

**NOT MEASUREMENT
SENSITIVE**



NASA TECHNICAL STANDARD

National Aeronautics and Space Administration

NASA-STD-4009A

**Approved: 2018-03-14
Superseding NASA-STD-4009
(Baseline)**

**SPACE TELECOMMUNICATIONS RADIO SYSTEM (STRS)
ARCHITECTURE STANDARD**

NASA-STD-4009A

DOCUMENT HISTORY LOG

Status	Document Revision	Change Number	Approval Date	Description
Baseline			2014-06-05	NASA-STD-4009 is based on NASA/TM—2010-216809.
Revision	A		2018-03-14	<p>Significant changes were made to this NASA Technical Standard. It is recommended that it be reviewed in its entirety before implementation.</p> <p>Key changes were: Removed all precondition and postcondition to a known state; removed configuration file processing; changed methods to only process one property at a time; removed the query all functionality; changed the requirements for C and C++ for functional equivalence; associated types with constants; STRS Device standardization with DEV_Open, DEV_Close, DEV_Load, DEV_Unload; added new requirements, methods, types, and constants; and deprecated methods, types and constants.</p>

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

FOREWORD

This NASA Technical Standard is published by the National Aeronautics and Space Administration (NASA) to provide uniform engineering and technical requirements for processes, procedures, practices, and methods that have been endorsed as standard for NASA programs and projects, including requirements for selection, application, and design criteria of an item.

This NASA Technical Standard is approved for use by NASA Headquarters and NASA Centers and Facilities, and may be cited in contract, program, and other Agency documents as a technical requirement. It may also apply to the Jet Propulsion Laboratory (a Federally Funded Research and Development Center (FFRDC)), other contractors, recipients of grants and cooperative agreements, and parties to other agreements only to the extent specified or referenced in applicable contracts, grants, or agreements.

This NASA Technical Standard establishes a description of an architecture standard for NASA space communication radio systems. This architecture is a required standard for communication radio system developments among NASA space missions. Although the architecture was defined to support space-based platforms, the architecture is applicable to ground station radios.

This NASA Technical Standard strives to provide commonality among NASA radio developments to take full advantage of emerging software-defined radio technologies from mission to mission. This architecture serves as an overall framework for the design, development, operation, and upgrade of these software-based radios.

Requests for information should be submitted via “Feedback” at <https://standards.nasa.gov>. Requests for changes to this NASA Technical Standard should be submitted via MSFC Form 4657, Change Request for a NASA Engineering Standard.

Original signed by
Ralph R. Roe, Jr.
NASA Chief Engineer

03/14/2018
Approval Date

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
DOCUMENT HISTORY LOG.....	2
FOREWORD.....	3
TABLE OF CONTENTS.....	4
LIST OF APPENDICES	6
LIST OF FIGURES	7
LIST OF TABLES	8
 1. SCOPE	 11
1.1 Purpose.....	11
1.2 Executive Summary	12
1.2.1 Key Architecture Requirements.....	12
1.2.2 STRS Overview	13
1.2.3 Roles and Responsibilities	14
1.2.4 Background.....	16
1.3 Applicability	17
1.4 Tailoring.....	17
 2. APPLICABLE DOCUMENTS.....	 18
2.1 General.....	18
2.2 Government Documents	18
2.3 Non-Government Documents	18
2.4 Order of Precedence.....	18
 3. ACRONYMS, ABBREVIATIONS, AND DEFINITIONS	 19
3.1 Acronyms and Abbreviations	19
3.2 Definitions	22
 4. HARDWARE ARCHITECTURE.....	 30
4.1 Generalized Hardware Architecture and Specification	31
4.1.1 Components	34
4.1.2 Functions.....	34
4.1.3 Interfaces.....	35
4.1.3.1 External Interfaces	35
4.1.3.2 Networking	37
4.1.3.3 Internal Interfaces	37
4.2 Module Type Specification.....	38

TABLE OF CONTENTS (Continued)

<u>SECTION</u>	<u>PAGE</u>
4.2.1 General Purpose Processing Module	38
4.2.1.1 GPM Components.....	39
4.2.1.2 GPM Functions	40
4.2.1.3 GPM Interfaces	40
4.2.1.4 GPM Requirements.....	41
4.2.2 Signal-Processing Module	41
4.2.2.1 SPM Components	42
4.2.2.2 SPM Functions.....	42
4.2.2.3 SPM Interfaces.....	43
4.2.3 Radio Frequency Module.....	44
4.2.3.1 RFM Functions	45
4.2.3.2 RFM Components.....	46
4.2.3.3 RFM Interface.....	46
4.2.3.4 RFM Requirements.....	46
4.2.4 Security Module	46
4.2.5 Networking Module.....	47
4.2.6 Optical Module	47
4.3 Hardware Interface Description.....	47
4.3.1 Control and Data Interface.....	49
4.3.2 Operating Power Interface	49
4.3.3 Thermal Interface and Power Consumption	50
5. APPLICATIONS	51
5.1 Application Implementation	51
5.2 Application Selection.....	52
5.3 Application Repository Submissions.....	52
6. CONFIGURABLE HARDWARE DESIGN ARCHITECTURE.....	54
6.1 Specialized Hardware Interfaces	55
7. SOFTWARE ARCHITECTURE.....	57
7.1 Software Layer Interfaces	57
7.2 Infrastructure.....	65
7.3 STRS APIs	66
7.3.1 STRS Application-Provided Application Control API.....	66
7.3.2 STRS Infrastructure-Provided Application Control API.....	76

NASA-STD-4009A

TABLE OF CONTENTS (Continued)

<u>SECTION</u>	<u>PAGE</u>
7.3.3 STRS Infrastructure Application Setup API.....	81
7.3.4 STRS Infrastructure Data Sink	86
7.3.5 STRS Infrastructure Data Source	87
7.3.6 STRS Infrastructure-Provided Device Control API	87
7.3.7 STRS Device-Provided Device Control API.....	91
7.3.8 STRS Infrastructure File Control API	94
7.3.9 STRS Infrastructure Messaging API	97
7.3.10 STRS Infrastructure Time Control API.....	100
7.3.11 STRS Predefined Data	106
7.3.12 Error Handling	112
7.4 Portable Operating System Interface	113
7.4.1 STRS Application Environment Profile	114
7.5 Network Stack.....	117
7.6 Operating System.....	117
7.7 Hardware Abstraction Layer.....	117
8. EXTERNAL COMMAND AND TELEMETRY INTERFACES	120

LIST OF APPENDICES

<u>APPENDIX</u>	<u>PAGE</u>
A POSIX® API Profile.....	124
B Reference Documents	130
C Acknowledgments	133
D Requirements Compliance Matrix	134

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
1	Roles and Responsibilities	15
2	Hardware Architecture Diagram Key	32
3	Notional STRS Hardware Architecture Implementation.....	33
4	GPM Architecture Details	38
5	SPM Architecture Details	41
6	RFM Architecture Details.....	45
7	Waveform Component Instantiation.....	51
8	Notional High-Level Software and Configurable Hardware Design Waveform Application Interfaces	56
9	STRS Software Execution Model.....	59
10	STRS Layered Structure in UML	60
11	STRS Operating Environment	62
12	POSIX®-Compliant Versus POSIX®-Conformant OS	64
13	STRS Infrastructure	65
14	STRS Application and Device Structure	67
15	Profile Building Blocks	115
16	Command and Telemetry Interfaces.....	120

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
1	STRS Module Interface Characterization.....	48
2	Example—Operating Power Interface (Platform Supplied).....	50
3	STRS Architecture Subsystem Key.....	61
4	STRS Software Component Descriptions.....	63
5	APP_Configure()	70
6	APP_Destroy().....	71
7	APP_GetHandleID().....	71
8	APP_GetHandleName().....	71
9	APP_GroundTest()	72
10	APP_Initialize()	72
11	APP_Instance()	73
12	APP_Query()	73
13	APP_Read()	74
14	APP_ReleaseObject()	74
15	APP_RunTest().....	75
16	APP_Start().....	75
17	APP_Stop()	76
18	APP_Write()	76
19	STRS_Configure()	77
20	STRS_GroundTest()	78
21	STRS_Initialize()	78
22	STRS_Query()	79
23	STRS_ReleaseObject()	79
24	STRS_RunTest().....	80
25	STRS_Start().....	80
26	STRS_Stop().....	81
27	STRS_AbortApp()	81
28	STRS_GetErrorQueue().....	82
29	STRS_GetHandleName()	82
30	STRS_HandleRequest().....	83
31	STRS_InstantiateApp().....	83
32	STRS_IsOK().....	84
33	STRS_Log().....	85
34	STRS_ValidateHandleID().....	85
35	STRS_ValidateSize().....	86

NASA-STD-4009A

LIST OF TABLES (Continued)

<u>TABLE</u>	<u>PAGE</u>
36 STRS_Write()	86
37 STRS_Read()	87
38 STRS_DeviceClose()	88
39 STRS_DeviceFlush()	88
40 STRS_DeviceLoad()	89
41 STRS_DeviceOpen()	89
42 STRS_DeviceReset()	90
43 STRS_DeviceUnload()	90
44 STRS_SetISR()	91
45 DEV_Close()	92
46 DEV_Flush()	92
47 DEV_Load()	92
48 DEV_Open()	93
49 DEV_Reset()	93
50 DEV_Unload()	93
51 STRS_FileClose()	94
52 STRS_FileGetFreeSpace()	94
53 STRS_FileGetSize()	95
54 STRS_FileGetStreamPointer()	95
55 STRS_FileOpen()	96
56 STRS_FileRemove()	96
57 STRS_FileRename()	97
58 STRS_MessageQueueCreate()	98
59 STRS_MessageQueueDelete()	98
60 STRS_PubSubCreate()	99
61 STRS_PubSubDelete()	99
62 STRS_Register()	100
63 STRS_Unregister()	100
64 Document STRS Clock/Timer	101
65 STRS_GetNanoseconds()	101
66 STRS_GetSeconds()	102
67 STRS_GetTime()	102
68 STRS_GetTimeAdjust()	103
69 STRS_GetTimeWarp()	103
70 STRS_SetTime()	104
71 STRS_SetTimeAdjust()	104
72 STRS_Sleep()	105
73 STRS_TimeSynch()	105

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

LIST OF TABLES (Continued)

<u>TABLE</u>		<u>PAGE</u>
74	STRS Predefined Data	107
75	Queryable Platform Parameter Names	112
76	Queryable Application Parameter Names.....	112
77	Replacements for Unsafe Functions	116
78	Sample HAL Documentation.....	119
79	Suggested Services Implemented by the STRS Command and Telemetry Interfaces.....	122
80	POSIX® Subset Profiles PSE51, PSE52, and PSE53	124

SPACE TELECOMMUNICATIONS RADIO SYSTEM (STRS) ARCHITECTURE STANDARD

1. SCOPE

This NASA Technical Standard describes the Space Telecommunications Radio System (STRS) architecture for software-defined radios (SDRs), an open architecture for NASA space and ground radios. STRS provides a common, consistent framework to abstract the application software from the STRS platform hardware to reduce the cost and risk of using complex reconfigurable and reprogrammable radio systems across NASA missions. It achieves this objective by defining an architecture to enable the reuse of applications (waveforms and services implemented on the SDR) across heterogeneous SDR platforms and reduce dependence on a single vendor. The NASA Technical Standard provides a detailed description and set of requirements to implement the architecture. The NASA Technical Standard focuses on the key architecture components and subsystems by describing their functionality and interfaces for both the hardware and the software, including the applications. The intended audience for this NASA Technical Standard is composed of software and hardware developers who need architecture specification details to develop an STRS platform or application.

A corresponding NASA Technical Handbook, NASA-HDBK-4009A, Space Telecommunications Radio System (STRS) Architecture Standard Rationale, provides the rationale for the decisions made to develop the architecture, provides additional information to clarify the requirements, gives further examples, and answers questions from users.

This NASA Technical Standard is only one of a set of documents to be provided by the mission and used by the STRS platform providers or STRS application developers in the development of an STRS-compliant radio and/or applications. This NASA Technical Standard defines a standard part of the architecture for software-defined radios. The complete architecture is determined by the project. Typical radio acquisition specifications, which include size, weight, power, radiation and safety requirements, connector details, performance and behavior requirements, documentation, and data rights agreements are to accompany this NASA Technical Standard in a radio procurement.

1.1 Purpose

The purpose of this NASA Technical Standard is to establish an open architecture specification for NASA space and ground SDRs. Currently, most missions either use hardware radios, which cannot be modified once deployed, or software-defined radios with an architecture that depends on the radio provider and involves significant effort to add new applications. The development of this NASA Technical Standard is part of the larger STRS program currently underway to define NASA's application of software-defined, reconfigurable technology to meet future space communications and navigation system needs. Software-based SDRs enable advanced operations that potentially reduce mission life-cycle costs for space or ground platforms.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

SDR technology allows radios to be reconfigured to perform different functions without the necessity of using multiple radios to accomplish each communication function, enabling radio count reduction to decrease mass and power resources.

The STRS project provides the infrastructure and guidance for a repository of applications developed for SDRs using this NASA Technical Standard. Adherence to this NASA Technical Standard for the development of SDR platforms and applications and submittal of the applications to the repository will enable the missions to leverage earlier efforts by reusing various software components compliant with the architecture developed in other NASA programs. This will reduce the cost and risk of deploying SDRs for future NASA missions.

The hardware, configurable hardware design, software architecture, and the supporting documentation defined by this STRS Standard provide the ability to port applications among heterogeneous platforms reducing the knowledge of a second platform, reduce the reliance on the initial STRS platform providers, and enable the implementation of waveforms and services that a project envisions for its SDRs.

1.2 Executive Summary

1.2.1 Key Architecture Requirements

The key goals in the development of the STRS architecture are to decrease the development time, cost, and risk of using SDRs while still accommodating advances in technology. The advent of software-based applications allows minimal rework to reuse applications and to adapt to evolving requirements. The architecture does not include mission-specific functional and performance requirements such as contents or format of the external interfaces to the SDR; waveform-specific requirements such as data rate, coding scheme, and modulation and demodulation techniques; specific hardware; or security, fault tolerance, redundancy, and fault mitigation approaches. Instead, the architecture is careful to enable all solutions that the mission might require as they relate to the mission-specific functional and performance specifications.

The requirements for the architecture are derived from the following STRS goals and objectives:

- Usable across most NASA mission types (scalability and flexibility).
- Decrease development time and cost.
- Increase reliability of SDRs.
- Accommodate advances in technology with minimal rework (extensibility).
- Adaptable to evolving requirements (adaptability).
- Leverage existing or developing standards, resources, and experience (state-of-the-art and state-of-practices).
- Maintain vendor independence.
- Enable waveform application portability and reusability.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

To meet these goals and objectives, the STRS architecture has an open architecture design that accommodates the range of radio form factors that are envisioned by NASA for all mission classes.

The architecture can also not preclude the implementation of mission-developed services on the SDR such as:

- Multiple waveforms operating simultaneously across any RF band defined in the SDR specification.
- Commanded built-in-test (BIT) and status reporting.
- Real-time operational diagnostics.
- Automated system recovery and initialization.
- Networking and navigation within the SDR.
- Secure transmission.
- Shared processing among on-board elements.

1.2.2 STRS Overview

This STRS Standard consists of hardware, configurable hardware design, and software architectures with accompanying description, guidance, and requirements. The hardware architecture is defined in section 4. Section 5 outlines the process and requirements associated with application development. The configurable hardware design architecture is defined in section 6. The software architecture is defined in section 7. An overview of each is provided below.

The terms “software” and “configurable hardware design” are used in this NASA Technical Standard to distinguish the architecture items that apply to code (source code, object code, executables, etc.) implemented on a processor; and designs (hardware description language (HDL) source, loadable files, data tables, etc.) implemented in a configurable hardware device such as a field programmable gate array (FPGA). Both items can change the functionality of the radio in-situ using program control. The term “software” is also used in a generic sense in this NASA Technical Standard to discuss all configurable items of the radio, including configurable hardware design. The terminology used is not meant to imply design and implementation process.

The STRS hardware architecture is specified in a modular fashion at a functional level. The hardware architecture requirements are written so that the hardware provider defines the functional breakdown (modules) of the system and publishes the functions and interfaces for each module and for the entire STRS platform in a hardware interface description (HID) document. Using this information enables NASA and others developing applications or additional modules, or interfacing to the platform, to have the knowledge to integrate and test the hardware interfaces and understand the features and limitations of the platform.

This NASA Technical Standard encourages the development of applications that are modular, portable, reconfigurable, and reusable. STRS applications use the STRS infrastructure-provided application program interfaces (APIs) and services to load, verify, execute, change parameters,

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

terminate, or unload an application. The STRS applications are submitted to the NASA STRS Application Repository to allow applications to be reused in the future according to any accepted release agreements. The NASA STRS Application Repository contains software, configurable hardware design, metadata, and documentation for STRS applications, STRS Devices, and operating environments (OEs). The appropriate application artifacts provide future missions the information to use the application with limited effort.

The configurable hardware design architecture provides guidance to the development of applications that are partially or fully implemented in a hardware device such as an FPGA. Early consideration to enable reuse during the development of configurable hardware design is critical. Suggestions are provided to decrease the reuse and porting effort, and requirements are included for the development of configurable hardware design to use the platform-specified abstraction.

The STRS software architecture is the focus of the current version of the STRS Standard. The software architectural model describes the relationship between the software elements, defined in layers, in an STRS-compliant radio. The model illustrates the different software elements used in the software execution and defines the API layers between an STRS application and the OE, and between the OE and the hardware platform.

The STRS software layers are separated to enable developers to implement the software layers differently according to their requirements while still complying with the STRS architecture. A key aspect is the abstraction of the STRS application, which is either a waveform or service, from the underlying OE software to promote portability and reusability of the STRS application. The STRS software architecture uses three primary interfaces, as follows: (1) the STRS APIs; (2) the STRS hardware abstraction layer (HAL) specification; and (3) the Portable Operating System Interface (POSIX®). The STRS APIs provide the interfaces that allow applications to be instantiated and use platform services. These APIs also enable communication between STRS applications and the STRS infrastructure. The HAL provides a software view of the specialized hardware by abstracting the physical hardware of interfaces. It is to be published so that software and configurable hardware design running on the platform's specialized hardware can integrate with the STRS infrastructure.

1.2.3 Roles and Responsibilities

The final configuration of an SDR and its applications is generally a product of multiple organizations performing various tasks. The separation of requirements, responsibilities, and resulting tasks is assigned in this NASA Technical Standard by logical role where each role has requirements that may be satisfied by an individual or delegated to a subordinate organization(s). As Figure 1, Roles and Responsibilities, illustrates, the effort begins with a mission need for a radio, which could support communications, navigation, and in some instances even networking functions. The mission system engineer defines radio system requirements. For each mission, the STRS integrators, STRS platform providers, and STRS application developers are selected. Eventually, the platform and applications are integrated into the STRS-compliant radio product. Both the hardware and software are tailored to meet mission-specific needs.

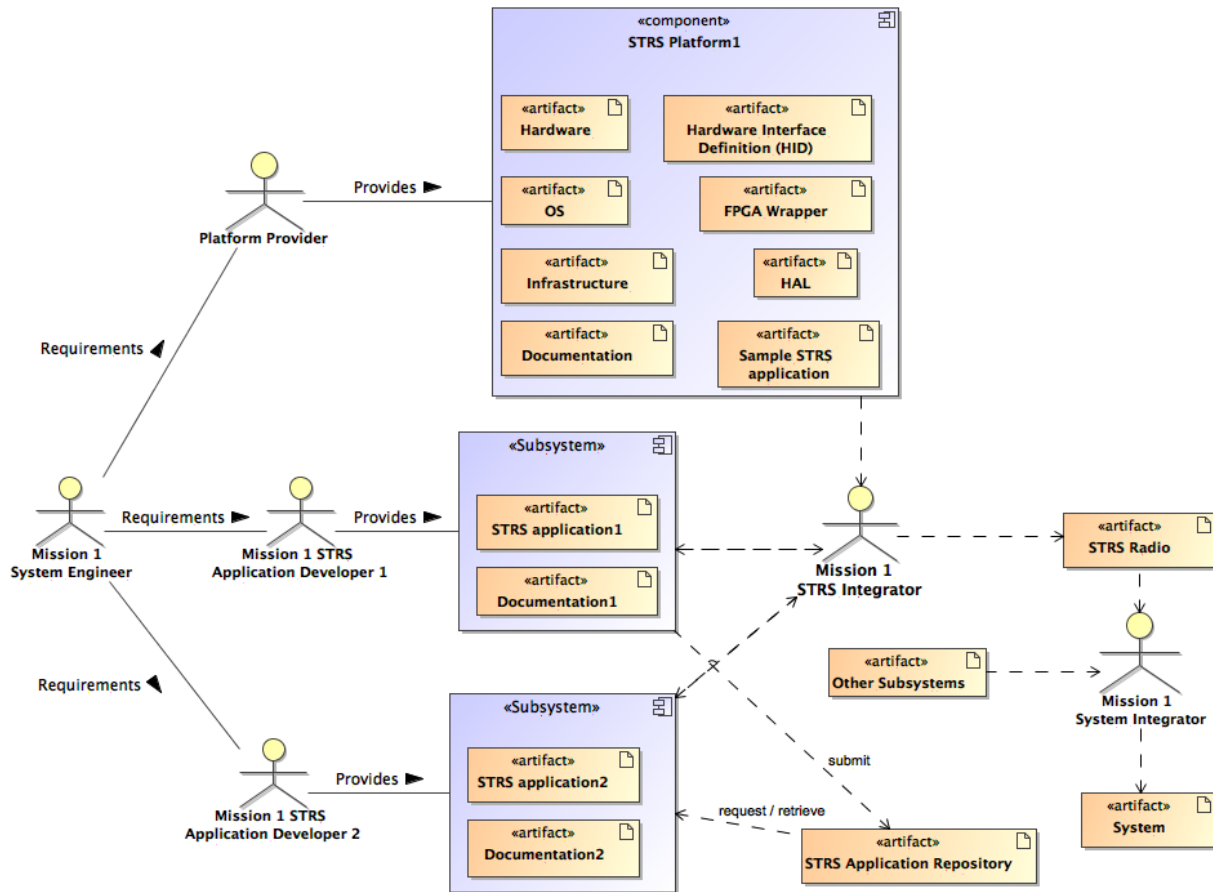


Figure 1—Roles and Responsibilities

The STRS platform provider is the organization responsible for the design and development of the SDR hardware platform, including the STRS OE (e.g., infrastructure, OS), and associated documentation. The OE and hardware platform are a unique set and become the SDR platform.

