configuration parameters, as illustrated in Figure 56.

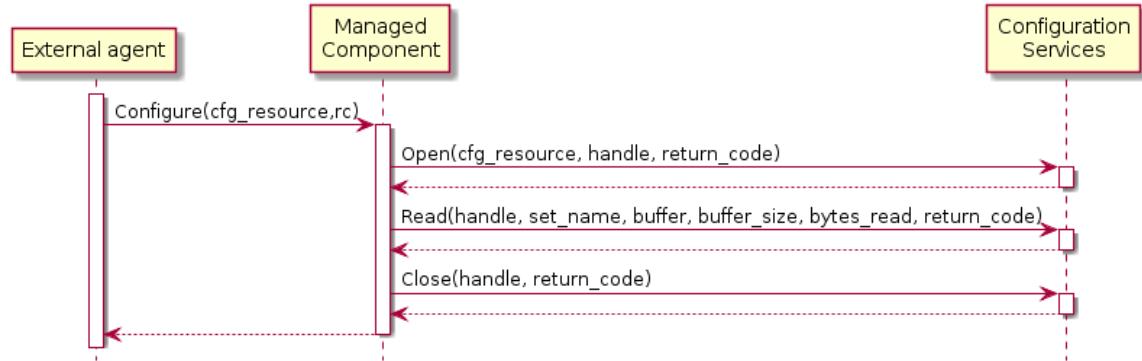


Figure 56: Sequence Diagram for Using Configuration Services in LCM Services

The operations *Framework_Connect()* and *Framework_Disconnect()* are inherited from the *ConnectableInstance* interface. These operations correspond to execution points #5 and #7, respectfully. No guidance is provided for the definition of these operations.

The operations *Query_State()* and *Request_State_Transition()* are inherited from the *ExampleStateful::StatefulInstance* interface. Both operations correspond to execution point #6.

9.5.3 Data Model for State Representation

Note that from the declaration of the Stateful Interface in Section 9.5.1 distinct state representations are possible for REQUESTED_STATE_VALUE_TYPE and REPORTED_STATE_VALUE_TYPE. The state representations referenced by the Stateful Interface are required to be represented by a Platform Data Model View. This View element is then referenced by the lcmPort association of a UnitOfPortability element in the UoPModel.

10 Programming Language Mapping Rules Guidance

This chapter provides guidance for implementing the programming language mapping rules in the FACE Technical Standard §3.14. Guidance is organized by programming language, then by guidance topic. Topics are independent unless explicitly related.

10.1 C Mapping Rules Guidance

No guidance is provided for implementing the C programming language mapping rules in the FACE Technical Standard §3.14.7.

10.2 C++ Mapping Rules Guidance

Guidance topics in this section address the C++ programming language mapping rules in the FACE Technical Standard §3.14.8.

10.2.1 C++ Union Implementation

The FACE Technical Standard §3.14.8.8.3 describes the interface declaration for a C++ union mapped from an IDL enumeration. The Software Supplier must provide definitions for the member functions identified in the mapping rule. The definition must implement the behavior described in the rule. The implementation is responsible for managing the storage necessary to contain the underlying union representation in order to preserve the value type semantics of a union instance, including when copying the union value in the copy constructor and copy-assignment operator.

10.3 Ada Mapping Rules Guidance

Guidance topics in this section address the Ada programming language mapping rules in the FACE Technical Standard §3.14.9.

10.3.1 IDL Module to Ada Packages

This section provides guidance and examples for the mapping rules in the FACE Technical Standard §3.14.9.3.

When feasible, the implementation should use direct mappings rather than indirect mappings and should minimize the length of the declaration sequences in the indirect mappings.