The STRS platform provider is responsible for all the documentation associated with the platform. The STRS platform provider is responsible for the FPGA platform-specific wrapper and software header files specifying the required interface, constants, typedefs, and structs. The STRS platform provider is also responsible for any STRS script formats or software configuration file formats, any extensible markup language (XML) schema, and any transformation tool for controlling instantiation, and their associated documentation, if deemed necessary. If the STRS platform provider delegates responsibility for part of the OE to a separate infrastructure provider, the responsibility for the appropriate files and documentation may be delegated to that provider as well. If the STRS platform provider delegates responsibility for part of the hardware to a separate hardware provider, the responsibility for the pertinent HID documentation may be delegated to that hardware provider as well. The STRS platform provider

NASA-STD-4009A

is ultimately responsible to integrate and deliver all aspects of the platform and OE documentation.

The mission and the STRS application developer have the responsibility to evaluate the contents of the STRS repository against the mission-developed application requirements and determine if a new application should be developed or if an appropriate application exists in the repository that is a candidate for a port to the defined platform. Depending on the results of this decision, the STRS application developer either creates a new application or ports an existing STRS application, usually retrieved from the STRS repository. The STRS application developer performs unit tests, and documents the functionality.

The STRS integrator brings the hardware platform and software application together on the SDR platform. The STRS integrator could be the STRS platform provider, the STRS application developer(s), a mission engineer, or even a third party. The STRS integrator's role is to have the application properly running on the SDR platform to meet the communication, navigation, or other functions of the mission. Once the STRS radio integration is complete, it is delivered to a system integrator who incorporates it into the mission spacecraft system. Software updates are possible during the STRS radio and system integration. Following system integration, the STRS application developer delivers the version of the software used for the deployed system, and the associated documentation, to the STRS repository.

It is likely that multiple applications will be developed for a single STRS platform prior to deployment and during its operational lifetime. During operations, after the radio has been deployed, additional application providers, who may be independent of the original platform or application provider, could develop additional applications for the original STRS radio. The new providers develop applications for the SDR platform much like the original application provider and deliver the application to the same or possibly a different STRS integrator. Following successful integration, the application software is delivered to the STRS Application Repository. Mission operations performs the role of system integrator when uploading the application to the STRS radio.

For the next mission (mission 2), either a derivative of the initial platform or a new STRS-compliant platform is envisioned. The mission 2 application provider may withdraw applications from the repository to use for the new STRS radio project. The mission 2 application follows a similar path of delivery to the mission 2 STRS integrator who incorporates the new hardware platform material and delivers the mission 2 STRS radio based on the original application and new hardware platform. As more and more missions deploy SDRs, new platforms and applications may be developed but also platforms and software are reused, marking the significant difference with the new technology compared to legacy radios.

1.2.4 Background

The deployment of SDRs for NASA missions was a new concept in 2002 due to the development of reconfigurable components useable for space radios. The need to reduce the cost and risk of using SDRs was identified and the development of the STRS architecture was initiated. In 2007,

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

the architecture was determined to be ready for flight implementation in a technology development project. This project was originally called the Communication, Navigation, and Networking reConfigurable Testbed (CoNNeCT). CoNNeCT was later renamed the SCaN Testbed. Three SDRs, compliant with the STRS architecture, were procured in 2008 and 2009 for the Space Communications and Navigation (SCaN) Testbed, using the architecture defined in a technical memorandum and referred to in the procurement specifications as Version 1.02.1. The SCaN Testbed was launched in July 2012 and operates on an external truss on the International Space Station (ISS). The SCaN Testbed is an experimental communications system that provides the capability for S-Band, Ka-Band, and L-Band communication with space and ground assets. Investigation of SDR technology and the STRS architecture are the primary focus of the SCaN Testbed. As a completely reconfigurable testbed, the SCaN Testbed provides experimenters an opportunity to develop and demonstrate experimental waveforms and applications for communication, networking, and navigation concepts and to advance the understanding of operating SDRs in space. Lessons learned from the STRS platform provider, STRS application developers, and STRS integrators of the SCaN Testbed provided critical insight for the development of the current NASA Technical Standard contained in this document. The updates from the Version 1.02.1 Technical Memorandum to the NASA-STD-4009 can be requested from the STRS project.

1.3 Applicability

This NASA Technical Standard is applicable to space and ground SDRs developed by or for NASA missions.

This NASA Technical Standard is approved for use by NASA Headquarters and NASA Centers and Facilities and may be cited in contract, program, and other Agency documents as a technical requirement. It may also apply to the Jet Propulsion Laboratory (a Federally Funded Research and Development Center (FFRDC)), other contractors, recipients of grants and cooperative agreements, and parties to other agreements only to the extent specified or referenced in applicable contracts, grants, or agreements.

Verifiable requirement statements are numbered and indicated by the word “shall”; this NASA Technical Standard contains 120 requirements. Explanatory or guidance text is indicated in italics beginning in section 4. To facilitate requirements selection and verification by NASA programs and projects, a Requirements Compliance Matrix is provided in Appendix D.

1.4 Tailoring

This NASA Technical Standard does not depend on any other NASA Procedural documents. Tailoring is not allowed. Compliance may be adapted by a project manager or contract manager in rare instances using the available project or contract procedures.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

2. APPLICABLE DOCUMENTS

2.1 General

The documents listed in this section contain provisions that constitute requirements of this NASA Technical Standard as cited in the text.

2.1.1 [NTS-1] The latest issuances of cited documents shall apply unless specific versions are designated.

2.1.2 [NTS-2] Non-use of specifically designated versions shall be approved by the responsible Technical Authority.

Applicable documents may be accessed at <https://standards.nasa.gov> or obtained directly from the Standards Developing Body or other document distributors. When not available from these sources, information for obtaining the document is provided.

2.2 Government Documents

None

2.3 Non-Government Documents

Institute of Electrical and Electronics Engineers (IEEE)

Note: The following document is the current version of the POSIX® standard as of 2003 applicable to requirement STRS-90.

IEEE 1003.13™

IEEE Standard for Information Technology—Standardized
Application Environment Profile (AEP)—POSIX®
Realtime and Embedded Application Support

See Appendix B for reference documents.

2.4 Order of Precedence

2.4.1 The requirements and standard practices established in this NASA Technical Standard do not supersede or waive existing requirements and standard practices found in other Agency documentation.

2.4.2 [NTS-3] Conflicts between this NASA Technical Standard and other requirements documents shall be resolved by the responsible Technical Authority.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

3. ACRONYMS, ABBREVIATIONS, AND DEFINITIONS

3.1 Acronyms and Abbreviations

A	ampere
ADC	analog-to-digital converter
AEP	application environment profile
AGC	automatic gain control
ANSI	American National Standards Institute
API	application program interface
APP	application
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
BIT	built-in test
BSP	board support package
C&DH	command and data handling
CCSDS	Consultative Committee for Space Data Systems
CoNNeCT	Communication, Navigation, and Networking reConfigurable Testbed (This name has been replaced with SCaN.)
COTS	commercial off the shelf
DAC	digital-to-analog converter
DEC VMS	Digital Equipment Corporation Virtual Memory System
DLL	dynamic link library
DSP	digital signal processor
EDIF	electronic design interchange format
EEPROM	electrically erasable, programmable read-only memory
FFRDC	Federally Funded Research and Development Center
FIFO	first in, first out
FIPS PUB	Federal Information Processing Standard Publication
FPGA	field programmable gate array
GPIO	general purpose input output
GPM	general purpose processing module
GPP	general purpose processor
GPS	global positioning system
HAL	hardware abstraction layer
HDBK	Handbook
HDL	hardware description language
HID	hardware interface description
HW	hardware
I/O	input/output

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

ID	identification, identifier
IEC	International Electrotechnical Commission
IEEE	The Institute of Electrical and Electronics Engineers
IF	intermediate frequency
INCITS	InterNational Committee for Information Technology Standards
IP	internet property
ISO	International Organization for Standardization
ISS	International Space Station
JTC	Joint Technical Committee
LLC	logical link control
LNA	low-noise amplifier
LRU	logical replaceable unit
MAC	media access control, a sublayer of the open system interconnection data link layer
MDA	model-driven architecture
MIL	military
MMU	memory management unit
mVpp	millivolt peak-to-peak voltage
NASA	National Aeronautics and Space Administration
NM	network module
NPR	NASA Procedural Requirement
NTS	NASA Technical Standard
OAL	OEM adaptation layer
OE	operating environment
OEM	original equipment manufacturer
OM	optical module
OMG	Object Management Group
ORMSC	Operational Research MSc Programmes
OS	operating system
OSS	open source software
OTAP	over-the-air programming
PIM	platform-independent model
POSIX®	Portable Operating System Interface
PROM	programmable read-only memory
PSE51	minimal realtime system profile, defined in IEEE 1003.13
PSE52	realtime controller system profile, defined in IEEE 1003.13
PSE53	dedicated realtime controller system profile, defined in IEEE 1003.13
PSE54	multi-purpose realtime system profile, defined in IEEE

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

1003.13

PUB	publish/publication
RAM	random access memory
RF	radio frequency
RFM	radio frequency module
ROM	read-only memory
RT	reconfigurable transceiver
RTOS	real-time operating system
SCA	Software Communications Architecture
SCaN	Space Communications and Navigation (new name for CoNNeCT)
SDR	software-defined radio
SEC	security module
SEU	single-event upset
SPM	signal-processing module
SRAM	static random access memory
STD	standard
STRS	Space Telecommunications Radio System
SUB	subscribe
SWRADIO	software radio
TT&C	telemetry, tracking, and command
TCP	transmission control protocol
TM	technical memorandum
TMR	triple-mode redundancy
TP	technical publication
UML	Unified Modeling Language
UNIX®	computer operating system developed by AT&T Bell Laboratories
USA	United States of America
V	volt
V&V	verification and validation
VDD	version description document
VHDL	VHSIC hardware description language
VHSIC	very-high-speed integrated circuit
VMS	Virtual Memory System
V _{pp}	peak-to-peak voltage
Windows NT®	Windows operating system—NT, new technology
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML 1.0 Schema Definition

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation

3.2 Definitions

To improve the understanding of material presented in the STRS documents, new terms and definitions that are rapidly emerging in the field of SDRs are provided below, as follows:

Adaptability: Ease with which a system satisfies differing system constraints and user needs.

Application: Executable software program that exhibits predetermined functionality and may contain one or more software modules.

Note: A primary example of an STRS application is the waveform application. An STRS application is to comply with the architecture.

Application Program Interface (API): Formalized set of software calls and routines that can be referenced by the application program in order to access supporting system or network services.

Architecture: Organizational structure of a system, the relationships between its components, and the principles and guidelines governing their design and evolution over time.

Autonomous Operation: Implementation decision-making algorithm that can be implemented on a system level (fully autonomous) or at the subsystem level (semi-autonomous) according to mission requirements.

Availability: Degree to which a system or component is operational and accessible when required for use.

Board Support Package (BSP): Hardware abstraction of the general purpose processing module (GPM) for the POSIX®-compliant operating system (OS), which contains the boot, generic and processor-specific drivers required for the specific hardware.

Note: The BSP leverages commercial-off-the-shelf (COTS) device drivers and other software necessary for applications to access the specific hardware.

Built-In Test: Internal test to determine whether or not the STRS radio and each subsystem are working properly.

Note: STRS health management uses BIT to automatically monitor the health of the system and to pass any identified problem to the fault management. STRS fault management uses BIT to automatically monitor, diagnose, and isolate system problems.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Component: Hardware or software that makes up a system, which may be subdivided into other parts or units.

Note: The terms “module,” “component,” and “unit” are often used interchangeably or defined to be subelements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

Configurable Hardware Design: The electronic files used to configure the portion of the SDR hardware that can be updated after deployment.

Note: Configurable hardware design is often informally—and often incorrectly—referred to as firmware. Notes: (1) This term is sometimes used to refer only to the hardware device or only to the computer instructions or data, but these meanings are deprecated. (2) The confusion surrounding this term has led some to suggest that it be avoided altogether.”

For this NASA Technical Standard, to avoid confusion the term “firmware” is not being used. The term “configurable hardware design” was selected instead. For a configurable hardware device such as an FPGA, it includes the FPGA source code written in HDL, the image stored in random access memory (RAM) and used by the FPGA, and supporting hardware configuration files, if applicable.

Data Publisher: Software component that transmits data to one or more subscribers.

Note: In the STRS architecture, it may be implemented by waveforms and parts of the STRS infrastructure.

Data Subscriber: Software component that receives data from the data publisher.

Note: In the STRS architecture, it may be implemented by waveforms and parts of the STRS infrastructure.

Deployment: All the processes involved in getting new software or hardware up and running properly in its environment, including installation, configuration, running, testing, and making necessary changes.

Evolvability: Ease with which a system or component can be modified to take advantage of new software or hardware technologies.

External Interface: Functional and physical connections at the boundaries of a system that are designed to interoperate with other systems or components.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Note: Examples include interfaces to or from the flight computer, power, data sources, data sinks, antennas, mounting locations, and optical links.

Fault Management: Set of functions that detect, isolate, and correct malfunctions within the system or provide notifications.

Flexibility: Ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Flight Computer: Separate computer that is used to monitor and control the STRS radio.

Note: The flight computer may be connected to the STRS radio electrically, electromagnetically, optically, etc. The flight computer may contain the watchdog timer for the STRS radio.

General-Purpose Processing Module: Hardware module that contains and executes the STRS OE and STRS applications and services software.

Note: The GPM usually consists of the general purpose processor (GPP), appropriate memory (both volatile and nonvolatile), system bus, the spacecraft (or host) telemetry, tracking, and command (TT&C) interface, ground support telemetry and test interface, and the components to support the radio configuration. The GPP may be remote or include a system-on-a-chip, if necessary.

Guidelines: Nonbinding statements intended to direct the broader and longer-term aspects of the STRS architecture.

Handle ID: An identifier used to control access to applications, devices, files, messaging queues, and other similar resources.

Hardware Abstraction Layer: Library of functions that provides a software view of the specialized hardware by abstracting the physical hardware interfaces.

Hardware Device: Physical entity that is capable of performing a function.

Hardware Interface Description: Documentation containing information about each module's physical and electrical connections, performance, capability, size, weight and power, as applicable, to enable integration between components of the system.

Health Management: Monitoring the health and performance of a system, subsystem, device, or process.

Note: Health management invokes fault management, when corrective action is needed.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Hierarchical Structure: Structure that characterizes a system in which components are contained by other components and/or provide services to the next higher-level components.

Note: Hierarchical structure is a key attribute of an open architecture that enables system description, design, development, installation, operation, upgrades, and maintenance to be performed at a given layer or layers. This type of structure allows each layer to be modified without affecting the other layers.

Interoperability: (1) Ability of a system to work with or use the parts or equipment of another system; (2) capability of different radio systems or radio networks to communicate and exchange information with each other.

Note: Dissimilar systems or networks may achieve interoperability by changing their operating parameters to a common compatible format or by operating through a bridge that translates between incompatible formats. An alternate definition is to determine and adapt all radio parameters required for broadest communication compatibility across all target networks.

Legacy Radio: Nonprogrammable radio designed for one fixed configuration that produces a single waveform at a specified frequency.

Note: The radio may have limited options for tuning, data rate, and so forth or may even carry multiple types of data, but it is incapable of adapting to new waveforms.

Maintainability: Ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.

Method: Implementation of an operation, which specifies the algorithm or procedure associated with an operation.

Module: Self-contained hardware or software component that interacts with a larger system.

Note: A software module (program module) performs specific tasks within a software system. A hardware module is a physical grouping of devices capable of implementing specific functions.

Open Architecture: Architecture whose functions, interfaces, components, and/or design rules are defined and published.

Open Source or Open-Source Software (OSS): Any computer software distributed under a license that allows users to change and share the software freely.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Note: OSS has its source code freely available, and end-users have the right to modify and redistribute the software to others.

Open System: System that has specified, publicly maintained, and readily available standards.

Over-the-Air Programming (OTAP): Method of providing software updates by means of a communication channel realized by the STRS radio itself.

Portability: Ease with which a system application or service can be transferred from one hardware or software environment to another.

Portable Operating System Interface: Family of The Institute of Electrical and Electronics Engineers (IEEE) standards 1003.n that describes the fundamental operating system (OS) services and functions necessary to provide a UNIX®-like kernel interface to applications.

Note: POSIX® is not an OS but ensures that programming interfaces are available to the application programmer.

Queue: List in which items are appended to the last position in the list and retrieved from the first position in the list; that is, the next item to be retrieved is the item that has been in the list for the longest time.

Radio Frequency Module (RFM): Module that converts to and from carrier frequencies and provides the signal-processing module with baseband or intermediate frequency (IF) signals and provides the transmission and reception equipment with radio frequency (RF) signals.

Note: RFM-associated components may include filters, RF switches, duplexers, low noise amplifiers (LNAs), power amplifiers, analog-to-digital converters (ADC), and digital-to-analog converters (DAC). This module handles the interfaces that control the final stage of the transmission or first stage of the reception of the wireless signals, including antennas.

Real-Time Operating System (RTOS): OS that guarantees a certain capability within a specified time constraint.

Reconfigurability: Ability to modify functionality of a radio by changing the operational parameters without requiring a software update.

Reconfigurable Radio: Radio whose functionality can be changed either through manual reconfiguration of radio modules or under software control.

Note: Software reconfiguration control of such radios may involve any element of the radio-communication network. SDRs are a subset of reconfigurable radios.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Reconfigurable Transceiver (RT): Radio with limited processing and selectable remote reconfiguration (e.g., filter parameters and modulations).

Reconfiguration: Act of modifying the functionality of a radio by changing the operational parameters without updating the software.

Reentrant Function: Function that can be entered before completion of a prior execution of that same function and execute correctly.

Note: A function that is reentrant is automatically thread-safe, but not necessarily the reverse.

Reliability: Ability of a system or component to perform a required function under stated conditions for a specified period of time.

Reprogrammability: Ability to modify functionality of a radio by changing the operational software or configurable hardware design either wholly or partially.

Reusability: Degree to which a software module or other work product can be used in more than one computing program or software system.

Scalability: Degree to which components or functions in an implementation can be sized in systematic proportions for varying capacities.

Selectable: Ability to choose from a range of choices.

Note: For example, a selectable parameter may be modified to change system characteristics at runtime.

Services: Software programs that provide functionality available for use by other applications.

Signal-Processing Module (SPM): Module that contains the implementations of the signal processing used to handle the transformation of received digitally formatted signals into data packets and/or the conversion of data packets into digitally formatted signals to be transmitted.

Note: The SPM may include the spacecraft data interface, application specific integrated circuits (ASICs), FPGAs, digital signal processors (DSPs), memory, and connection fabric or bus.

Slice: A physical hardware form factor like a circuit board.

Software: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Note: In certain contexts in this NASA Technical Standard, the term “software” also encompasses configurable hardware design. For example, in the term “software defined radio,” the word “software” includes configurable hardware design. In other contexts, the word “software” is meant to imply code running on a processor, especially in the Software Architecture section. In this case, even if the processor is embedded within configurable hardware, the software that executes on the processor is not “configurable hardware design.”

Software-Defined Radio: Radio in which some or all of the physical layer functions are implemented in software and/or configurable hardware design.

Software-Defined Radio Architecture: Comprehensive, consistent set of functions, components, and design rules according to which radio communications systems may be organized, designed, constructed, deployed, operated, and evolved over time.

Note: A useful architecture partitions functions and components such that (1) functions are assigned to components clearly, and (2) physical interfaces among components correspond to logical interfaces among functions.

Software Device: A software abstraction of a hardware device or group (aggregate) of hardware devices.

Note: An STRS device is a software device that is part of the STRS infrastructure, having a well-defined and portable API that may use the HAL to read, write, and control hardware devices.

Software Radio: Extension of an SDR with more functionality implemented in GPPs as opposed to ASICs and FPGAs. A software radio implements communications functions primarily through software in conjunction with minimal hardware.

Note: Software radios are the ideal SDR in which digitization occurs at the antenna.

Space Telecommunications Radio System: Project that defines and maintains the SDR architecture for NASA.

Specialized Hardware: Separate hardware that can be initialized or controlled using software.

Standards: Technical specifications that are widely used, consensus-based, published, and maintained by recognized industry standards organizations.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

STRS Command: Source that abstracts the command functionality usually found in the interface to the flight computer.

STRS Device: An extension of an STRS Service to provide additional functionality beyond what is allowed for an STRS application such as communicating with specialized hardware.

STRS Infrastructure: Part of the STRS OE that configures and controls STRS applications and services as well as specialized hardware via the HAL.

Note: Additional functionality may be required for radio robustness and mission-dependent requirements.

STRS Operating Environment: Portion of the STRS radio that contains the STRS Infrastructure, the POSIX®-conformant RTOS, the HAL, and optional middleware software.

Note: The STRS OE executes STRS services and waveform applications.

STRS Platform: Combination of hardware and software components, including the STRS OE, capable of executing software applications.

STRS Radio: SDR that is compliant with this NASA Technical Standard and that runs one or more waveforms.

STRS Service: An STRS application encapsulating some functionality that may be used by other applications in a portable way.

System: Collection of components organized to accomplish a specific function or a set of functions.

System Architecture: Abstract description of the entities of a system, and the relationship between the entities.

Thread-Safe: Function that works correctly during simultaneous execution by multiple threads, without unwanted interaction between the threads. A thread is a part of a program that can execute independently of other parts. A thread is the smallest sequence of programmed instructions that can be managed independently by an OS scheduler.

Note: A function that is reentrant is automatically thread-safe, but the reverse is not necessarily true.

Upgradeability: Ability to make changes to a portion of the system easier by limiting the changes, as much as possible, to the updated part.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Note: It is clear that greater upgradeability is greater ability.

Usability: Ease with which a user can learn to operate, prepare input for, and interpret the output of a system or component.

Use Cases: Situations that capture the requirements of a system by describing how the system should interact with the users or other systems (the actors) to achieve specific goals.

Watchdog Timer: Software and/or hardware that monitor the health of a system and, if a problem is detected within a certain time period, take the appropriate action to restore the system back to health.

Waveform: Set of transformations applied to information (e.g., voice or data) that is transmitted over the air and the corresponding set of transformations to convert received signals back to their information contents.

Note: Traditionally, a waveform was simply an electromagnetic signal whose amplitude varies with time.

Waveform Application: Code that implements all the functions and algorithms necessary to realize a waveform.

Note: The waveform application can be distributed among various processing elements, including specialized hardware (e.g., FPGAs and DSPs). In STRS, if the waveform application requires run-time support for functions that it cannot provide directly, it is to use the STRS APIs in the infrastructure to access the desired functions whether or not they are provided by the infrastructure directly or by other waveforms or services.

4. HARDWARE ARCHITECTURE

In addition to providing benefits by defining a standard software infrastructure for NASA's radios, this NASA Technical Standard also defines standards for the hardware portion of the radio. Hardware technologies usually change more rapidly than software, and each radio implementation generally has very specific spacecraft dependencies and requirements. Therefore, the STRS hardware architecture is specified at a functional level rather than at the physical implementation level. Also, a functional-level architecture will remain applicable over a longer time frame. It should be noted that programs have the latitude to standardize hardware requirements at the implementation level for multiple radio procurements.

The STRS hardware architecture was developed with consideration of several key constraints and conditions for operating space SDRs. One major issue driving the hardware architecture formulation was the need for flexibility, so that a single architecture is capable of addressing the range of different mission classes. The mission classes have radio requirements that range from requiring small radios that are highly optimized to meet severe size, weight, and power constraints, to missions requiring complex radios with multiple operating frequencies and higher

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

data rates. This implies that the hardware architecture accommodates a range of reconfigurable processing technologies, including GPPs, DSP, and FPGAs, as well as ASICs with selectable parameters. Currently, reconfigurable signal processing is primarily performed in specialized signal-processing hardware for the frequencies and data rates used in NASA space missions, and this is expected to continue for some time. In addition to providing capability, specialized signal processing is generally more power efficient than general purpose processing. In almost all cases, a DSP-based implementation is a lower power option than FPGA-based. The needs for specialized processing are supplemented by the software infrastructure, which is more suited for execution in a GPP. The architecture also enables technology infusion over time, accommodating the rapidly evolving capabilities of processor speeds and signal processing. In addition, the conversion point, where the signal is digitized, is moving closer to the antenna. Considering these points, the architecture provides a flexible framework, emphasizing common terminology for hardware functions and interfaces, common documentation, and common formats and requirements for waveform and service STRS application developers to utilize a platform's capabilities. The architecture does not prescribe a specific hardware implementation approach.

An STRS platform is to be delivered with a complete HID, which is described in section 5.3. The HID specifies the electrical interfaces, connector requirements, and physical requirements for the delivered radio. Each module's HID abstracts and defines the module functionality and performance.

(STRS-1) An STRS platform shall have a known state after completion of the power-up process.

4.1 Generalized Hardware Architecture and Specification

Figure 2, Hardware Architecture Diagram Key, illustrates the symbols and terminology used within the hardware architecture diagrams. The hardware diagram illustrates the radio functions and the interconnects for each module. The modules are a logical and functional division of common radio functions that comprise an STRS platform. Modules are not intended to represent physical entities of the platform. As developers choose how to distribute and implement the radio functions among hardware elements, the specification provides the guidance on the interfaces and abstractions that are to be provided to comply with the architecture. The module and function connections provided in the diagrams are data path, control, signal clock, and external interfaces.

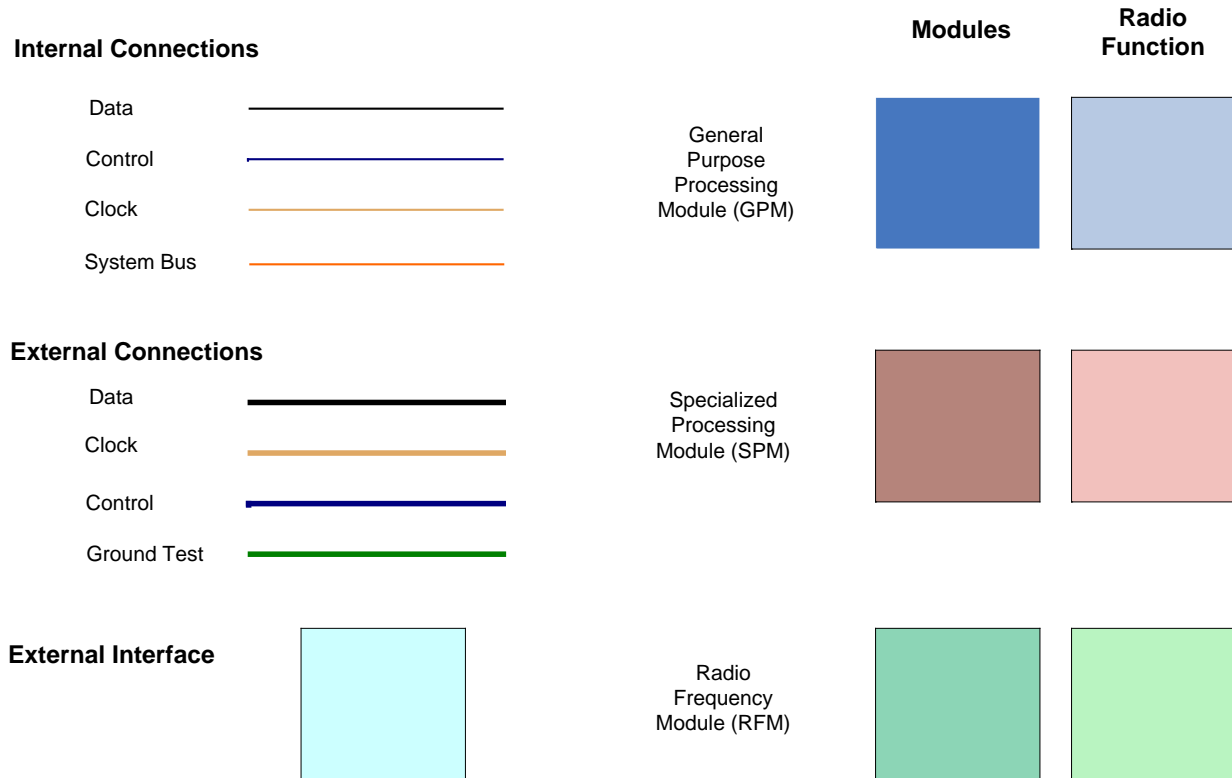


Figure 2—Hardware Architecture Diagram Key

Figure 3, Notional STRS Hardware Architecture Implementation, shows the high-level STRS hardware architecture. The figure illustrates the functional attributes and interfaces for each module. A module is a combination of logical and functional representations of platform and applications implemented in a radio. The modules are divided into their typical functions to provide a common description and terminology reference. Each STRS platform provider has the flexibility to combine these modules and their functionality as necessary during the radio design process to meet the specific mission requirements. Additional modules can be added for increased capability.

The hardware architecture does not specify a physical implementation internally on each module, nor does it mandate the standards or ratings of the hardware used to construct the radios. Thus, the radio supplier can encapsulate company proprietary circuit or software designs, provided the modules meet the specific architecture rules and expose the interfaces defined for each module. There is flexibility to physically combine or split these modules as necessary during the radio design process to meet the specific mission requirements or to optimize the design. For example, all RF and signal-processing components or functions may be integrated onto a single printed circuit board, easing footprint, interface, and integration issues, or an approach with multiple boards and enclosures could be used. Similarly, an FPGA could potentially contain both the SPM functions and the GPP, or the SPM functions could be split between an FPGA and the GPM.

NASA-STD-4009A

Each mission, or class of missions, may choose to standardize certain interfaces and physical packaging. This approach provides NASA with the flexibility to adopt different implementation standards for various mission classes. Thus, if a series of radios are required with common operating requirements, physical construction details, such as bus chassis or card slice, can be part of the acquisition strategy for cost-effective modularity at a lower level to match the life cycle of the hardware. Another example of the flexibility is where a large mission class program may choose to standardize the details of the RF-to-signal-processing interface. This might be done to facilitate the use of different RF modules, but the same signal processing module, for radios used for several similar missions.

Figure 3 depicts radio functions, or elements, expected for each module in a notional sense. It should be noted that not all the elements shown in each module are necessarily required for implementation. This architecture specifies the functionality of each module, but it does not necessarily specify how they are implemented. Mission requirements will dictate the implementation approach to each module, and the modules required in each radio.

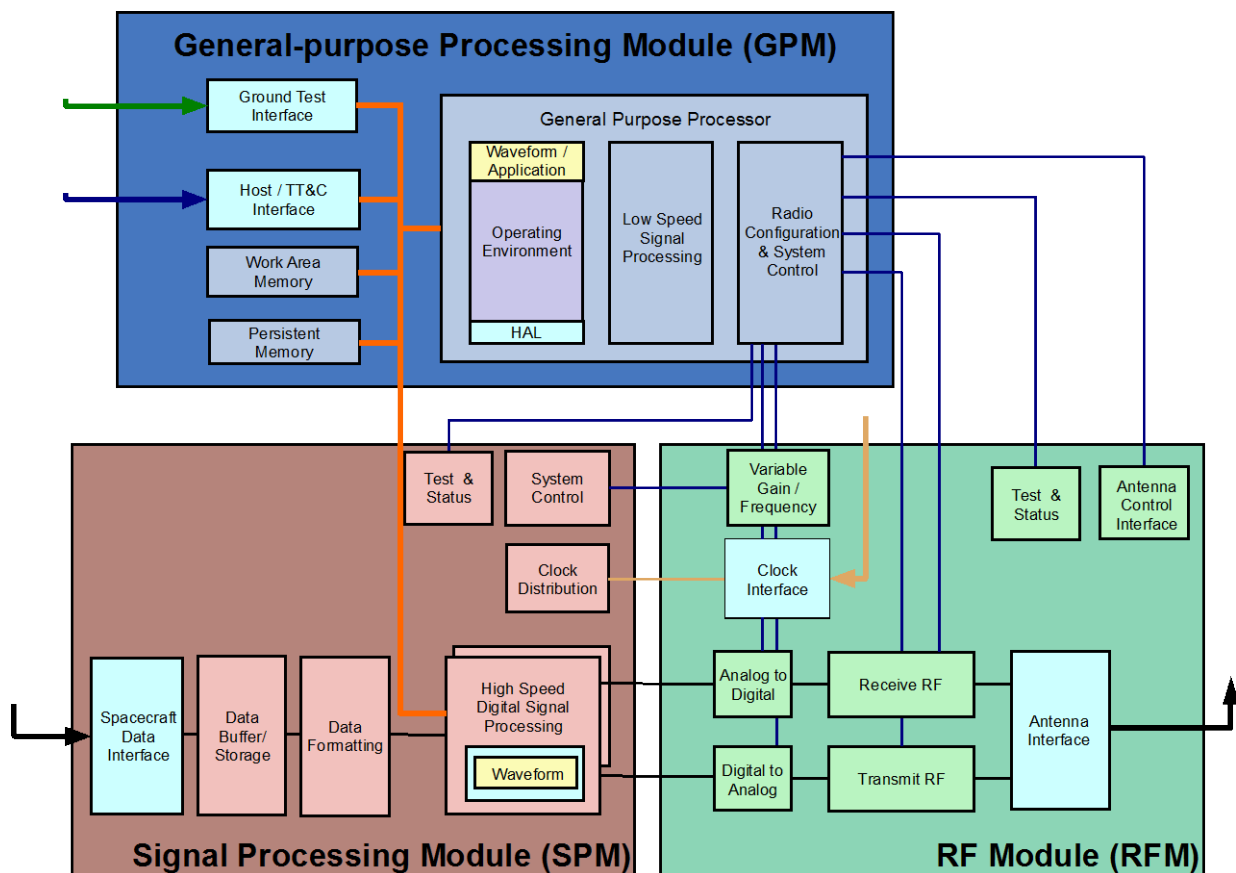


Figure 3—Notional STRS Hardware Architecture Implementation

4.1.1 Components

The approach taken in the STRS is to describe the radio hardware architecture in a modular fashion. The generic hardware architecture diagram identifies three main functional components or modules of the STRS radio. Although not shown in Figure 3, additional modules (e.g., optical, networking, and security) can be added for increased capability and will be included in the specification as it matures. The hardware architecture currently consists of the following modules:

General-Purpose Processing Module: Consists of the GPP, appropriate memory (both volatile and nonvolatile), system bus, the spacecraft (or host) TT&C interface, ground test interface, and the components to support the radio configuration.

Signal-Processing Module: This module contains the implementations of the signal processing used to handle the transformation of received digitally formatted signals into data packets and/or the conversion of data packets into digitally formatted signals to be transmitted. Also included is the spacecraft data interface. Components include ASICs, FPGAs, DSPs, memory, and connection fabric or bus.

Radio Frequency Module: This module handles the RF functionality to provide the SPM with the filtered, amplified, and digitally formatted signal. In the case of transmission, the module formats, filters, and amplifies the output signal. Its associated components include filters, RF switches, diplexer, LNAs, power amplifiers, ADCs, and DACs. This module handles the interfaces that control the final stage of transmission or the first stage of reception of the wireless signals, including antennas.

Security Module (SEC): Though not directly identified in the generic hardware diagram, an SEC is also being proposed to allow STRS radios to support future security requirements. The details of this module will be defined in later revisions of the architecture.

Network Module (NM): The architecture supports Consultative Committee for Space Data Systems (CCSDS) and Internet Property (IPs) and networking functions. However, the Network Module (NM) may be realized as a combination of both the GPM and SPM.

Optical Module (OM): This module supports the integration of optical equipment when used. The detail of this module will be defined in later revisions of the architecture. (It has many similarities to RFM, but these are for optical carriers.)

4.1.2 Functions

Test and status, fault monitoring and recovery, and radio and TT&C data-handling functions are to be implemented on all modules to some level. The details are mission specific and are stated as part of the radio acquisition. The related control and interface requirements for the shared module functions are stated in the corresponding module section.

NASA-STD-4009A

Test and Status: Each module (or combination of modules) should provide a means to query the current health of the module and run diagnostics.

Fault Monitoring and Recovery: Each module (or combination of modules) should incorporate detection of operational errors, upsets, and major component failures. These may be caused by the radiation environment (e.g., single-event upsets (SEUs)), temperature fluctuations, or power supply anomalies. In addition to detection, mitigation and fail-safe techniques should be employed. Each module should have a default power-up mode to provide the minimal functionality required by the mission. This fail-safe mode should have minimal software and/or configurable hardware design dependency.

Radio Data Path: SDRs can be implemented with or without the GPM in the data path. The STRS architecture supports the separation of the RFM and SPM data paths from the GPM. Giving the GPM access to the data path as an optional capability rather than a required capability allows for a more efficient implementation for medium and small mission classes and improves the overall performance for near-term implementations. If space-qualified GPM components mature with the performance capabilities required for signal processing, the GPM can exist within the data path and take on more signal-processing functionality, increasing flexibility.

STRS Radio Startup Process: The startup of the STRS infrastructure is expected to be initiated by the STRS platform boot process, so that it can receive and send external commands and instantiate applications. The startup process might include built-in tests for self-diagnostics to verify nominal system functionality. In order to control an STRS platform at power-up and to recover from error conditions, an STRS platform is to have a known power-up condition that sets the state of all modules. To support upgrades to the OE, an STRS platform requires the ability to alter the state (boot parameters) and/or select a boot image. The exact mechanisms and procedures used will be platform and mission specific but need to be sufficient to support upgrades to OE components, such as the OS, BSP, and STRS infrastructure.

4.1.3 Interfaces

4.1.3.1 External Interfaces

There are several key external interfaces in this architecture:

- *Host TT&C.*
- *Ground Test.*
- *Data.*
- *Clock.*
- *Antenna.*
- *Operating power.*
- *Mission defined.*

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

The host TT&C interface represents the typically low-latency, low-rate interface for the spacecraft (or other host) to communicate with the radio. The host telemetry typically carries all information sourced within the radio. This type of information traditionally is called the telemetry data and includes health, status, and performance parameters of the radio as well as the link in use. In addition, this telemetry often includes radiometric tracking and navigation data. The command portion of this interface contains the information that has the radio itself as the destination of the information. Configuration parameters, configuration data files, new software data files, and operational commands are the typical types of information found on this interface.

The Ground Test Interface provides a “development-level” view of the radio and is exclusively used for ground-based integration and testing functions. It typically provides low-level access to internal parameters not typically available to the Spacecraft TT&C Interface. It can also provide access when the GPM is not functioning (i.e., during boot).

The Data Interface is the primary interface for data that are sourced from the other end of the link and for data that are sunk to the other end of the link. This interface is separate from the TT&C interface because it typically has a different set of transfer parameters (protocol, speeds, volumes, etc.) than the TT&C information. A common interface point in the spacecraft for this type of interface is the spacecraft solid-state recorder rather than the spacecraft command and data-handling (C&DH) subsystem. This interface is also characterized by medium to high latency and high data rates.

The Clock Interface is used to input to the radio the frequency reference sufficient for supporting navigation and tracking. This type of input frequency reference is essential to the operation of the radio and provides references to the SPM and RFM. There does not have to be an external clock interface if the SPM or RFM contains an oscillator that performs this function

The Antenna Interface is used to connect the electromagnetic signal (input or output) to the radiating element or elements of the spacecraft. It also includes the necessary capability for switching among the elements if required by the mission. Steering the elements, if a function of the overall telecommunications system, is possible through this interface, but it is not typically employed because of overall operational constraints.

The Power Interface, which is not included on the diagram, is described as part of this specification at the highest levels. The Power Interface defines the types and conditions of the input energy to power the radio.

The Mission-defined Interface, which is not included in the diagram, could monitor conditions that the radio encounters such as external temperature, solar radiation, magnetic field strength, attitude, etc. The mission would assign what to do with these values. A thermal interface that monitors temperature could be used to activate a heating element or adjust dynamic factors dependent on temperature in a known way.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

4.1.3.2 Networking

A networking interface does not necessarily map directly to the SPM, GPM, or RFM. The networking interface might handle only spacecraft TT&C data or both spacecraft TT&C data and radio data. This architecture allows for those capabilities.

4.1.3.3 Internal Interfaces

To support the overall goals of the architecture, the internal interfaces (GPM system bus, GPM RFM control, SPM-to-GPM test, frequency reference, and data path) should be well documented and available without restriction.

The GPM system bus (orange lines in Figure 3) provides the primary interconnect between elements of the GPM. The GPM system bus may provide an interface between the microprocessor, the memory elements, and the external interfaces (TT&C and Test) of the GPM. The GPM system bus is the primary interface between the GPM and the SPM, as shown in the interconnection with the major SPM processing elements. Finally, the GPM system bus provides the interface by which the reprogrammable and reconfigurable elements of the SDR are modified. It supports both the read and write access to the SPM elements, as well as the reloading of hardware configuration files to the FPGAs.

The interface between the GPM and the RFM is primarily a control/status interface. Various RFM elements are controlled by the set of GPM RFM control lines (blue lines in Figure 3). Coming from the System Control block in the GPM, this control bus can be either fixed by the System Control function or programmed by the GPM software and validated and routed by the System Control function. It is important to have a hardware-based confirmation and limit-check on the software controlling any RFM elements. The System Control module of the GPM provides this functionality, thus keeping the GPM RFM Control bus within operational limits.

The Ground Test Interface (green line in Figure 3) provides specific control and status signals from different modules or functions to the Ground Test Interface block. This interface is used during development and testing to validate the operation of the various radio functions. This interface is very specific to the implementation and realization of the different modules.

The Frequency Reference Interface provides an important interface between the RFM and the SPM functions. It ties the two modules together in a way that allows for the SDR to implement tracking and navigation functions. The characteristics of this interface are defined by the various amounts of tracking accuracy required by the mission for the SPM to accomplish. This interface can be as simple as a single, common frequency reference that is conditioned from an outside source and distributed in the least degrading fashion possible to the SPM.

Finally, the data paths are the various streams of bits, symbols, and RF waves connecting the major blocks of the primary data path. For any particular implementation, the data path or bitstreams are defined by the particular application implemented in the functional blocks. The

interface between the RFM and SPM should be well defined and have characteristics suitable for that level of conversion between the analog and digital domains.

The hardware architecture can be further specified in a manner that is important for implementers to consider and follow, if the implementation dictates the necessity of particular components. Details of the GPM, SPM, and RFM are provided in subsequent sections.

4.2 Module Type Specification

4.2.1 General Purpose Processing Module

Figure 4, GPM Architecture Details, provides a closeup of the GPM detail. The GPM consists of one or more general purpose or digital signal-processing elements and support hardware components, embedded OS, software applications and interfaces to support the configuration, control, and status of the radio. The number of processing elements and the extent of support hardware will vary depending on the mission-class processing and data-handling requirements from a single system on a chip implementation for smaller mission classes to multiple logical replaceable units (LRUs) for the largest mission classes. In addition, fault tolerance requirements can also increase the number of hardware processing elements, support hardware components, and interface points required to meet the range of mission classes. The majority of processing functions of the GPM will be under software control and supported by an OS. Mission-specific data handling speeds may require the use of separate specialized support hardware (FPGA or ASIC chips) to alleviate the burden on the processing elements. Such specialized support hardware could include encryption, packet routing, and network processing-type functions.