10.3.1.1 Example 1: Simple IDL Module

This example illustrates a direct mapping from an IDL module to an Ada package.

```
// IDL
module A {
    typedef long Foo;
```

```

        const Foo KA1 = 100;
    }
-- Ada
with FACE;
package A is
    subtype Foo is FACE.Long;
    KA1 : constant Foo := 100;
end A;

```

10.3.1.2 Example 2: IDL Modules with Nesting and Reopening

This example illustrates an indirect mapping from IDL modules to Ada packages.

```

// IDL
module A {
    typedef long Foo;
    const Foo KA1 = 100;
}
module A {
    module B {
        typedef Foo Bar;
        const Foo KB = KA1 + 1;
    }
    const B.Bar KA2 = KA1 + B.KB;
    typedef B.Bar Bar1;
}

-- Ada (showing the range of mappings)
-- The notation <<T>> for a (sub)type T denotes any (sub)type
-- equivalent to T. The notation <<expr>> for a constant expression
-- expr denotes any expression or constant object equivalent to expr.
-- The notation <<obj(expr)>> for a constant expression expr denotes
-- any constant object equivalent to expr (thus a declared object and
-- not the expression expr). In the mapping of constant declarations,
-- both a direct mapping and an alternative indirect mapping are shown,
-- with the latter as a comment. The equivalent entities referenced in
-- the declarations are declared in auxiliary packages constructed by
-- the IDL-to-Ada compiler.

with FACE, ...; -- ..." is a list of compilation units, identifying
                  -- the auxiliary packages that embody the indirect
                  -- mappings
package A is
    subtype Foo is <<FACE.Long>>;
    KA1 : constant <<Foo>> := <<100>>;
-- KA1 :           <<Foo>> renames <<obj(100)>>;
    KA2 : constant <<A.B.Bar>> := <<KA1 + A.B.KB>>;
-- KA2 :           <<A.B.Bar>> renames <<obj(KA1 + A.B.KB)>>;
    subtype Bar1 is <<A.B.Bar>>;
end A;
package A.B is
    subtype Bar is <<Foo>>;
    KB : constant <<Foo>> := <<KA1 +1>>
-- KB :           <<Foo>> renames <<obj(KA1 +1)>>;
end A.B;

```

Here are the user-visible packages that result from one possible mapping; these depend on several auxiliary packages, but the user does not need to be concerned with their details.

```
with IDL_Modules.A.IDL_1_1, IDL_Modules.A.IDL_2_1_B,
IDL_Modules.A.IDL_2_2;
package A is
    subtype Foo is IDL_Modules.A.IDL_1_1.Foo;
    KA1 : Foo renames IDL_Modules.A.IDL_1_1.KA1;
    KA2 : IDL_Modules.A.IDL_2_1_B.Bar renames IDL_Modules.A.IDL_2_2.KA2;
    subtype Bar1 is IDL_Modules.A.IDL_2_2.Bar1;
end A;

with IDL_Modules.A.IDL_1_1, IDL_Modules.A.IDL_2_1_B;
package A.B is
    subtype Bar is IDL_Modules.A.IDL_2_1_B.Bar;
    KB : IDL_Modules.A.IDL_1_1.Foo renames IDL_Modules.A.IDL_2_1_B.KB;
end A.B;
```

Here is a skeletal piece of application code.

```
with FACE, A, A.B;
procedure Proc is
    use type FACE.Long;
    X : A.Foo;
    Y : A.B.Bar;
    Z : FACE.Long := A.KA1 + A.KA2 + A.B.KB;
begin
    null;
end Proc;
```