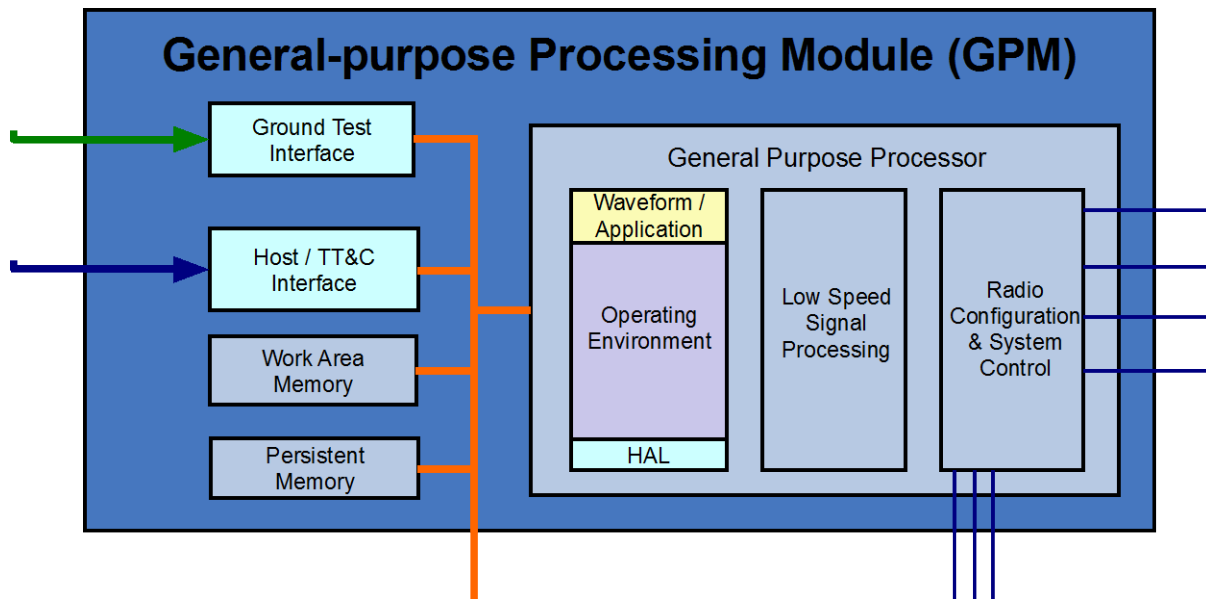


Figure 4—GPM Architecture Details

4.2.1.1 GPM Components

The GPM contains, as necessary, a GPP and various memory elements as shown in Figure 4. Depending on the particular mission class, not all memory elements are required. The GPP will typically be implemented as a microprocessor, but it could take many forms, depending on the mission class. Because the GPM is the primary control component of the radio, it is a required module for an STRS radio. A description of each element follows.

The GPP functions include the OE, the Hardware Abstraction Layer (HAL), and potentially application functions. The OE contains the STRS infrastructure, which provides the functionality for the interfaces defined by the STRS APIs specification. The OE also contains the OS and the POSIX® abstraction layer. The HAL is the library of software functions in the STRS OE that provides a platform-vendor-specific view of the specialized hardware by abstracting the underlying physical hardware interfaces. The HAL allows specialized hardware to be integrated with the GPM so that the STRS OE can access functions implemented on the specialized hardware of the STRS platform.

The Persistent Memory Storage element holds both the permanent (e.g., programmable read-only memory (PROM)) and reprogrammable code for the GPP element. In today's technology, this code is implemented using a reprogrammable technology, such as electrically erasable, programmable read-only memory (EEPROM). It is also possible, but not typically qualifiable, to implement this code storage in Flash memory.

The Persistent Memory also provides the reprogrammable storage for the SPM FPGA elements (i.e., configurable hardware design). The GPM may be responsible for programming and scrubbing the SPM FPGAs and, if so, would have access to the appropriate "code" for the FPGAs. This memory block is typically implemented using a nonvolatile memory technology, such as EEPROM, but could, in particular implementations, be implemented with PROM technology.

The Work Area Memory element is provided as operational, scratch memory for the GPP element. This memory element is implemented in concert with the GPP element and may contain both data and code, as appropriate for the execution of the radio application running in the GPM.

Finally, the GPM contains a System Control element to control and moderate the GPM system bus. This element provides the necessary control for the System Bus, including the various memory and SPM elements interfaced by the System Bus. In addition, the System Control element provides a validated interface to the RFM hardware via the GPM RFM Control Interface. As the software running on the GPP element commands the RFM elements into certain states, those commands are interpreted by the System Control element and validated in a manner that will prevent damaging configurations of the RFM; for example, tying the transmit amplifier directly to the receive amplifier, bypassing the diplexer element. This level of validation in the GPM-to-RFM interfaces would prevent damage to the radio from a software bug. The System

NASA-STD-4009A

Control element is typically implemented by a non-reprogrammable (in-flight) FPGA allowing for flexibility between instantiations of a particular implementation.

4.2.1.2 GPM Functions

The GPM will provide the overall configuration and control of the STRS architecture and may include any or all of the following functions:

- a. Management and Control.*
 - (1) Module discovery.*
 - (2) Configuration control.*
 - (3) Command, control, and status.*
 - (4) Fault recovery.*
 - (5) Encryption.*
- b. STRS infrastructure, radio configuration and control.*
 - (1) Radio control.*
 - (2) System management.*
 - (3) Application upload management.*
 - (4) Device control.*
 - (5) Message center.*
- c. External network interface processing.*
- d. Internal data routing.*
- e. Waveform data link layer.*
 - (1) Media Access Control (MAC) and Logical Link Control (LLC) layer.*
 - (2) Physical layer processing.*
- f. Onboard data switch.*

4.2.1.3 GPM Interfaces

- a. TT&C Interface.*
- b. Ground Test Interface.*
- c. Provides programmable general-purpose input output (GPIO) to support.*
 - (1) Interrupt source/sink.*
 - (2) Application data transfer.*
- d. Provides control/configuration interfaces.*
 - (1) RFM, antenna, power amplifier, and SPM.*
- e. System Bus interface.*

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

4.2.1.4 GPM Requirements

(STRS-2) A module's diagnostic information shall be available via the STRS APIs.

(STRS-109) An STRS platform shall have a GPM that contains and executes the STRS OE and the control portions of the STRS applications and services software.

4.2.2 Signal-Processing Module

An SPM is optional for an STRS platform. The SPM may implement the signal processing used to transform received digital signals into data packets and/or the conversion of data packets into digital signals to transmit. The complexity of this module is based on the applications and data rates selected for a mission. The SPM modules contain components and capabilities to manipulate and manage digital signals that need higher processing capabilities than that supplied by the GPM. The configurable hardware design architecture describes a common interface for the application on the SPM, as described in section 6. Figure 5, SPM Architecture Details, illustrates the SPM module details.

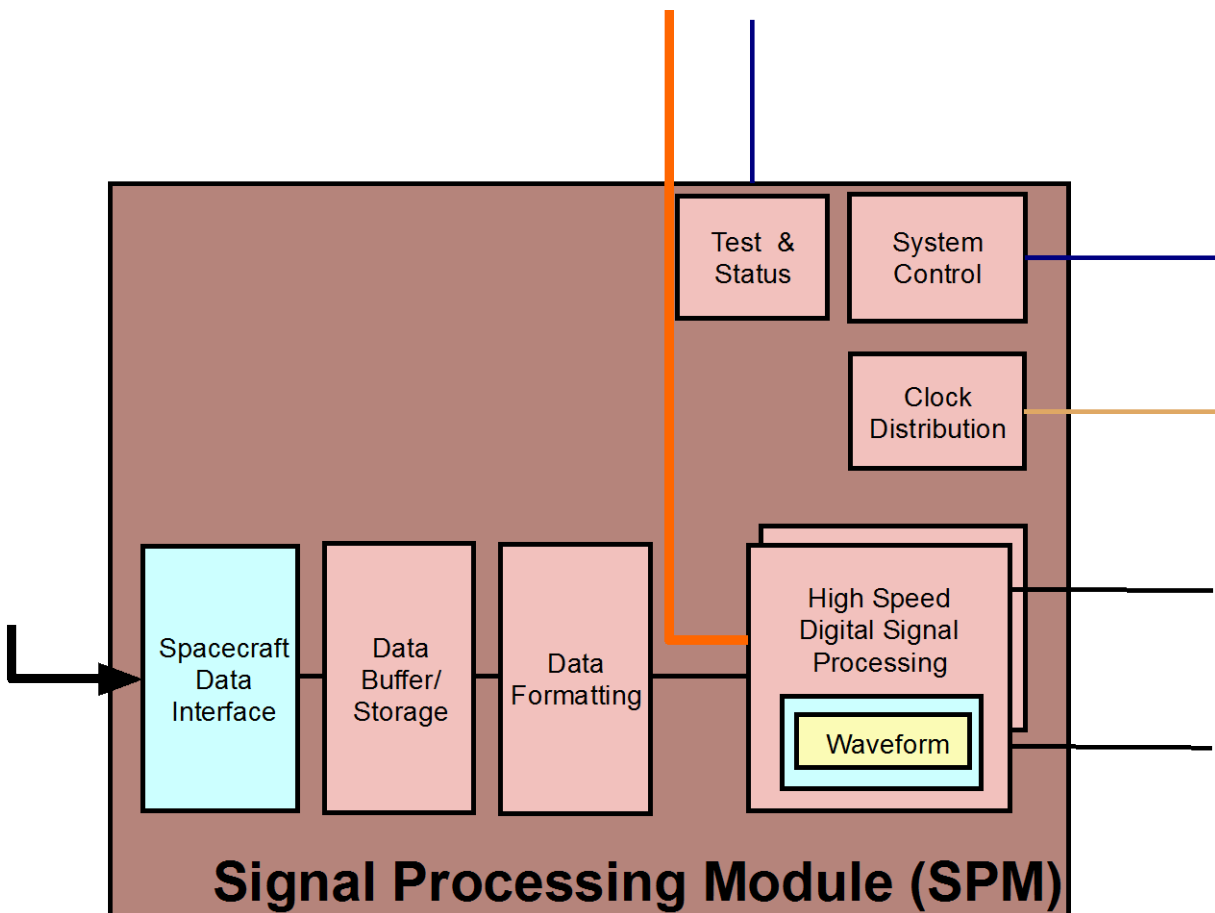


Figure 5—SPM Architecture Details

4.2.2.1 SPM Components

The SPM will initially be implemented primarily with FPGAs, DSPs, reconfigurable processors, ASICs, and other integrated circuits. However, technologies will change over time, so the specific implementation is left to the STRS platform provider.

It is also anticipated that STRS platforms may use dedicated SPM physical hardware slices (e.g., separate circuit boards) for specific applications and technologies. For example, a dedicated global positioning system (GPS) receiver slice can complement the existence of reconfigurable SPM slices in the same radio. The dedicated slice offloads demand on the less specific SPM. If an STRS platform contains an SPM slice, the slice should meet the module interface specifications for control and configuration and have an interface with the GPM via the GPM system bus and the SPM-to-GPM test interface. These two interfaces work in concert to provide a control and reprogramming path to the SPM from the GPM and the application running on the GPM.

4.2.2.2 SPM Functions

The SPM performs the digital signal-processing functions, which are used to convert symbols to bits (and vice versa). These functions are typically implemented on FPGAs, DSPs, or ASICs. It is recommended that reconfigurable and reprogrammable devices be used because this allows for new applications to be implemented on the SDR in the future without a hardware redesign. However, mission-specific requirements may dictate that the application be implemented on a non-reprogrammable hardware device.

In addition to the digital signal-processing functions, a data-formatting function is typically provided to convert blocks of data stored in the data storage element into bitstreams appropriate for encoding into symbols (and vice versa). In many cases, it is possible to implement the data-formatting function in the same device as the digital signal-processing function, but that is an implementation detail dependent on the mission class.

A data storage element is used to provide a queuing buffer between the data interface and the bitstream coders and decoders. This data storage function can be implemented in either volatile or nonvolatile memory, depending on the requirements of the mission implementation.

An SPM may implement any or all of the following digital communication functions depending upon the mission waveforms:

- *Digital up conversion—interpolation, filtering, and “local oscillator” multiplication of baseband samples to obtain an IF or RF output sample stream, appropriate for digital-to-analog conversion. This is typically the last transmit function implemented in the SPM, and the output samples are sent to the RFM.*
- *Digital down conversion—multiplication with “local oscillator,” downsampling, and filtering IF or RF samples to obtain a baseband output sample stream. This is*

typically the first receive function implemented in the SPM, with input samples coming from the analog-to-digital conversion in the RFM.

- *Digital filtering—averaging, low-pass, high-pass, band-pass, polyphase, and other filters used for pulse shaping, matched filter, etc. This may overlap with some of the functionality in the Up and Down Conversion.*
- *Carrier recovery and tracking—retrieval of the waveform carrier within the receive sample stream. Typical SPM functions for carrier recovery include shifting the recovered carrier frequency to compensate for local oscillator variations and Doppler shifts in the link.*
- *Synchronization (data, symbol, etc.)—alignment of received samples with symbol and data boundaries. There may be some integration with the Digital Down Conversion and Carrier Recovery and Tracking functions.*
- *Forward error correction coding—encoding transmit data so that receive data errors may be corrected to some level, enhancing the waveform performance.*
- *Digital automatic gain control (AGC)—scaling of the receive samples to optimize downstream operations.*
- *Symbol mapping (modulation)—translating transmit data bits to modulation symbol samples.*
- *Data detection (demodulation)—translating receive symbol samples to data bits.*
- *Spreading and despreading—a form of encoding data to obtain certain energy dispersion in the frequency domain.*
- *Scrambling and descrambling—a form of encoding data to ensure a certain level of randomness in the digital data stream, usually for synchronization of the receiver.*
- *Encryption and decryption—a form of encoding data for security purposes.*
- *Data Input/Output (I/O) (high-speed direct from or to source or sink)—interface for transmit and/or receive data to come in or out of the module. This may involve buffering and some protocol handling.*

4.2.2.3 SPM Interfaces

The SPM's functions and external interfaces are shown in Figure 5. Interfaces shown include those common to all module types as well as those specific for the SPM. These SPM-specific interfaces may not all be required for some missions. Note that the implementation of these interfaces may combine two or more on one physical transport. For example, the Data Interface and Control and Configuration Interfaces may both use the same physical Serial Rapid I/O connection.

- *Data I/O to or from RFM—This is the digital sample stream going to the RFM's DACs for transmission, and the digital samples from the RFM's ADCs. However, if the DACs and ADCs are preferred to be a part of the SPM, then this interface is replaced with analog baseband or IF signals.*

NASA-STD-4009A

- *Waveform control and feedback to RFM—This interface will be waveform dependent. It may be used, for example, to send feedback to an AGC or control frequency hopping.*
- *Data interface external to the radio—High-data-rate waveforms may need a direct interface to the SPM if the GPM is not designed to handle the data.*
- *System bus—Data to or from GPM—This interface exchanges the packetized data for transmission and reception.*
- *Control and configuration from GPM—Waveform loads and reconfigurable parameters are managed through this interface.*
- *Test and status to GPM—Tests are initiated through this interface by the GPM, and results are returned. This is a more basic interface (electrically and protocol-wise) than the Control and Configuration interface.*
- *Radiometric tracking.*

The HID is to contain the characteristics of each reconfigurable device. Reconfigurable capacity is usually measured by the number of FPGA gates, slices, logic elements, or bytes. This information can be used by future STRS application developers to determine the waveforms that can be implemented on a given platform.

4.2.3 Radio Frequency Module

The RFM handles the conversion to and from the carrier frequency, providing the SPM and/or the GPM with digital baseband or IF signals, and the transmission and reception equipment with RF to support the SPM and GPM functions. Its components typically include DACs, ADCs, RF switches, up converters, down converters, diplexer, filters, LNAs, power amplifiers, etc. Current and near-term RF technologies cannot be expected to allow multiband operation using a single channel RFM, and thus multiband radios will need to use multiple RFM slices. The RFM provides a band of frequency tunability on each slice. This tunability can be software controlled through the provided interfaces.

The RF module handles the interfaces that control the final stage of transmission or first stage of reception of the wireless signals, including antennas, optical telescopes, steerable antennas, external power amplifiers, diplexers, triplexers, RF switches, etc. These external radio equipment components would otherwise be integrated with the RFM except for the physical size and location constraints for transmission and reception. The interfaces are primarily the associated control interfaces for these components. The RFM HID encompasses the control and interface mechanism to the external components. The focus of the RF HID is to provide a standardized interface to the control of each of these devices, to synchronize the operation of the radio with any of these devices.

The other primary capability of the RFM is the conditioning and distribution of the frequency reference as defined by the Frequency Reference Interface. This provides a common reference

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

for the RFM and SPM modules to enable the tracking and navigation functionality typically provided by SDRs. Figure 6, RFM Architecture Details, illustrates the RFM module details.

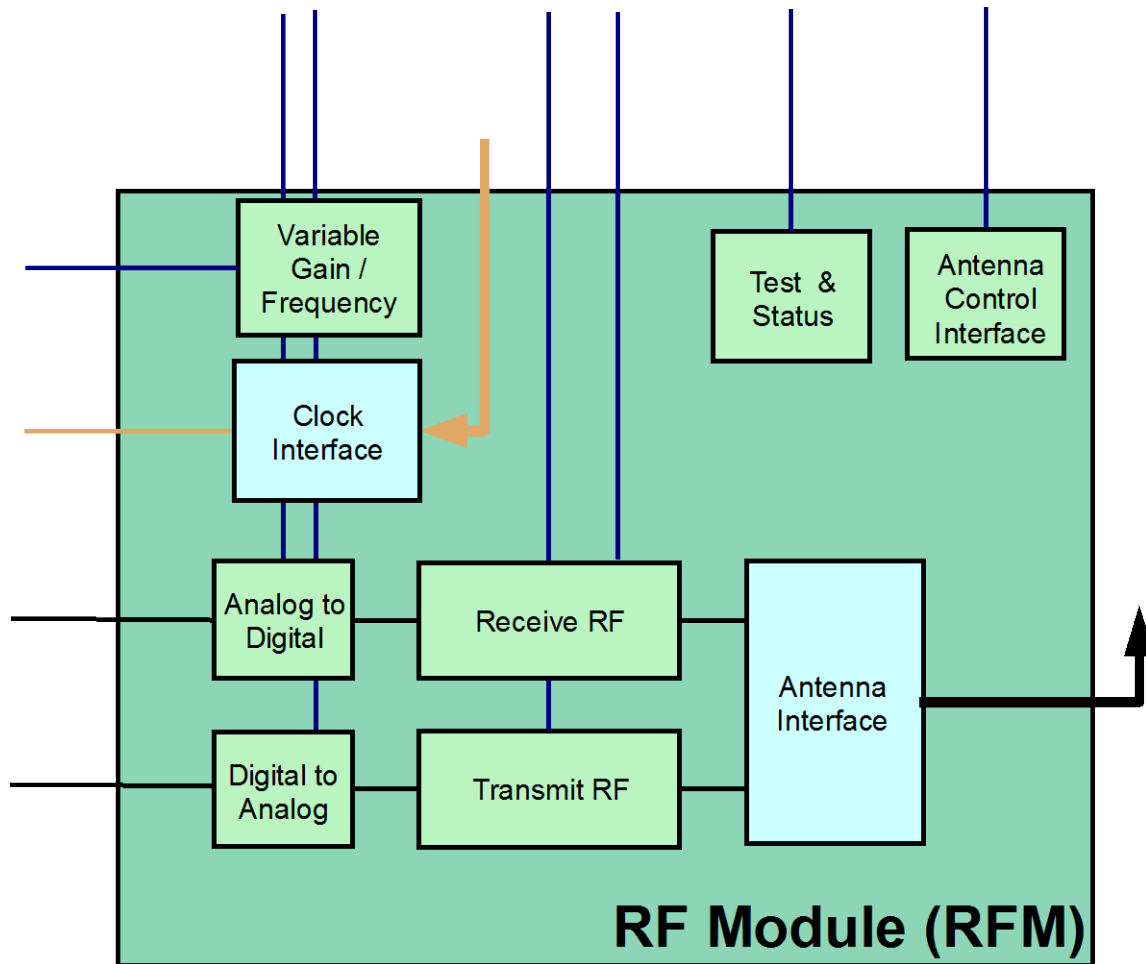


Figure 6—RFM Architecture Details

4.2.3.1 RFM Functions

The RFM transforms the antenna signal to or from a signal usable to the radio. The RFM functions are likely to include the following:

- Frequency conversion and gain control.
- Analog filtering.
- Analog-to-digital and digital-to-analog conversion.
- Radiometric tracking.

4.2.3.2 RFM Components

The RFM can be implemented with a variety of integrated circuits. The control of these circuits can be implemented with a variety of different component technologies, including ASICs, discrete electronics, programmable logic devices including FPGAs and DSPs, or even microprocessors. The choice of technologies is left up to the developer of the particular implementation. It is expected that the control of the devices will become more sophisticated over time and that the level of control will increase, resulting in more complex control circuitry and logic devices being used.

4.2.3.3 RFM Interface

- a. *External RF interface(s) to the radio.*
- b. *Provides read and write access to interface registers to monitor and perform control, status, and failure and fault-recovery functions (e.g., via RS-422 or SpaceWire).*
 - (1) *Control (power level tunability, frequency tunability, antenna parameter tunability, etc.).*
 - (2) *Status (report status of components and system operation).*
 - (3) *Failure and fault-recovery functions (detect component or system failure and determine appropriate action).*
- c. *Provides diagnostic test registers.*
- d. *Provides I/O for exchanging digitized waveform signal data.*

4.2.3.4 RFM Requirements

(STRS-6) The STRS platform provider shall describe, in the HID document, the behavior and performance of the RF modular component(s).

The behavior and performance of the RF modular components should be sufficiently described such that future waveform developments may take advantage of the RF capability and/or account for its performance. Information in the HID may include such items as center frequency, IF and RF frequency(s), bandwidth(s), IF and RF input/output level(s), dynamic range, sensitivity, overall noise figure, AGC, frequency accuracy and stability, and frequency-tuning resolution.

4.2.4 Security Module

The STRS architecture has been designed to address security concerns as part of the architecture. Although this section is currently not complete, the goal is to address the security services required from an SDR. This approach supports the evolutionary nature of the STRS architecture. It is expected that this section will be expanded as new technologies and operational modes are developed or extended.

The architecture will support selectable data-protection services for those users needing them, including both confidentiality and authentication. Missions may select security options provided by the infrastructure or may develop their own.

The authentication of commands sent to SDRs is supported, including changing the configuration or uploading new programs for either the infrastructure or new applications. The security section of the architecture will include support for key management, encryption standards, and mitigating threats other than the information and communication security threats currently identified.

4.2.5 Networking Module

The STRS architecture has been structured such that networks can be implemented in an SDR; that is, an SDR can be a node in a network. The SDR may be connected to another node using the appropriate logical and physical interfaces that may be wired and/or wireless. The STRS architecture can accommodate network protocols as services that can be made available to applications and devices. STRS supports the ability to upload new software and dynamic hardware images. Therefore, advancements and replacement of existing protocols can be accomplished without affecting a spacecraft's mission resources.

4.2.6 Optical Module

The STRS architecture supports the use of optical communications in SDRs. The use of optical communications techniques poses challenges in many areas, but optical communications also have the potential for great benefit. STRS interfacing to optical communication equipment follows the same techniques shown in integration with high-data-rate hardware. The OM would be controlled through the STRS HAL interface that allows configuration and control of the digital components in the module, which abstracts the optical functionality.

4.3 Hardware Interface Description

The STRS platform provider is to provide an HID document, which describes the physical interfaces, functionality, and performance of the entire platform and each platform module. The HID specifies the electrical interfaces, connector requirements, and all physical requirements for the delivered radio. The HID abstracts and describes the functionality and performance of each module. In this manner, STRS application developers can know the features and limitations of the platform for their applications. The information in the HID provides the knowledge for NASA and others to integrate and test the hardware interfaces. The information in the HID may allow future module replacement or additions without the design of a completely new platform. For example, a Security Module could be added that was not originally planned, or a follow-on mission could use a different frequency band and only an RFM change would be needed. Include all waveform interfaces and any other interfaces that could be important to a waveform developer or a hardware integrator.

NASA-STD-4009A

In addition to the GPM, SPM, and RFM HID descriptions and requirements stated within each module section, the following interface descriptions and requirements are also specified for an STRS platform.

(STRS-4) The STRS platform provider shall describe, in the HID document, the behavior and capability of each major module or component available for use by a waveform, service, or other application (e.g., FPGA, GPP, DSP, or memory), noting any operational limitations.

(STRS-5) The STRS platform provider shall describe, in the HID document, the reconfigurability behavior and capability of each reconfigurable component.

The description of the behavior and capability of modules or components available to STRS application developers or reconfigurable components may include device type, processing capability, clock speeds, memory size(s), types(s), and speed(s), noting any constraints, as well as any limitation on the number of configurable hardware design reloads, as applicable, partial reload ability, built-in functionality, and any corresponding restriction on the number of gates.

(STRS-7) The STRS platform provider shall describe, in the HID document, the interfaces that are provided to and from each modular component of the STRS platform.

The specific modular components or hardware slices of an STRS platform will vary among different implementations. The STRS platform provider or STRS integrator is expected to describe each modular component and their respective physical and logical interfaces as described in this section. Table 1, STRS Module Interface Characterization, provides typical interface characteristics that should be included when identifying external interfaces or internal interfaces between modules for STRS.

Table 1—STRS Module Interface Characterization

STRS Module Interface Characterization Table	
Parameter	Description and Comments
Name	Interface name (data, control, operating power, RF, security, etc.).
Interface type	Point to point, point-multipoint, multipoint, serial, bus, other.
Implementation level	Component, module, board, chassis, remote node.
Reference documents and standards	Applicable documents for interface standards or description of custom interfaces.
Notes and constraints	Variances from standards, physical and logical functional limitations.
Transfer speed	Clock speed, throughput speed.
Signal definition	Description of functionality and intended use.
Physical Implementation	
Technology	For example, GPP, DSP, FPGA, ASIC, and description.
Connectors	Model number, pin out (including unused pins).

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

STRS Module Interface Characterization Table	
Parameter	Description and Comments
Data plane	Width, speed, timing, data encoding, protocols.
Control plane	Control signals, control messages or commanding, interrupts, message protocol.
Functional Implementation	
Models	Data plane model, control plane model, test bench model.
Power	Voltages, currents, noise, conducted immunity, susceptibility.
APIs	Custom or standard, particular to OS environment.
Software	Device drivers, development environment, and tool chain.
Logical Implementation	
Addressing	Method, schemes.
Channels	Open, close.
Connection type	Forward, terminate, test.

4.3.1 Control and Data Interface

The control and data communications buses and links between modules within the radio are to be described by the STRS platform provider to the level of detail necessary to facilitate integration of another vendor's module. If modules communicate using the IEEE 1394, A High-Performance Serial Bus, interface, for example, this will be specified in the HID with appropriate connector and pinout information. Any nonstandard protocols used should also be specified. In some cases, this may be handled by the software HAL. Module interfaces will be completely described, including any unused pins.

(STRS-8) The STRS platform provider shall describe, in the HID document, the control, telemetry, and data mechanisms of each modular component (i.e., how to program or control each modular component of the platform, and how to use or access each device or software component, noting any proprietary and nonstandard aspects).

Besides the interface descriptions already provided for each modular component, developers should provide specific information necessary for future STRS application developers to know how to interact with the command and control aspects of the platform. The description of the control, telemetry, and data mechanism of each modular component should facilitate the porting of the application software to the platform.

4.3.2 Operating Power Interface

The operating power interface description for the radio has two parts: (1) the platform as a supplier to the various modules; and (2) the power consumption of the different modules, if multiple modules are provided.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Table 2, Example—Operating Power Interface (Platform Supplied), shows an example listing of a platform operating power interface. There are four distinct sets of power requirements for the platform shown. For each module delivered with the radio, as well as those built by other vendors, the HID is to specify the needed voltages, currents, and connections. Voltages are to be specified with a maximum and minimum tolerance, and associated currents are to be specified with nominal and maximum values. Connectors for operating power are to be specified, including pinouts. If power is routed through a multipurpose connector such as a backplane connector, then the pins actually used are to be documented. Power is a limited commodity for most missions, and understanding the STRS platform power needs is critical.

Table 2—Example—Operating Power Interface (Platform Supplied)

Parameter	Values			
Voltage available	–15 V	+2.5 V	+5 V	+15 V
Maximum current/chassis (platform)	2 A	1.7 A	3 A	2 A
Maximum current/slot (module)	1 A	1 A	1 A	1 A
Backplane supply pins	17, 19	59, 61	44, 46, 48	21, 23
Backplane return pins	18, 20	60, 62	43, 45, 47	22, 24
Connector type	-----	-----	-----	-----
Voltage ripple	100 mVpp	1 mVpp	5 mVpp	100 mVpp
Notes	Slot 1 and 2 only	-----	-----	Slot 1 and 2 only

(STRS-9) The STRS platform provider shall describe, in the HID document, the behavior and performance of any power supply or power converter modular component(s).

4.3.3 Thermal Interface and Power Consumption

The power consumption and resulting heat generation of a reprogrammable FPGA will vary according to the amount of logic used, the switching rate of the waveform logic, and the clock frequency(s). The power consumption may not be constant for each possible waveform that can be loaded on the platform. The STRS platform provider should document the maximum allowable power available and thermal dissipation of the FPGA(s) on the basis of the maximum allowable thermal constraints of FPGA(s) of the platform. For human spaceflight environments, touch temperature requirements may limit dissipation further; therefore, these reductions are to be factored into the given dissipation limits.

(STRS-108) The STRS platform provider shall describe, in the HID document, the thermal and power limits of the hardware at the smallest modular level to which power is controlled.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

5. APPLICATIONS

5.1 Application Implementation

As shown in Figure 7, *Waveform Component Instantiation*, an example STRS platform consists of one or more GPMs with GPPs, and optionally one or more SPMs containing DSPs, FPGAs, and ASICs. Application (waveform and service) components loaded and executed on these modules provide the signal-processing algorithms necessary to generate or receive RF signals. To aid portability and reusability, the applications are to use the appropriate infrastructure APIs to access platform services. Using “direct to hardware” access instead would increase the effort to port the application to a platform with different hardware. The STRS infrastructure provides the APIs and services necessary to load, verify, execute, change parameters, terminate, or unload an application. The STRS infrastructure utilizes the HAL to abstract communications with the specialized hardware, whereas the HID identifies the hardware interfaces and how modules are physically integrated on a platform.

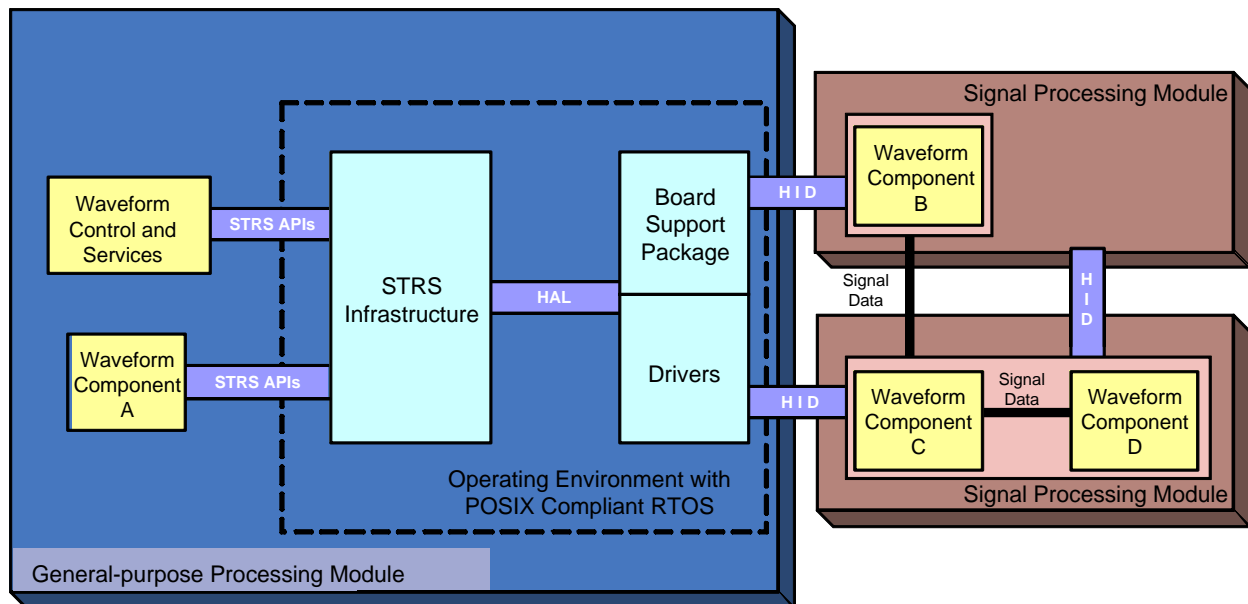


Figure 7—Waveform Component Instantiation

(STRS-10) An STRS application shall use the STRS infrastructure-provided APIs and POSIX® API for access to platform resources.

(STRS-11) The STRS infrastructure shall use the STRS platform HAL APIs to communicate with application components on the platform specialized hardware via the physical interface defined by the STRS platform provider.

5.2 Application Selection

STRS platform providers have the option of providing telemetry values to indicate what types of applications are installed. The method for selecting the application will be a combination of the platform's capabilities as well as the specification defined by the STRS Command and Telemetry interfaces in section 8.

STRS allows two types of configuration files: a platform-specific component, and an application-specific component. An application-specific configuration file specifies information used to initialize an STRS application. It is up to the project manager/platform provider to decide whether either of these is defined and, if so, what the format is. Suggestions are discussed in the STRS Handbook, NASA-HDBK-4009A.

5.3 Application Repository Submissions

The STRS architecture facilitates the use of reusable and highly reliable applications. Highly reliable and reusable applications require good coding practices, good documentation, and thorough testing. The documentation and application artifacts are to be submitted to the NASA STRS Application Repository, <https://strs.grc.nasa.gov/repository/>. The use of the artifacts in the NASA STRS Application Repository will be subject to the appropriate license agreements. Therefore, the agreements defining the release, distribution, and ownership of the artifacts are to be submitted to the repository including license agreements, type of release, and any restrictions. Types of releases are discussed in NPR 2210.1, Release of NASA Software. NASA will provide the STRS application developer information on the requests and distribution of items and lessons learned using the application. If the STRS application developer receives independent requests for the application, this request should be forwarded to the NASA STRS Application Repository manager to assure process consistency.

The goal of the NASA STRS Application Repository is to reduce future application development time and porting time since STRS application developers will have access to validated code. The STRS Application Repository is an archive of the developed configurable hardware design and software for the various applications. The repository allows STRS application developers access to existing STRS application artifacts that have been populated by NASA and STRS application developers. The documentation of STRS application behavior should include the STRS application developer's implementation of the STRS Application-provided Application Control API methods as described in section 7.3.1.

The documentation can also include a Design Description Document, HID, HAL, Verification and Validation (V&V) Plan, V&V Procedure, and V&V Results, OE-Specific Developer's Guide and User's Guide.

(STRS-12) The following application or OE development artifacts shall be submitted to the NASA STRS Application Repository:

NASA-STD-4009A

- (1) Application (or OE) or system component software and configurable hardware design simulation model(s) and/or documentation. (Design Description Document)
- (2) Documentation of external interfaces for STRS application, devices, or configurable hardware design (e.g., signal names, descriptions, polarity, format, data type, and timing constraints). (HID)
- (3) Documentation of STRS application or OE behavior and adaptability (e.g., configurable and queryable data items). (Design Description Document, User's Guide)
- (4) Application or OE function sources (e.g., C, C++, header files, very-high-speed integrated circuit HDL (VHDL), and Verilog). (Artifacts)
- (5) Application or OE libraries, if applicable (e.g., electronic design interchange format (EDIF), dynamic link library (DLL)). (Artifacts)
- (6) Documentation of application (or OE) development environment and/or tool suite as follows: (Design Description Document)
 - A. Include the development environment and/or tool suite name, purpose, developer, version, and configuration specifics (e.g., ISE Design Suite System, Xilinx, 14.4, EDK and SDK; MATLAB®, Model base design support automatic code generation, MathWorks, R2016a).
 - B. Include a description of the hardware on which the development environment and/or tool suite is executed, its OS, OS developer, OS version, and OS configuration specifics (e.g., Microsoft® Windows 7, Service pack 2; Linux® Ubuntu, (Xenial Xerus) 16.04).
 - C. Include a description of the output of the development environment and/or tool suite, its STRS infrastructure/OE description, developer, version, and unique implementation items (e.g., Type of file, .mdl, .slx; GRC's STRS Reference Implementation; IP generated from Xilinx).
 - D. Include a description of licensing agreements for development environment and/or tool suite.
- (7) Test plans, procedures, and results documentation. (V&V Plan, V&V Procedure, and V&V Results)
- (8) Identification of software development standards used. (Version Description Document (VDD)/Metadata)
- (9) Version of this NASA Technical Standard used. (VDD/Metadata)
- (10) Information, along with supporting documentation, required to make the appropriate decisions regarding ownership, distribution rights, and release (technology transfer) of the application or OE and associated artifacts. (Transfer Rights/Agreements)
- (11) Version Description Document, if available, or other document containing the version number of each separable artifact in the release, defined down to the lowest level components. (VDD)

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

- (12) Documentation of the platform component hardware used by the application or OE, its function and the interconnections. If the component executes an operating system, document the OS, OS developer, OS version, and OS configuration. (HID)
- (13) Documentation when an OE is submitted to the STRS Application Repository, providing guidelines to aid a waveform/application developer and integrator in the task of developing an STRS compliant waveform/application. (OE-Specific Developer's Guide)

6. CONFIGURABLE HARDWARE DESIGN ARCHITECTURE

Configurable hardware design is embedded in a hardware device, such as an FPGA. Configurable hardware design is distinguished from software residing in a GPP, ease depends on OS/OE. This section addresses the use of configurable hardware design from design and development through testing and verification and operations. It addresses aspects of model-based design techniques and design for space environment applications.

Proper testing of configurable hardware design is critical in the development of reliable and reusable code. Development tools that enable early development and testing should be used so that problems can be identified and resolved early in the SDR life cycle. Many real-world signal degradations and SEUs can be simulated to identify potential issues with the waveform and waveform functions early in development, even before hardware is available. Applications implemented in configurable hardware should be modular with clear interfaces to enable individual application component simulations and incremental testing.

The configurable hardware design architecture supports the modeling of STRS applications implemented in configurable hardware at the system, subsystem, and function levels. Model-based design techniques aid in the development of modular application functions. Application development models done in a platform (or target) independent manner aid in application testing, reuse, and portability. A platform-independent model (PIM) design can be used to target different platforms. PIM design flows might include high level models combined with manual code writing. On resource-constrained platforms, optimized code would be written. On non-resource-constrained platforms, PIMs may be used to auto generate code. These design flows can be employed to significantly reduce the porting effort.

Application portability and reusability should be considered in all facets of the design process from concept to implementation to testing. The coding technique of the application is also essential to reduce the application porting effort. Having defined syntax standards for HDLs (e.g., Verilog or VHDL) makes them appear to be easily portable across devices and software synthesizers, but this is an incorrect assumption. There are many things that can make hardware description languages hard to port. For example, the use of device-specific fixed hardware logic on the FPGA will decrease the portability. The use of specialized hardware may be necessary to meet the timing constraints of the application; however, the STRS application developer should document any application function that uses the specialized hardware so that the effort to port

the application function(s) can be determined. Non-boolean-type logic such as clock generation can also reduce portability. One method to decrease the porting effort would be to create a module that does the clock generation from which the rest of the application functions receive the necessary clock(s).

Development of configurable hardware design for STRS radios should include provisions for mitigating space environmental effects such as SEUs. Near-term application of static random access memory (SRAM)-based FPGAs may require triple-mode redundancy (TMR), configuration memory scrubbing, and other mitigation techniques, depending on the intended mission environment and desired reliability. Commercial design tools are becoming available to aid in this process and some newer FPGAs have versions available with embedded TMR.

A key feature of SDRs is that they can be reconfigured after deployment. The ability to load new applications and services will benefit missions in several ways, including using one SDR (instead of several separate radios) to handle different applications for various phases of a mission, some planned and some unplanned. An STRS platform should receive STRS application software and configurable hardware design updates after deployment.

6.1 Specialized Hardware Interfaces

Standardizing and documenting the interface from the waveform applications on the GPP to the portion of the waveform in the specialized processing hardware, such as FPGAs, is intended to provide commonality among different STRS platforms and to aid portability of application functional components implemented in configurable hardware design. Figure 8, Notional High-Level Software and Configurable Hardware Design Waveform Application Interfaces, depicts the high-level interface relationship between GPM, SPM, and RFM modules in an STRS radio.

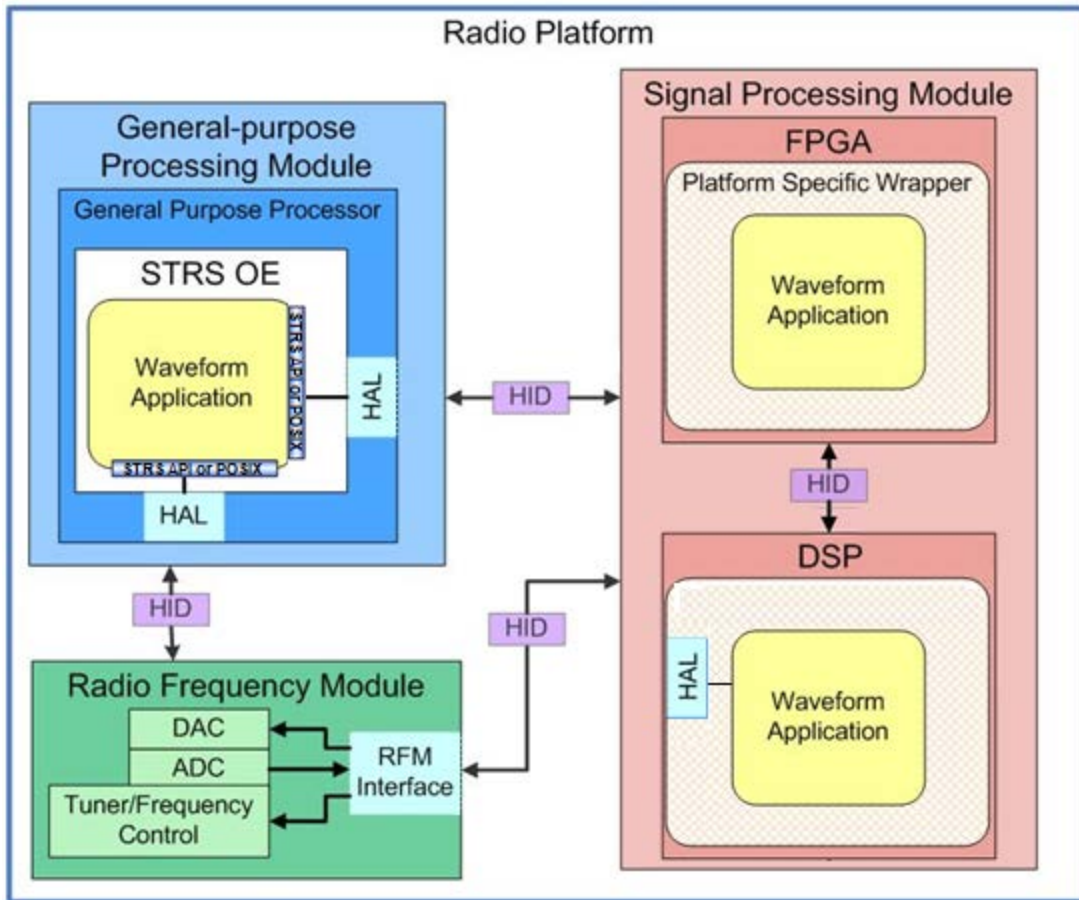


Figure 8—Notional High-Level Software and Configurable Hardware Design Waveform Application Interfaces

The STRS architecture provides a common mechanism for the software to instantiate, configure, and execute the software and configurable hardware design applications on various platforms using different hardware devices. Reconfiguration may include changing the parameters of installed applications and uploading new applications after deployment.

The application accepts configuration and control commands from the GPM and uses STRS APIs or POSIX® APIs that interface to the device drivers associated with the SPM and RFM modules. The device drivers communicate via the HAL on the GPM that abstracts the physical interface specification described in the HID in transferring command and data information between the modules.