Here are the auxiliary packages that implement the indirect mappings. The content of these packages depends on the implementation of the IDL-to-Ada compiler and does not affect user code.

```
package IDL_Modules is end IDL_Modules;

package IDL_Modules.A is end IDL_Modules.A;

with FACE;
package IDL_Modules.A.IDL_1_1 is
    subtype Foo is FACE.Long;
    KA1 : constant Foo := 100;
end IDL_Modules.A.IDL_1_1;

with FACE, IDL_Modules.A.IDL_1_1;
package IDL_Modules.A.IDL_2_1_B is
    use FACE;
    subtype Bar is IDL_Modules.A.IDL_1_1.Foo;
    KB : constant IDL_Modules.A.IDL_1_1.Foo := IDL_Modules.A.IDL_1_1.KA1
+ 1;
end IDL_Modules.A.IDL_2_1_B;

with FACE, IDL_Modules.A.IDL_1_1, IDL_Modules.A.IDL_2_1_B;
package IDL_Modules.A.IDL_2_2 is
    use FACE;
    KA2 : constant IDL_Modules.A.IDL_2_1_B.Bar :=
        IDL_Modules.A.IDL_1_1.KA1 + IDL_Modules.A.IDL_2_1_B.KB;
    subtype Bar1 is IDL_Modules.A.IDL_2_1_B.Bar;
```

```
end IDL_Modules.A.IDL_2_2;
```

10.4 Mapping Rules Guidance

No guidance is provided for implementing the Java programming language mapping rules in the FACE Technical Standard §3.14.10.

A Acronyms

The following acronyms are used within this Guide:

| Acronym | Definition |
|---------|---|
| AEP | Application Environment Profile |
| API | Application Programming Interface |
| ARINC | Aeronautical Radio Inc. |
| ARP | Aerospace Recommended Practices |
| BIT | Built-In Test |
| BSP | Board Support Package |
| CC | Common Criteria |
| CCA | Clinger-Cohen Act |
| CCD | Component Configurability Definition |
| CCS | Component Configuration Set |
| CDRL | Contract Data Requirements Lists |
| CNSS | Committee of National Security Systems |
| CPID | Configuration Parameter IDentifiers |
| COE | Common Operating Environment |
| CORBA | Common Object Request Broker Architecture |
| COTS | Commercial Off-The-Shelf |
| CR | Change Request |
| CSEP | Configuration Set Encoding Package |
| CSID | Configuration Set Identifier |
| CSIL | Configuration Set Identifier List |
| CSP | Component State Persistence |
| CTS | Clear To Send |

| Acronym | Definition |
|----------------|---|
| CTS | Conformance Test Suite |
| DDS | Data Distribution Service |
| DMA | Dynamic Memory Access |
| DO | Document |
| DoDI | Department of Defense Instruction |
| DPM | Device Protocol Mediation |
| EIA | Electronic Industries Alliance |
| FACE | Future Airborne Capability Environment |
| FIFO | Firs In First Out |
| FISMA | Federal Information Security Management Act |
| FS | Framework Services |
| FSC | Framework Support Capability |
| GP | General Purpose |
| GPS | Global Positioning System |
| HMFM | Health Monitoring/Fault Management |
| I/O | Input/Output |
| IA | Information Assurance |
| ID | Identification, Identifier |
| IDL | Interface Definition Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IOMM | I/O Message Model |
| IOS | Input/Output Services |
| IOSS | Input/Output Services Segment |
| IPC | Inter-Process Communication |

| Acronym | Definition |
|----------------|--|
| ISO/IEC | International Organization for Standardization/International Electrotechnical Commission |
| IT | Information Technology |
| JNI | Java Native Interface |
| JVM | Java Virtual Machine |
| LCM | Life Cycle Management |
| MCDU | Multi-purpose Control and Display Unit |
| MIL-STD | Military Standard |
| MMU | Memory Management Unit |
| NSA | National Security Agency |
| NSTISSP | National Security Telecommunications and Information Systems Security Policy |
| OMG | Object Management Group |
| OS | Operating System |
| OSS | Operating System Segment |
| PCS | Portable Components Segment |
| POSIX | Portable Operating System Interface |
| PSSS | Platform-Specific Services Segment |
| QoS | Quality of Service |
| RFC | Request for Comments |
| RTCA | Radio Technical Commission for Aeronautics |
| RTOS | Real-Time Operating System |
| SAE | Society of Automotive Engineers |
| SCA | Software Communications Architecture |
| SSE | Systems Security Engineering |
| TACAN | Tactical Air Navigation |
| TLS | Transport Layer Security |