For FPGAs, the interface to the application is through a platform-specific wrapper. The platform-specific wrapper accepts command and data information from the GPM and provides them to the application. The platform-specific wrapper also abstracts details of the platform from the STRS application developer, such as pinout information. The platform-specific wrapper should also provide clock generation, signal registering, and synchronization functions, and any other non-waveform-specific functions that the platform requires.

Documentation of the platform-specific wrapper is necessary so that STRS application developers can interface applications to the platform. This documentation should include detailed timing constraints, such as signal hold times, minimum pulse widths, and duty cycles. The signal timing constraints refer to the protocol of a particular interface describing events happening on a particular clock cycle. For clock generation, one should document what clock domains are in the design, how each clock is generated, and the resources that are involved. Signal synchronization describes any additional logic needed when clock domains are changed across the interface. The signal registering methods refer to any configurable hardware design interfaces between modules and if the input and output were registered, latched, or neither.

(STRS-13) If the STRS application has a component resident outside the GPM (e.g., in configurable hardware design), then the component shall be controllable from the STRS OE.

(STRS-14) The STRS SPM developer shall provide a platform-specific wrapper for each user-programmable FPGA, which performs the following functions:

- (1) Provides an interface for command and data from the GPM to the waveform application.
- (2) Provides the platform-specific pinout for the STRS application developer. *This may be a complete abstraction of the actual FPGA pinouts with only waveform application signal names provided.*

(STRS-15) The STRS SPM developer shall provide documentation on the configurable hardware design interfaces of the platform-specific wrapper for each user-programmable FPGA, which describes the following:

- (1) Signal names and descriptions.
- (2) Signal polarity, format, and data type.
- (3) Signal direction.
- (4) Signal-timing constraints.
- (5) Clock generation and synchronization methods.
- (6) Signal-registering methods.
- (7) Identification of development tool set used.
- (8) Any included noninterface functionality.

7. SOFTWARE ARCHITECTURE

7.1 Software Layer Interfaces

The STRS architecture is predicated on the need to provide a consistent and extensible development environment on which to construct NASA space applications. The breadth of this goal implies that the specification be based on the following: (1) Core interfaces that allow flexibility in the development of application software; and (2) HIDs that enable technology infusion.

The software architecture model shows the relationship between the software layers expected in an STRS-compliant radio. The model illustrates the different software elements used in the software execution and defines the software interface layers between applications and the OE and the interface between the OE and the hardware platform.

Figure 9, STRS Software Execution Model, represents the software architecture execution model. The software model achieves the following objectives:

- a. Abstracts the application from the underlying OE software to promote portability and reusability of the application.*
- b. Within the abstraction layer, minimizes custom routines by using commercial software standard interfaces such as POSIX®.*
- c. Depicts the STRS software components as layers to specify their relationship to each other and their separation from each other which enables developers to implement the layers differently according to their needs while still complying with the architecture.*
- d. Introduces a lower-level abstraction layer between the OE and the platform hardware.*

Note that although software abstraction for general processors is typically accomplished with board support packages and device drivers, the abstraction of hardware languages or configurable hardware design is less defined. The model represents the software and configurable hardware design abstraction in this layer.

- e. Indicates the relationship between the OE software and the different hardware processing elements (e.g., processor and specialized hardware).*

The OE adheres to the interface descriptions provided in Figure 9. This NASA Technical Standard provides two primary interface definitions, as follows: (1) The STRS APIs; and (2) The STRS HAL specification, each with a control and data plane specification for interchanging configuration and run-time data. The STRS APIs provide the interfaces that allow applications to be instantiated and use platform services. These APIs also enable communication between application components. The HAL specification describes the physical and logical interfaces for intermodule and intramodule integration.

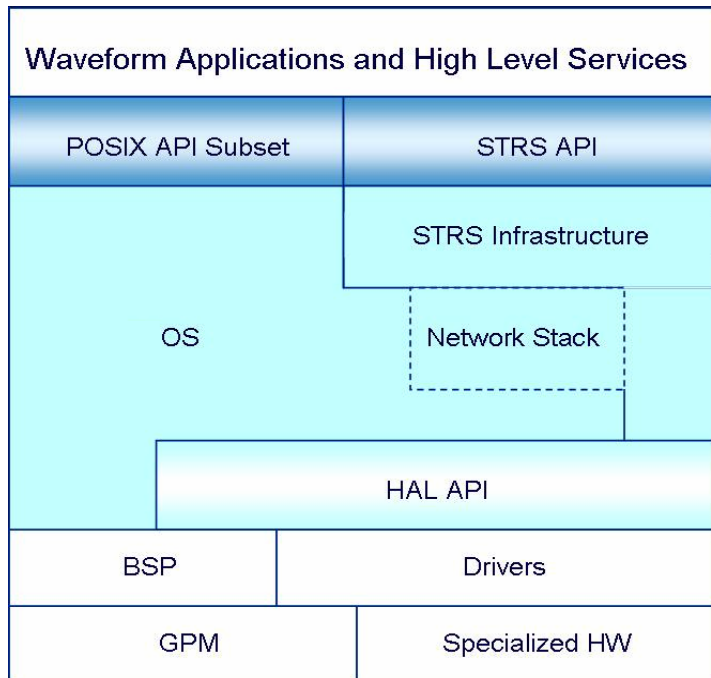


Figure 9—STRS Software Execution Model

The STRS software architecture presents a consistent set of APIs to allow waveform applications, services, and communication equipment to interoperate in meeting an application specification. Figure 10, STRS Layered Structure in UML, represents a view of the platform OE that depicts the boundaries between the STRS infrastructure provided by the STRS platform provider and the components that can be developed by third-party vendors (e.g., waveform applications and services).

A key enabler of application portability and reusability is the removal of application dependencies on the infrastructure that take advantage of explicit knowledge of the infrastructure implementation. When waveforms and services conform to the API specification, they are easier to port to other STRS platform implementations.

Figure 10 extends the view of the software architecture from the diagram introduced in Figure 9 to include additional detail of the infrastructure, POSIX®-conformant OS, and hardware platform. The STRS Software Execution Model (Figure 9) was transformed using the Unified Modeling Language (UML). The UML supports the description of the software systems using an object-oriented style. This approach clarifies the interfaces between components, adding additional detail. Table 3, STRS Architecture Subsystem Key, provides a key that describes the interaction between elements of the architecture.

NASA-STD-4009A

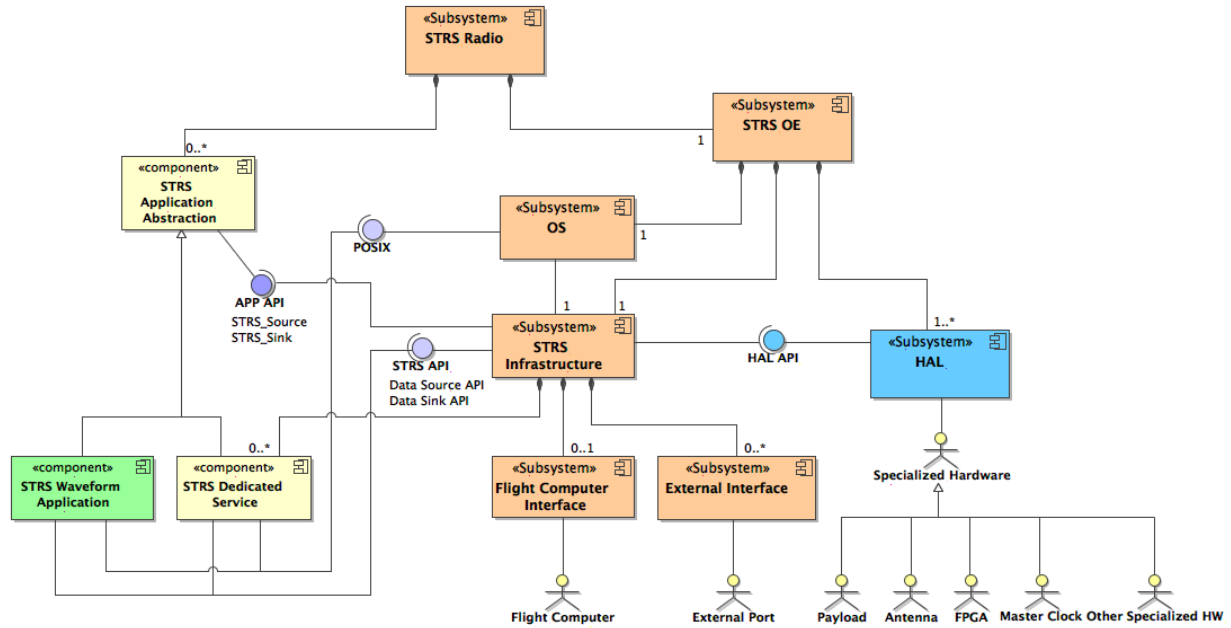


Figure 10—STRS Layered Structure in UML

Table 3—STRS Architecture Subsystem Key


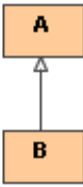
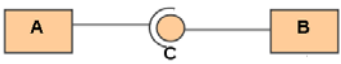
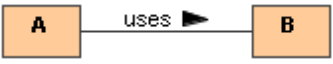
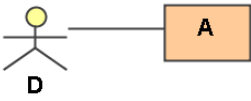
Diagram Element	Name	Explanation
	Composition	A contains X items of type B. B is a part of the aggregate A. B does not exist independently from A. X may be a number or a range from <i>m</i> to <i>n</i> depicted by “ <i>m...n</i> ” where <i>n</i> may be an asterisk to indicate no upper limit.
	Generalization or Inheritance	B is derived from A. B is a kind of A. B inherits all the properties of A. A is a more general case of B. Since B is more specialized, it frequently contains some additional attributes and/or more functionality than A.
	Interface	C is an interface provided by B; that is, C contains the means to invoke behavior that resides in B. A uses interface C to access B.
	Association	A is associated with B. The optional description “uses” indicates that A is associated with B such that A “uses” B.
	Association	D acts upon A, and A responds to D, or possibly vice versa. D is normally an actor outside the system.

Figure 11, STRS Operating Environment, describes the elements of the detailed OE depicted in Figure 9. In the case that the OS does not support the POSIX® subset, the missing functionality is to be implemented in the STRS infrastructure. Figure 11 also illustrates the inclusion of a POSIX® abstraction layer in the infrastructure. As a note, this abstraction is not only for a non-POSIX® OS, but the POSIX® abstraction layer would implement any POSIX® functions required but not implemented by the OS.

In Figure 11 the arrows identify interface dependencies and isolations. The waveform applications will not directly call the driver API but use the provided STRS APIs, thus providing the “abstraction layer” that helps isolate the application from the platform.

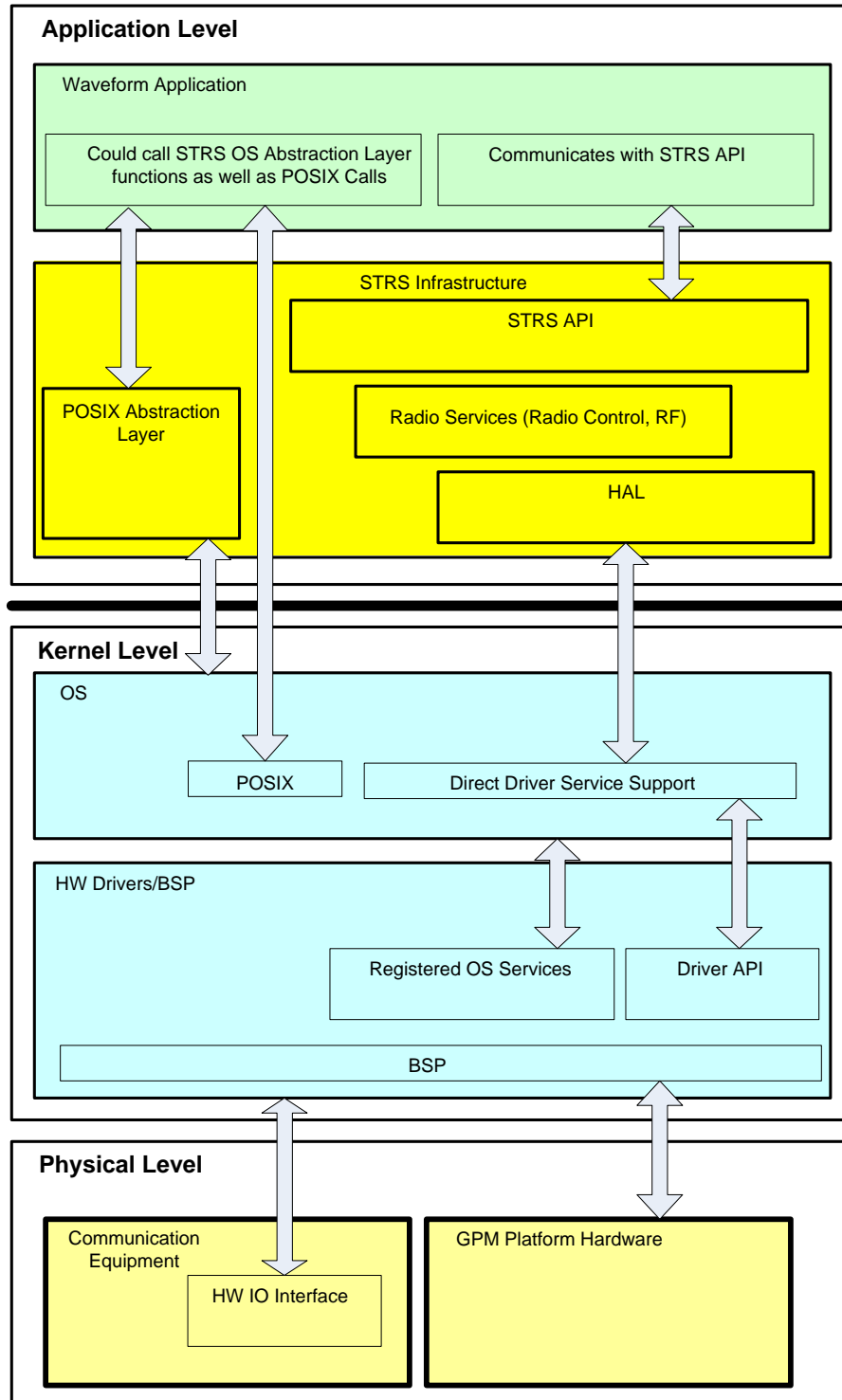


Figure 11—STRS Operating Environment

In Table 4, STRS Software Component Descriptions, the different layers of the STRS software model are described.

NASA-STD-4009A

Table 4—STRS Software Component Descriptions

Layer	Description
Waveform application and services	Waveform application and services provide the radio GPP functionality using the STRS infrastructure.
STRS infrastructure	The STRS infrastructure implements the behavior and functionality identified by the STRS APIs as well as other required radio functionality.
STRS API	The STRS APIs provides consistent interfaces for the STRS infrastructure to control applications and services, and for the applications and services to access STRS infrastructure services.
APP API	The APP API is the interface implemented by waveforms and services whose functions are used by the STRS infrastructure.
POSIX® Abstraction Layer	This optional interface (see Figure 12, POSIX®-Compliant Versus POSIX®-Conformant OS) provides POSIX® OS services to the waveform application and services on platforms with an OS that does not provide POSIX® interfaces.
Radio control services	These services are responsible for handling the radio commands and telemetry for the STRS. Applications use the STRS interface to communicate telemetry and receive commands from flight computer.
HAL	The HAL provides the device control interfaces that are responsible for all access to the hardware devices in the STRS radio. The HAL API is the interface to the software drivers and BSP that communicates with the hardware.
POSIX® API	The STRS defines a minimum POSIX® application environment profile (AEP) for the allowed OS services. The POSIX® AEP can be implemented by either a POSIX®-conformant OS or by a POSIX® Abstraction Layer in conjunction with a nonconformant OS.
OS	This is the operating system that supports the POSIX® API and other OS services. The POSIX® Abstraction Layer will provide applications with a consistent AEP interface that is mapped into the chosen OS functions.
POSIX® OS	This is the STRS POSIX® AEP-conformant portion of the OS.
Direct service support	This layer identifies the ability for the STRS infrastructure to have a direct interface to the hardware drivers on the platform.
HW drivers/BSP	The hardware drivers provide the platform independence to the software and infrastructure by abstracting the physical hardware interfaces into a consistent device control API.
Registered OS services	These are services that are integrated with the chosen OS to provide services such as MAC-layer interface to physical Ethernet hardware.
Driver API	OS-supplied APIs are abstracted from applications via the device control API.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Layer	Description
BSP	The BSP is the software that implements the device drivers and parts of the kernel for a specific piece of hardware. It provides the hardware abstraction of the GPM module for the POSIX®-compliant OS. A BSP contains source files, binary files, or both. A BSP contains an original equipment manufacturer (OEM) adaptation layer (OAL), which includes a boot loader for initializing the hardware and loading the OS image. Essentially, the OAL is all of the software that is hardware specific. The OAL is actually compiled and linked into the embedded OS.
HW I/O interfaces	Device drivers have been created for these physical interfaces.
GPM	This is the general-purpose processing module on which the STRS infrastructure executes.
Specialized hardware	This is the physical layer of the hardware modules existing on the STRS platform.

Figure 12 illustrates the difference between a POSIX®-conformant OS and a nonconformant OS. On the left side, the POSIX® AEP is provided entirely by the OS. The POSIX® APIs are included in those for the OS. On the right side, the OS is not POSIX® AEP conformant but is partially compliant. The POSIX® AEP is shown in two parts. One part shows the POSIX® APIs that are included in the OS. The other part shows the part of the POSIX® AEP that is not provided by the OS but is to be provided as the POSIX® abstraction layer. The STRS OE includes a POSIX® PSE51-conformant OS or POSIX® abstraction layer for missing APIs.

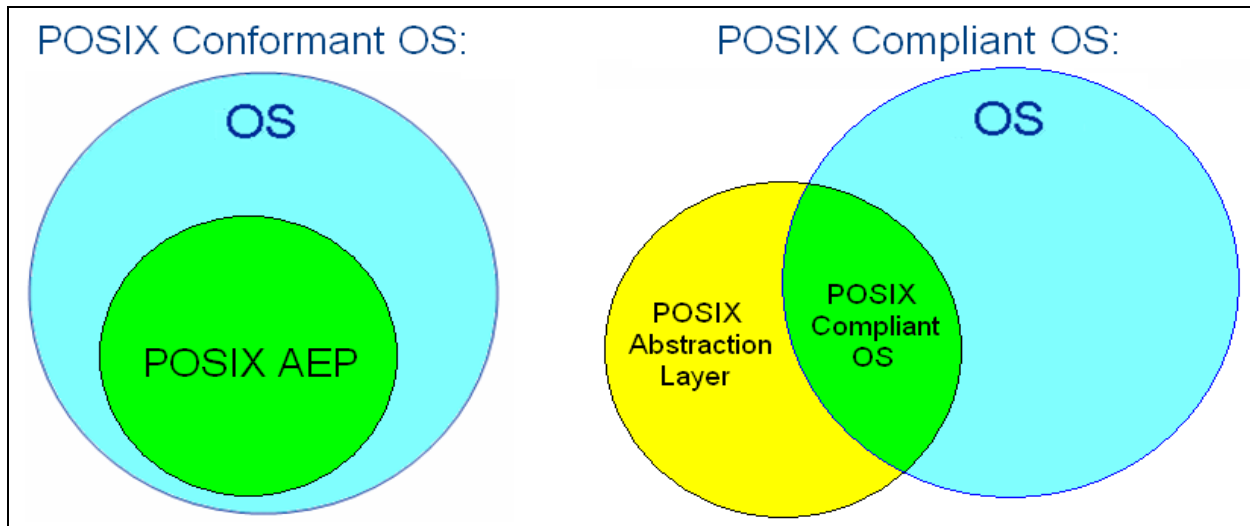


Figure 12—POSIX®-Compliant Versus POSIX®-Conformant OS

7.2 Infrastructure

The STRS infrastructure is part of the OE and provides the functionality for the interfaces defined by the STRS APIs specification. The infrastructure exposes a standard set of method names to the applications to facilitate portability. Although the STRS infrastructure may use any combination of POSIX®, OS, BSP functions, or other infrastructure methods to implement a radio function, which may vary on different platforms, the STRS APIs will be the same to allow portability. The STRS APIs are the well-defined set of interfaces used by STRS applications to access specific radio functions or used by the infrastructure to control the applications.

The infrastructure is composed of multiple subsystems that interoperate to provide the functionality to operate the radio. The components shown in Figure 13, STRS Infrastructure, represent the high-level subsystems and services needed to control STRS applications within the STRS platform. These services are provided by the platform infrastructure and support applications as they execute within the STRS platform. The infrastructure functions will include fault management techniques, which are necessary to increase radio robustness and support mission-dependent requirements. In order to support one of the primary objectives of the STRS (upgradeability), an STRS platform should be able to receive updated versions of the OE to support applications developed for newer versions of this NASA Technical Standard, after deployment.