| Acronym | Definition |
|----------------|----------------------------|
| TPM | Transport Protocol Module |
| TS | Transport Services |
| TSS | Transport Services Segment |
| UA | User Application |
| UML | Unified Modeling Language |
| UoC | Unit of Conformance |
| UoP | Unit of Portability |
| USM | UoP Supplied Model |
| XMI | XML Metadata Interchange |
| XML | eXtensible Markup Language |
| XSD | XML Schema Definition |
| XSI | X/Open System Interface |

Index

| | | | |
|---------------------------------------|------------|---|----------|
| access permission | 32 | Injectable Interface..... | 276 |
| Ada | 208 | inline assembly..... | 207 |
| alias name | 32 | integer size | 205 |
| aperiodic process | 13 | inter-partition communications | 23 |
| application-detected faults..... | 179 | IOMM | 212 |
| ARINC 653 Health Services | 176 | IOSS | 2, 212 |
| ARINC 653 processes | 12 | IPv4/IPv6 | 33 |
| ARINC 653 Specification | 166 | Java | 210 |
| BIT | 178 | JNI..... | 210 |
| blackboard | 26 | JVM..... | 210 |
| BSP..... | 177 | LCM Services | 246, 283 |
| buffer | 26 | memory allocation..... | 28 |
| C and C++ | 205 | memory deallocation..... | 28 |
| C++..... | 208 | message queue..... | 27 |
| CCD..... | 192 | MMU..... | 9 |
| CCS | 195 | multicore | 9 |
| COE..... | 1 | multi-threading..... | 12 |
| Computing Platform Health Manager | 180 | mutex..... | 26, 27 |
| conditional compilation | 206 | networking | 33 |
| configurable parameter..... | 193 | non-volatile storage | 186 |
| configuration categories | 189 | OSS | 2, 3 |
| Configuration Services API..... | 189 | periodic process..... | 13 |
| configuration services artifacts..... | 192 | portability issues | 203 |
| conformance testing | 202 | POSIX API usage..... | 152 |
| COTS..... | 268 | POSIX constants | 45 |
| counting semaphore..... | 26, 27 | POSIX header files..... | 42 |
| CPID..... | 193 | POSIX multiprocess | 11 |
| CSEP | 192, 198 | POSIX standard..... | 35 |
| CSIL | 192, 196 | POSIX threads..... | 13, 40 |
| CSP..... | 268 | programming language mapping rules | 293 |
| CTS | 42, 202 | programming language run-time | 201, 202 |
| dependency injection | 277 | PSE 51-54 | 35 |
| Distribution Capability | 272 | QoS | 274 |
| DMA | 9 | RFC | 33 |
| DPM | 223 | SCA standard | 153, 166 |
| endianness | 204 | scheduling | 41 |
| event | 26 | shared memory | 21 |
| FACE approach | 1 | thread concurrency | 40 |
| FACE HMF Services | 176 | time management | 33 |
| FACE::Configuration | 214 | TPM | 266 |
| fault handler..... | 186 | TSS | 2, 249 |
| file system volume..... | 31 | TSS Callback Handler | 260 |
| FSC..... | 268 | TSS Create Connection | 264 |
| hardware-detected faults..... | 179 | TSS Data Store | 268 |
| HMF..... | 2, 34, 176 | TSS Initialize..... | 261 |
| HMF event flow..... | 182 | TSS Message Serialization..... | 267 |
| HMF implementation..... | 183 | TSS Receive | 257 |
| HMF layering..... | 178 | | |
| IDL | 229 | | |

| | | | |
|------------------------------|-----|-----------|-----|
| TSS Send..... | 254 | UoP | 3 |
| TSS Send Message Async | 269 | USM..... | 270 |
| unions | 209 | XML..... | 20 |
| UoC | 2 | | |