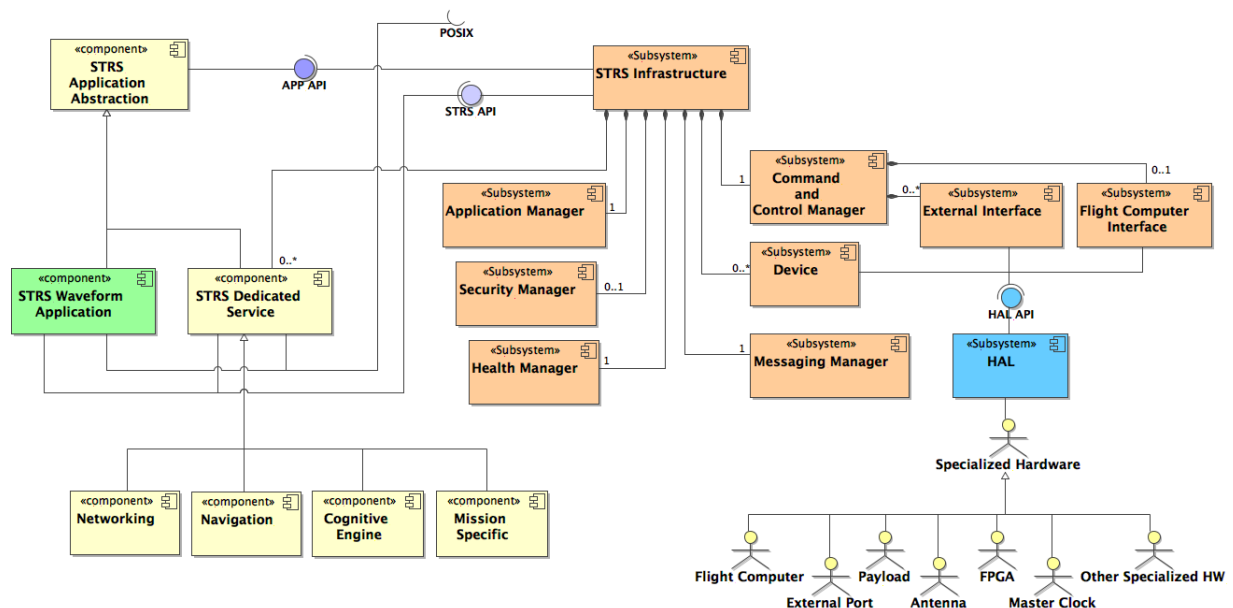


Figure 13—STRS Infrastructure

7.3 STRS APIs

The STRS APIs provide an open software specification so that the application engineers can develop STRS applications. The goal is to have a standard API available to cover all application program requirements so that the application programs can be reused on other hardware systems with minimal porting effort and cost for the application implemented in software and/or configurable hardware design with increased reliability. Size, weight, and power constraints may limit the functionality of the radio by imposing a tradeoff among the following: (1) The size of the API implementation; (2) The size of other internal operations; and (3) The size of the waveforms and services. The size of the selected GPP should be sufficient to contain the OS, the STRS infrastructure, and the appropriate portion of the waveforms and services to implement the required mission functionality, along with sufficient margin to support software upgrades. The STRS APIs are defined to support internal radio commands. The external interface commands, described in section 9, often use the internal commands defined by the STRS APIs to accomplish normal radio operations.

The API layer specification decouples the intellectual property rights of platform, application, and module developers. The API layer allows development and interoperability of different radio aspects while protecting the investment of the developers. The definitions of the APIs are based on a set of sequence diagrams derived from the use cases identified in Appendix A of the NASA/TP-2008-214813, STRS Software Architecture Concepts and Analysis, document.

The APIs are defined in the following sections. The APIs are grouped by type to simplify the description of the APIs while providing the detail for each requirement in tabular form. The table contains the name, description, calling sequence, return type, any preconditions, any post conditions, and examples. The examples shown in the table for each requirement are written from the point of view of the STRS application developer. The calling sequences for the infrastructure-provided APIs are callable from C language implementations of the STRS applications. If coding is done in C++, the infrastructure-provided API methods do not belong to any class and should be defined using extern "C."

The same handle name refers to the same application, device, file, queue, timer, or service across all applications. For information about errors, see section 7.3.12.

(STRS-105) The STRS infrastructure APIs shall have an ISO/IEC C language compatible interface.

7.3.1 STRS Application-Provided Application Control API

A key aspect of a software-architecture is the definition of the APIs that are used to facilitate software configuration and control of the target platform. The philosophy on which the STRS architecture is based avoids the conflict between open architecture and proprietary implementations by specifying a minimum set of APIs that are used to execute waveform applications and to deliver data and control messages to installed hardware components. The following APIs exhibit similar functionality to a resource interface in the Object Management

NASA-STD-4009A

Group (OMG)/software radio (SWRADIO) or Software Communications Architecture (SCA 2.2.2).

As shown in Figure 14, *STRS Application and Device Structure*, an STRS application implementation (e.g., waveform) is derived from the *STRS_ApplicationControl* API, the *STRS_Source* API when implementing *APP_Read*, and the *STRS_Sink* API when implementing *APP_Write*. The interfaces are implemented in groups so that *STRS_ApplicationControl* is derived from the *STRS_LifeCycle*, *STRS_PropertySet*, *STRS_TestableObject*, *STRS_ControllableComponent*, and *STRS_ComponentIdentifier* interfaces.

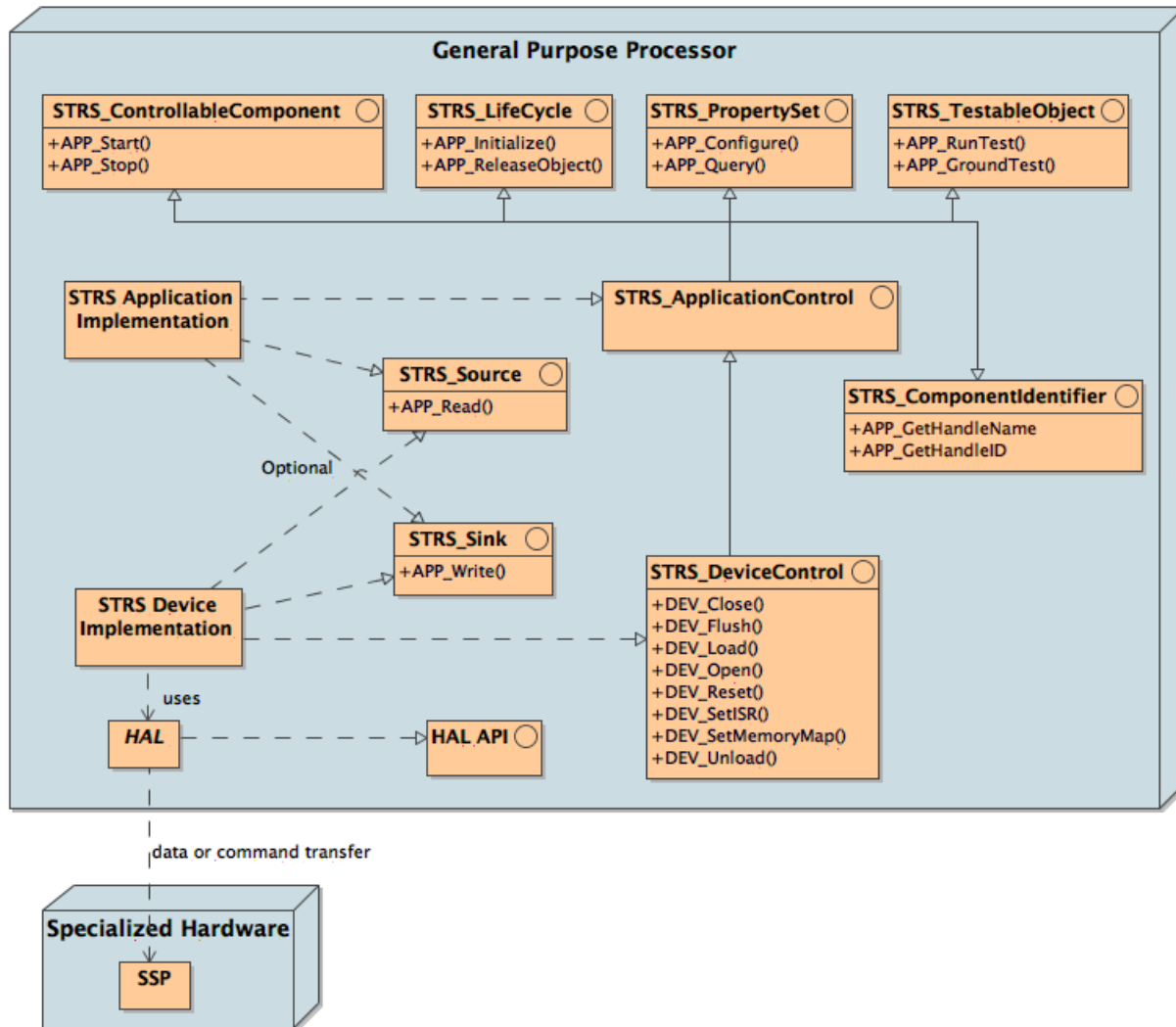


Figure 14—STRS Application and Device Structure

STRS requires a C and C++ standard based on ISO/IEC 9899, *Information technology—Programming languages—C*, and ISO/IEC 14882, *Information technology—Programming languages—C++*, respectively. In the USA, this is INCITS/ISO/IEC 9899:year and INCITS/ISO/IEC 14882:year, respectively, where the year will change periodically. The year is

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

not included in the requirement so that obsolete compilers are not mandated. In the USA, the InterNational Committee for Information Technology Standards (INCITS) coordinates technical standards activity between American National Standards Institute (ANSI) in the USA and joint ISO/IEC committees worldwide. INCITS is not included in the requirement so that the country of implementation may use its compilers.

(STRS-16) The STRS Application-provided Application Control API shall be implemented using ISO/IEC C or C++.

(STRS-17) The STRS infrastructure shall use the STRS Application-provided Application Control API to control STRS applications.

An OE may support applications written in either C, C++, or both. An application written for an OE that supports only C++ will entail extra effort to port it to an OE that supports only C and vice versa.

(STRS-18) The STRS OE shall support ISO/IEC C or C++, or both, language interfaces for the STRS Application-provided Application Control API at compile-time.

(STRS-19) The STRS OE shall support ISO/IEC C or C++, or both, language interfaces for the STRS Application-provided Application Control API at run-time.

The same include files are used for either C or C++ to access the appropriate prototypes.

(STRS-20) Each STRS application shall contain
#include "STRS_ApplicationControl.h"

(STRS-21) The STRS platform provider shall provide an "STRS_ApplicationControl.h" that contains the method prototypes for each STRS application and, for C++, the class definition for the base class STRS_ApplicationControl.

(STRS-22) If the STRS Application-provided Application Control API is implemented in C++, the STRS application class shall be derived from the STRS_ApplicationControl base class.

For example, the MyWaveform.h file should contain a class definition of the form class MyWaveform: public STRS_ApplicationControl {...};

A sink is used for a push model of passing data, that is, to write data to the waveform, device, file, or queue.

(STRS-23) If the STRS application provides the APP_Write method, the STRS application shall contain
#include "STRS_Sink.h"

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-24) The STRS platform provider shall provide an “STRS_Sink.h” that contains the method prototypes for APP_Write and, for C++, the class definition for the base class STRS_Sink.

(STRS-25) If the STRS Application-provided Application Control API is implemented in C++ and the STRS application provides the APP_Write method, the STRS application class shall be derived from the STRS_Sink base class.

For example, the MyWaveform.h file should contain a class definition of the form

```
class MyWaveform: public STRS_ApplicationControl,  
                  public STRS_Sink  
{...};
```

A source is used for a pull model of passing data: to read data from the waveform, device, file, or queue.

(STRS-26) If the STRS application provides the APP_Read method, the STRS application shall contain

```
#include "STRS_Source.h"
```

(STRS-27) The STRS platform provider shall provide an “STRS_Source.h” that contains the method prototypes for APP_Read and, for C++, the class definition for the base class STRS_Source.

(STRS-110) The STRS platform provider shall provide an “STRS_APIs.h” that contains the method prototypes for the STRS infrastructure APIs.

(STRS-28) If the STRS Application-provided Application Control API is implemented in C++ and the STRS application provides the APP_Read method, the STRS application class shall be derived from the STRS_Source base class.

For example, the MyWaveform.h file should contain a class definition of the form

```
class MyWaveform: public STRS_ApplicationControl,  
                  public STRS_Source  
{...};
```

If both APP_Read and APP_Write are provided in the same waveform, the C++ class will be derived from all three base classes named in requirements (STRS-22, STRS-25, and STRS-28).

For example, the MyWaveform.h file should contain a class definition of the form

```
class MyWaveform: public STRS_ApplicationControl,  
                  public STRS_Sink,  
                  public STRS_Source  
{...};
```

(STRS-111) Each STRS Device shall contain

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

```
#include "STRS_DeviceControl.h"
```

(STRS-112) The STRS platform provider shall provide an “STRS_DeviceControl.h” that contains the method prototypes for each STRS Device and, for C++, the class definition for the base class STRS_DeviceControl, which inherits from the base class STRS_ApplicationControl.

(STRS-113) If the STRS Device-provided Device Control API is implemented in C++, the STRS Device class shall be derived from the STRS_DeviceControl base class.

For example, the MyDevice.h file should contain a class definition of the form

```
class MyDevice: public STRS_DeviceControl  
    [, public STRS_Source]  
    [, public STRS_Sink]
```

```
{...};
```

Note: [] indicates optional.

The following are the STRS Application-provided Application Control APIs:

(STRS-29) Each STRS application shall contain a callable APP_Configure method as described in Table 5, APP_Configure().

Table 5—APP_Configure()

APP_Configure()	
Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states. The API is defined in STRS_PropertySet.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C.• name – (in STRS_Property_Name) name or other identification of data to be obtained• value – (in STRS_Property_Value *) location of data to process/store in application corresponding to name• lenValue – (in STRS_Buffer_Size) actual length of data in value
Return	status (STRS_Result) actual size stored unless error
Precondition	None
Postcondition	The appropriately named value is configured. When an error is returned, see the logs for more information.
Applicable to	Application developer

NASA-STD-4009A

(STRS-114) Each STRS application shall contain a callable APP_Destroy method as described in Table 6, APP_Destroy().

Table 6—APP_Destroy()

APP_Destroy()	
Description	Call the destructor for the specified target component. APP_Destroy is not a class method.
Parameters	<ul style="list-style-type: none">• pApp – (STRS_Instance *) pointer to application instance.
Return	None
Precondition	Application must be stopped and resources released.
Postcondition	STRS instance object is no longer valid.
Applicable to	Application developer

(STRS-115) The STRS infrastructure shall define a callable APP_GetHandleID method in each application as described in Table 7, APP_GetHandleID().

Table 7—APP_GetHandleID()

APP_GetHandleID()	
Description	Obtain the handle ID for the application, stored by the OE.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	handle ID of the current application (STRS_HandleID)
Precondition	Application is instantiated.
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-116) The STRS infrastructure shall define a callable APP_GetHandleName method in each application as described in Table 8, APP_GetHandleName().

Table 8—APP_GetHandleName()

APP_GetHandleName()	
Description	Obtain the handle name for the application, stored by the OE.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	handle name of the current application (char *)
Precondition	Application is instantiated.
Postcondition	None
Applicable to	OE developer: usually platform provider

NASA-STD-4009A

(STRS-30) Each STRS application shall contain a callable APP_GroundTest method as described in Table 9, APP_GroundTest().

Table 9—APP_GroundTest()

APP_GroundTest()	
Description	Perform unit and system testing, which is usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject. The method is similar to APP_RunTest except that it contains more extensive testing used especially before deployment to satisfy any additional project requirements. This method may be invalid upon deployment and if so, it may be eliminated.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.• testID – (in STRS_TestID) number of the test to be performed
Return	status (STRS_Result)
Precondition	None
Postcondition	The test is performed. The state is unchanged unless specifically required by mission.
Applicable to	Application developer

(STRS-31) Each STRS application shall contain a callable APP_Initialize method as described in Table 10, APP_Initialize().

Table 10—APP_Initialize()

APP_Initialize()	
Description	Initialize the target component (application, device). The API is defined in STRS_LifeCycle. The purpose is to set or reset the component to a known initial state. If no fault is detected, this method changes the internal state as appropriate.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	Application developer

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-32) Each STRS application shall contain a callable APP_Instance method as described in Table 11, APP_Instance().

Table 11—APP_Instance()

APP_Instance()	
Description	Store the two parameters passed in the calling sequence, so that they are available to the constructor. In C++, APP_Instance is a static method used to call the class constructor for C++. If no fault is detected, this method returns an instance pointer and initializes the internal state.
Parameters	<ul style="list-style-type: none"> id – (in STRS_HandleID) handle ID of this STRS application. name – (in STRS_HandleName *) handle name of this STRS application.
Return	Pointer to STRS_Instance, instance of class, in C or C++.
Precondition	None
Postcondition	None
Applicable to	Application developer

(STRS-33) Each STRS application shall contain a callable APP_Query method as described in Table 12, APP_Query().

Table 12—APP_Query()

APP_Query()	
Description	Obtain values for one or more properties in the target component (application, device). It is the responsibility of the target component to determine which properties can be interrogated in which internal states. The API is defined in STRS_PropertySet.
Parameters	<ul style="list-style-type: none"> inst – (STRS_Instance *) instance pointer, only for C implementation. name – (in STRS_Property_Name) name or other identification of data to be obtained value – (in STRS_Property_Value *) location to store data corresponding to name lenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value
Return	status (STRS_Result) actual size unless error
Precondition	The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).
Postcondition	Value is populated with data, as appropriate. When an error is returned, see the logs for more information.
Applicable to	Application developer

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-34) If the STRS application provides data to the infrastructure, then the STRS application shall contain a callable APP_Read method as described in Table 13, APP_Read().

Table 13—APP_Read()

APP_Read()	
Description	Method used to obtain data from the target component (application, device). This is optional. The API is defined in STRS_Source. The caller manages the buffer area, preallocating the buffer before calling APP_Read and processing the returned data without any effects on the data source application. The character data type (STRS_Message) does not have to contain valid characters. If the application is not in the appropriate internal state, then nothing is done and an error is returned.
Parameters	<ul style="list-style-type: none"> inst – (STRS_Instance *) instance pointer, only for C implementation. buffer – (out STRS_Message) a pointer to an area in which the application stores the requested data nb – (in STRS_Buffer_Size) number of bytes requested
Return	Error status (negative) or actual number of bytes (non-negative) obtained (STRS_Result)
Precondition	Storage for the buffer with space for nb bytes is allocated before calling APP_Read. If used for a C-style character string, the size should include space for a final '\0'.
Postcondition	The data from the application is stored in the buffer area.
Applicable to	Application developer

(STRS-35) Each STRS application shall contain a callable APP_ReleaseObject method as described in Table 14, APP_ReleaseObject().

Table 14—APP_ReleaseObject()

APP_ReleaseObject()	
Description	Free any resources that the target component (application, device) has acquired. An example would be to allow the target component to close any open files or devices. It is the responsibility of the target component to determine whether any release is done in which internal states. The API is defined in STRS_LifeCycle. The purpose of APP_ReleaseObject is to prepare the target component for removal.
Parameters	<ul style="list-style-type: none"> inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	All resources acquired by the target component are released. The application may not be usable unless instantiated or reinitialized.
Applicable to	Application developer

NASA-STD-4009A

(STRS-36) Each STRS application shall contain a callable APP_RunTest method as described in Table 15, APP_RunTest().

Table 15—APP_RunTest()

APP_RunTest()	
Description	Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the application. Application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.• testID – (in STRS_TestID) number of the test to be performed. Values of testID are mission dependent.
Return	status (STRS_Result)
Precondition	None
Postcondition	The test is performed.
Applicable to	Application developer

(STRS-37) Each STRS application shall contain a callable APP_Start method as described in Table 16, APP_Start().

Table 16—APP_Start()

APP_Start()	
Description	Begin normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent.
Parameters	<ul style="list-style-type: none">• inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	Application developer

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-38) Each STRS application shall contain a callable APP_Stop method as described in Table 17, APP_Stop().

Table 17—APP_Stop()

APP_Stop()	
Description	End normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	Application developer

(STRS-39) If the STRS application receives data from the infrastructure, then the STRS application shall contain a callable APP_Write method as described in Table 18, APP_Write().

Table 18—APP_Write()

APP_Write()	
Description	Method used to send data to the target component (application, device). This is optional. The API is defined in STRS_Sink. The caller manages the buffer area, preallocating and filling the buffer before calling APP_Write. The character data type (STRS_Message) does not have to contain valid characters. If the application is not in the appropriate internal state, then nothing is done and an error is returned.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.buffer – (in STRS_Message) pointer to the data for the application to processnb – (in STRS_Buffer_Size) number of bytes in buffer
Return	Error status (negative) or number of bytes (non-negative) written (STRS_Result)
Precondition	Storage for the buffer with space for nb bytes is allocated before calling APP_Write. If used for a C-style character string, the size should include space for a final '\0'.
Postcondition	The data has been captured by the application for its processing.
Applicable to	Application developer

7.3.2 STRS Infrastructure-Provided Application Control API

The STRS infrastructure provides the STRS Infrastructure-provided Application Control API to support application operation using the STRS Application-provided Application Control API in section 7.3.1. These STRS Infrastructure-provided Application Control API methods (section 7.3.2 beginning with “STRS_” correspond to the STRS Application-provided Application Control API (section 7.3.1) beginning with “APP_”, and are used to access those STRS

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Application-provided Application Control API methods. The STRS infrastructure implements these STRS Infrastructure-provided Application Control API methods for use by any STRS application, or any part of the infrastructure that is desired to be implemented in a portable way.

A property structure contains a list of the name and value pairs used to set or get execution parameters (section 7.3.10).

(STRS-40) The STRS infrastructure shall contain a callable STRS_Configure method as described in Table 19, STRS_Configure().

Table 19—STRS_Configure()

STRS_Configure()	
Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.• name – (in STRS_Property_Name) name or other identification of data to be obtained• value – (in STRS_Property_Value *) location of data to process/store in application corresponding to name• lenValue – (in STRS_Buffer_Size) actual length of data in value
Return	status (STRS_Result) actual size stored unless error
Precondition	None
Postcondition	The appropriate named value is configured. When an error is returned, see the logs for more information.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-41) The STRS infrastructure shall contain a callable STRS_GroundTest method as described in Table 20, STRS_GroundTest().

Table 20—STRS_GroundTest()

STRS_GroundTest()	
Description	Perform unit and system testing—usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. This method provides more exhaustive testing to satisfy any additional project requirements. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. This method may be invalid upon deployment.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.• testID – (in STRS_TestID) number of the test to be performed. Values are mission dependent.
Return	status (STRS_Result)
Precondition	None
Postcondition	The test is performed.
Applicable to	OE developer: usually platform provider

(STRS-42) The STRS infrastructure shall contain a callable STRS_Initialize method as described in Table 21, STRS_Initialize().

Table 21—STRS_Initialize()

STRS_Initialize()	
Description	Initialize the target component (application, device). The purpose is to set or reset the component to a known initial state.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-43) The STRS infrastructure shall contain a callable STRS_Query method as described in Table 22, STRS_Query().

Table 22—STRS_Query()

STRS_Query()	
Description	Obtain values for one or more properties in the target component (application, device).
Parameters	<ul style="list-style-type: none"> • fromWF – (in STRS_HandleID) handle ID of current component making the request. • toWF – (in STRS_HandleID) handle ID of target component that should respond to the request. • name – (in STRS_Property_Name) name or other identification of data to be obtained • value – (in STRS_Property_Value *) location to store data corresponding to name • lenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value
Return	status (STRS_Result) actual size unless error
Precondition	The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).
Postcondition	Value is populated with data as appropriate. When an error is returned, see the logs for more information.
Applicable to	OE developer: usually platform provider

(STRS-44) The STRS infrastructure shall contain a callable STRS_ReleaseObject method as described in Table 23, STRS_ReleaseObject().

Table 23—STRS_ReleaseObject()

STRS_ReleaseObject()	
Description	Free any resources that the target component (application, device) has acquired. An example would be to allow the target component to close any open files or devices. It is the responsibility of the target component to determine whether any release is done in which internal states. The purpose of STRS_ReleaseObject is to prepare the target component for removal.
Parameters	<ul style="list-style-type: none"> • fromWF – (in STRS_HandleID) handle ID of current component making the request. • toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	All resources acquired by the target component are released. The target component may not be usable unless reinstantiated or reinitialized.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-45) The STRS infrastructure shall contain a callable STRS_RunTest method as described in Table 24, STRS_RunTest().

Table 24—STRS_RunTest()

STRS_RunTest()	
Description	Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the target component. A responding application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.• testID – (in STRS_TestID) number of the test to be performed. Values of testID are mission-dependent.
Return	status (STRS_Result)
Precondition	None
Postcondition	The test is performed.
Applicable to	OE developer: usually platform provider

(STRS-46) The STRS infrastructure shall contain a callable STRS_Start method as described in Table 25, STRS_Start().

Table 25—STRS_Start()

STRS_Start()	
Description	Begin normal target component (application, device) processing. Nothing is done if the application (or device) is not in the appropriate internal state.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-47) The STRS infrastructure shall contain a callable STRS_Stop method as described in Table 26, STRS_Stop().

Table 26—STRS_Stop()

STRS_Stop()	
Description	End target component (application, device) processing. Nothing is done unless the application (or device) is in the appropriate internal state.
Parameters	<ul style="list-style-type: none">• fromWF – in STRS_HandleID) handle ID of current component making the request.• toWF – in STRS_HandleID) handle ID of target component that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

7.3.3 STRS Infrastructure Application Setup API

The STRS infrastructure Application Setup methods are general methods or are used to control one application from another.

(STRS-48) The STRS infrastructure shall contain a callable STRS_AbortApp method as described in Table 27, STRS_AbortApp().

Table 27—STRS_AbortApp()

STRS_AbortApp()	
Description	Abort an application or service.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toWF – (in STRS_HandleID) handle ID of target component that should respond to the request
Return	Status (STRS_Result)
Precondition	None
Postcondition	The target component is aborted, and application is stopped, resources released, and unloaded, if allowed by OE.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-49) The STRS infrastructure shall contain a callable STRS_GetErrorQueue method as described in Table 28, STRS_GetErrorQueue().

Table 28—STRS_GetErrorQueue()

STRS_GetErrorQueue()	
Description	Transform an error status into an error queue.
Parameters	<ul style="list-style-type: none">result – (in STRS_Result) return value of previous call.
Return	Handle ID (STRS_HandleID) corresponding to invalid STRS_Result; that is, return STRS_ERROR_QUEUE for STRS_ERROR, STRS_WARNING_QUEUE for STRS_WARNING, and STRS_FATAL_QUEUE for STRS_FATAL; otherwise, implementation defined.
Precondition	None
Postcondition	The corresponding error queue handle ID is returned.
Applicable to	OE developer: usually platform provider

(STRS-117) The STRS infrastructure shall contain a callable STRS_GetHandleName method as described in Table 29, STRS_GetHandleName().

Table 29—STRS_GetHandleName()

STRS_GetHandleName()	
Description	The handle name is obtained for the given handle ID. The handle ID of the current component (fromWF) is used for any error message. Using STRS_GetHandleName to determine the handle name of the current component while it is being instantiated gives undefined results.
Parameters	<ul style="list-style-type: none">fromWF – (in STRS_HandleID) handle ID of current component making the request.toID – (in STRS_HandleID) handle ID of the resource (application, device, file, queue) for which the handle name is to be obtained.toResourceName – (out char *) handle name of the desired resource.
Return	status (STRS_Result)
Precondition	Space must be allocated for handle name.
Postcondition	Handle name is filled in. On error, the first character of the handle name is filled with a zero unless the toResourceName variable is NULL.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-50) The STRS infrastructure shall contain a callable STRS_HandleRequest method as described in Table 30, STRS_HandleRequest().

Table 30—STRS_HandleRequest()

STRS_HandleRequest()	
Description	The handle ID is obtained for the given handle name. The handle ID of the current component (fromWF) is used for any error message. Using STRS_HandleRequest to determine the handle ID of the current component while it is being instantiated gives undefined results.
Parameters	<ul style="list-style-type: none"> fromWF – (in STRS_HandleID) handle ID of current component making the request. toResourceName – (in char *) name of desired resource (application, device, file, queue).
Return	Handle ID of the entity or error status. (STRS_HandleID)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-51) The STRS infrastructure shall contain a callable STRS_InstantiateApp method as described in Table 31, STRS_InstantiateApp().

Table 31—STRS_InstantiateApp()

STRS_InstantiateApp()	
Description	Instantiate an application, service, or device. The handle name specified for the application, service, or device is to be unique. The OE-specific name is used to identify the application for instantiation and may impose additional operations to be performed as documented by the platform provider. It is up to the OE to determine whether any resources are to be loaded to accomplish the instantiation.
Parameters	<ul style="list-style-type: none"> fromWF – (in STRS_HandleID) handle ID of current component making the request. handleName – (in char *) unique handle name for the application (or device) that should be instantiated. startName – (in char *) OE-specific name used to instantiate and configure the application (or device) into a known state.
Return	handle ID (STRS_HandleID) of the application (or device) instantiated or the error status
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-52) The STRS infrastructure shall contain a callable STRS_IsOK method as described in Table 32, STRS_IsOK().

Table 32—STRS_IsOK()

STRS_IsOK()	
Description	Return true, if return value of argument obtained from previous call is not an error status.
Parameters	<ul style="list-style-type: none">• result – (in STRS_Result) return value of previous call.
Return	true, if STRS_Result is not STRS_WARNING, STRS_ERROR, or STRS_FATAL: that is, non-negative (bool)
Precondition	Previous call returns a status result.
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-53) The STRS infrastructure shall contain a callable STRS_Log method as described in Table 33, STRS_Log().

(STRS-54) When an STRS application has a nonfatal error, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_ERROR_QUEUE.

(STRS-55) When an STRS application has a fatal error, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_FATAL_QUEUE.

(STRS-56) When an STRS application has a warning condition, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_WARNING_QUEUE.

(STRS-57) When an STRS application needs to send telemetry, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_TELEMETRY_QUEUE.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Table 33—STRS_Log()

STRS_Log()	
Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.
Parameters	<ul style="list-style-type: none"> fromWF – (in STRS_HandleID) handle ID of current component making the request. logTarget – (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors. msg – (in STRS_Message) a pointer to the data to process nb – (in STRS_Buffer_Size) number of bytes in buffer
Return	status (STRS_Result)
Precondition	None
Postcondition	Log message is distributed.
Applicable to	OE developer: usually platform provider

(STRS-118) The STRS infrastructure shall contain a callable STRS_ValidateHandleID method as described in Table 34, STRS_ValidateHandleID().

Table 34—STRS_ValidateHandleID()

STRS_ValidateHandleID()	
Description	Determines if a handle ID is STRS_OK or in error. After calling any STRS method that returns a handle ID, it is recommended that STRS_ValidateHandleID be called before any other STRS method.
Parameters	<ul style="list-style-type: none"> tstID – (in STRS_HandleID) the STRS_HandleID object from which the handle ID is extracted.
Return	(STRS_Result) STRS_OK when successful; otherwise, for error, STRS_WARNING, STRS_ERROR, or STRS_FATAL.
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

NASA-STD-4009A

(STRS-119) The STRS infrastructure shall contain a callable STRS_ValidateSize method as described in Table 35, STRS_ValidateSize().

Table 35—STRS_ValidateSize()

STRS_ValidateSize()	
Description	Determines if an STRS_File_Size is STRS_OK or in error. STRS_FileGetFreeSpace and STRS_FileGetSize return a type STRS_File_Size number. After calling any STRS method that returns an STRS_File_Size, it is recommended that STRS_ValidateSize be called before calling any other STRS method.
Parameters	<ul style="list-style-type: none"> tstSize – (in STRS_File_Size) the file size object from which the file size is extracted.
Return	(STRS_Result) STRS_OK when successful; otherwise, for error, STRS_WARNING, STRS_ERROR, or STRS_FATAL.
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

7.3.4 STRS Infrastructure Data Sink

The STRS Infrastructure Data Sink method, STRS_Write, is used to push data to any implemented data sink, such as an STRS application or STRS Device implementing APP_Write, a queue, a Pub/Sub, a file opened for writing, etc.

(STRS-58) The STRS infrastructure shall contain a callable STRS_Write method as described in Table 36, STRS_Write().

Table 36—STRS_Write()

STRS_Write()	
Description	Method used to send data to a target component (application, device, file, or queue) acting as a sink. The caller manages the buffer area, preallocating and filling the buffer before calling STRS_Write. The character data type (STRS_Message) does not have to contain valid characters.
Parameters	<ul style="list-style-type: none"> fromWF – (in STRS_HandleID) handle ID of current component making the request. toID – (in STRS_HandleID) handle ID of target component that should respond to the request and that implemented STRS_Sink. buffer – (in STRS_Message) a pointer to the data to process nb – (in STRS_Buffer_Size) number of bytes in buffer
Return	Error status (negative) or number of bytes (non-negative) written (STRS_Result)
Precondition	Storage for the buffer is allocated before calling STRS_Write having space for at least nb bytes. If used for a C-style character string, the size should include space for a final '\0'.
Postcondition	The data has been captured by the target component for its processing.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

7.3.5 STRS Infrastructure Data Source

The STRS Infrastructure Data Source method, STRS_Read, is used to pull data from any implemented data source or supplier. A data source may be an STRS application or STRS Device implementing APP_Read, a queue, or a file opened for reading.

(STRS-59) The STRS infrastructure shall contain a callable STRS_Read method as described in Table 37, STRS_Read().

Table 37—STRS_Read()

STRS_Read()	
Description	Method used to obtain data from a target component (application, device, file, or queue) acting as a source or supplier. The caller manages the buffer area, preallocating the buffer before calling STRS_Read and processing the returned data without any effects on the data source application. The character data type (STRS_Message) does not have to contain valid characters.
Parameters	<ul style="list-style-type: none"> • fromWF – (in STRS_HandleID) handle ID of current component making the request. • pullID – (in STRS_HandleID) handle ID of target component that should respond to the request and that implemented STRS_Source. • buffer – (out STRS_Message) a pointer to an area in which to store the data requested • nb – (in STRS_Buffer_Size) number of bytes requested
Return	Error status (negative) or actual number of bytes (non-negative) obtained (STRS_Result)
Precondition	Storage for the buffer is allocated before calling STRS_Read, having space for at least nb bytes. If used for a C-style character string, the size should include space for a final '\0'.
Postcondition	The data from the target component is stored in the buffer area.
Applicable to	OE developer: usually platform provider

7.3.6 STRS Infrastructure-Provided Device Control API

An STRS Device is a proxy for the data and/or control path to the actual hardware. An STRS Device is a “bridge” used to “decouple an abstraction from its implementation so that the two can vary independently.” An STRS Device is called using the methods in the STRS Infrastructure-Provided Device Control API (as described in the tables below), STRS Infrastructure-Provided Application Control API (section 7.3.2), Infrastructure Data Source API (section 7.3.5, if appropriate), and Infrastructure Data Sink API (section 7.3.4, if appropriate) to control the STRS Devices. The STRS Device may be implemented using any available platform-specific HAL to communicate with and control the specialized hardware. An STRS Device may also be used to hide the details of networking from the application. The purpose of abstracting the hardware interfaces in a standard manner is to make the applications more portable and reusable.

NASA-STD-4009A

(STRS-61) The STRS infrastructure shall contain a callable STRS_DeviceClose method as described in Table 38, STRS_DeviceClose().

Table 38—STRS_DeviceClose()

STRS_DeviceClose()	
Description	Close the open device.
Parameters	<ul style="list-style-type: none">fromWF – (in STRS_HandleID) handle ID of current component making the request.toDev – (in STRS_HandleID) handle ID of device that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is closed.
Applicable to	OE developer: usually platform provider

(STRS-62) The STRS infrastructure shall contain a callable STRS_DeviceFlush method as described in Table 39, STRS_DeviceFlush().

Table 39—STRS_DeviceFlush()

STRS_DeviceFlush()	
Description	Used the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.
Parameters	<ul style="list-style-type: none">fromWF – (in STRS_HandleID) handle ID of current component making the request.toDev – (in STRS_HandleID) handle ID of device that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-63) The STRS infrastructure shall contain a callable STRS_DeviceLoad method as described in Table 40, STRS_DeviceLoad().

Table 40—STRS_DeviceLoad()

STRS_DeviceLoad()	
Description	Load a binary image to the open device.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toDev – (in STRS_HandleID) handle ID of device that should respond to the request.• filename – (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.
Return	status (STRS_Result)
Precondition	None
Postcondition	The binary image is stored in the target device.
Applicable to	OE developer: usually platform provider

(STRS-64) The STRS infrastructure shall contain a callable STRS_DeviceOpen method as described in Table 41, STRS_DeviceOpen().

Table 41—STRS_DeviceOpen()

STRS_DeviceOpen()	
Description	Open the device.
Parameters	<ul style="list-style-type: none">• fromWF – (in STRS_HandleID) handle ID of current component making the request.• toDev – (in STRS_HandleID) handle ID of device that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is opened.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-65) The STRS infrastructure shall contain a callable STRS_DeviceReset method as described in Table 42, STRS_DeviceReset().

Table 42—STRS_DeviceReset()

STRS_DeviceReset()	
Description	Reinitialize the device, if possible. Reset is normally used after the corresponding device has been started and stopped, and before the device is started again to bring the hardware device to its power-on state.
Parameters	<ul style="list-style-type: none">fromWF – (in STRS_HandleID) handle ID of current component making the request.toDev – (in STRS_HandleID) handle ID of device that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-68) The STRS infrastructure shall contain a callable STRS_DeviceUnload method as described in Table 43, STRS_DeviceUnload().

Table 43—STRS_DeviceUnload()

STRS_DeviceUnload()	
Description	Unload the open device.
Parameters	<ul style="list-style-type: none">fromWF – (in STRS_HandleID) handle ID of current component making the request.toDev – (in STRS_HandleID) handle ID of device that should respond to the request.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is unloaded.
Applicable to	OE developer: usually platform provider

(STRS-69) The STRS infrastructure shall contain a callable STRS_SetISR method as described in Table 44, STRS_SetISR().

Table 44—STRS_SetISR()

STRS_SetISR()	
Description	Set the Interrupt Service Routine for the device.
Parameters	<ul style="list-style-type: none"> fromWF – (in STRS_HandleID) handle ID of the current component making the request. toDev – (in STRS_HandleID) handle ID of the device that should respond to the request. pfun – (in STRS_ISR_Function) function pointer to a static or non-class function to be called to service the interrupt
Return	status (STRS_Result)
Precondition	None
Postcondition	ISR function is activated.
Applicable to	OE developer: usually platform provider

7.3.7 STRS Device-Provided Device Control API

An STRS Device is a proxy for the data and/or control path to the actual hardware. An STRS Device is a “bridge” used to “decouple an abstraction from its implementation so that the two can vary independently.” An STRS Device is called using the methods in the STRS Infrastructure-Provided Device Control API (section 7.3.6), STRS Infrastructure-Provided Application Control API (section 7.3.2), Infrastructure Data Source API (section 7.3.5, if appropriate), and Infrastructure Data Sink API (section 7.3.4, if appropriate) to control the STRS Devices. The STRS Device may be implemented using any available platform-specific HAL to communicate with and control the specialized hardware. An STRS Device may also be used to hide the details of networking from the application. The purpose of abstracting the hardware interfaces in a standard manner is to make the applications more portable and reusable. A portable STRS Device is an STRS application that implements the STRS Application-Provided Application Control API (section 7.3.1) calls and the STRS Device-Provided Device Control calls shown below. The STRS Device implementation with the STRS_DeviceControl interface is shown in Figure 14.

NASA-STD-4009A

(STRS-120) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Close method as described in Table 45, DEV_Close().

Table 45—DEV_Close()

DEV_Close()	
Description	Close the open device.
Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is closed.
Applicable to	Device developer: usually platform provider

(STRS-121) If the hardware is to be flushed by the STRS Device, the STRS Device shall contain a callable DEV_Flush method as described in Table 46, DEV_Flush().

Table 46—DEV_Flush()

DEV_Flush()	
Description	Use the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.
Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device's buffered data is flushed.
Applicable to	Device developer: usually platform provider

(STRS-122) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Load method as described in Table 47, DEV_Load().

Table 47—DEV_Load()

DEV_Load()	
Description	Load a binary image to the open device.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.fileName - (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.
Return	status (STRS_Result)
Precondition	None
Postcondition	The binary image is stored in the target device.
Applicable to	Device developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-123) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Open method as described in Table 48, DEV_Open().

Table 48—DEV_Open()

DEV_Open()	
Description	Open the device.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is opened.
Applicable to	Device developer: usually platform provider

(STRS-124) If the hardware is to be reset by the STRS Device, the STRS Device shall contain a callable DEV_Reset method as described in Table 49, DEV_Reset().

Table 49—DEV_Reset()

DEV_Reset()	
Description	Reinitialize the device, if possible. Reset is normally used after the corresponding device has been started and stopped, and before the device is started again to bring the hardware device to its power-on state.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is reset to an initial state.
Applicable to	Device developer: usually platform provider

(STRS-125) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Unload method as described in Table 50, DEV_Unload().

Table 50—DEV_Unload()

DEV_Unload()	
Description	Unload the open device.
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.
Return	status (STRS_Result)
Precondition	None
Postcondition	The device is unloaded.
Applicable to	Device developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

7.3.8 STRS Infrastructure File Control API

The STRS Infrastructure File Control methods, along with STRS_Read and/or STRS_Write, provide a portable means for the applications to use storage, the duration of which is mission-dependent. The word “file” is used to mean a named storage area regardless of the existence of a file system. The file control methods in POSIX® PSE51 are not sufficient for the needs of STRS because an application strictly conforming to PSE51 can use the open(), fopen(), or freopen() functions only to open existing files, not to create new files. In addition, the PSE51 profile lacks functions to remove files or to provide information regarding available storage. For more information about POSIX®, see section 7.4. The STRS Infrastructure File Control methods use a handle ID to access storage.

(STRS-70) The STRS infrastructure shall contain a callable STRS_FileClose method as described in Table 51, STRS_FileClose().

Table 51—STRS_FileClose()

STRS_FileClose()	
Description	Close the open file. STRS_FileClose is used to close a file that has been opened by STRS_FileOpen.
Parameters	<ul style="list-style-type: none"> fromWF - (in STRS_HandleID) handle ID of current component making the request. toFile - (in STRS_HandleID) handle ID of file to be closed.
Return	status (STRS_Result)
Precondition	None
Postcondition	The file is closed and the handle ID is released.
Applicable to	OE developer: usually platform provider

(STRS-71) The STRS infrastructure shall contain a callable STRS_FileGetFreeSpace method as described in Table 52, STRS_FileGetFreeSpace().

Table 52—STRS_FileGetFreeSpace()

STRS_FileGetFreeSpace()	
Description	Get total size of free space available for file storage.
Parameters	<ul style="list-style-type: none"> fromWF - (in STRS_HandleID) handle ID of current component making the request. fileSystem - (in char *) used when more than one file system exists.
Return	Total size in bytes (STRS_File_Size).
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

NASA-STD-4009A

(STRS-72) The STRS infrastructure shall contain a callable STRS_FileGetSize method as described in Table 53, STRS_FileGetSize().

Table 53—STRS_FileGetSize()

STRS_FileGetSize()	
Description	Get the size of the specified file.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.fileName - (in char *) storage area name or fully qualified file name of the file for which the size is obtained.
Return	File size in bytes (STRS_File_Size).
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-73) The STRS infrastructure shall contain a callable STRS_FileGetStreamPointer method as described in Table 54, STRS_FileGetStreamPointer().

Table 54—STRS_FileGetStreamPointer()

STRS_FileGetStreamPointer()	
Description	Get the file stream pointer for the open file associated with the STRS handle ID. This is normally not used because either the common functions are built into the STRS architecture or the entire file manipulation is local to one application or device. This method may be needed for certain file operations not built into the STRS architecture and distributed over more than one application or device or the STRS infrastructure. For example, the file stream pointer may be used when multiple applications write to the same file using a queue or need features not found in STRS_Write. Having a file system is optional; if no file system is present, NULL will be returned. A NULL will also be returned if another error condition is detected.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toFile - (in STRS_HandleID) file handle ID.
Return	File stream pointer (FILE *) or NULL for error condition.
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-74) The STRS infrastructure shall contain a callable STRS_FileOpen method as described in Table 55, STRS_FileOpen().

Table 55—STRS_FileOpen()

STRS_FileOpen()	
Description	Open the file. This method is used to obtain an STRS handle ID when the file manipulation is either built into the STRS architecture or distributed over more than one application or device or the STRS infrastructure.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• filename - (in char *) file name of the file to be opened.• file access - (in STRS_Access) indicates if file is to be opened for reading, writing, both, or appending.• file type - (in STRS_Type) indicator whether file is text or binary.
Return	a handle ID used to read or write data from or to the file (STRS_HandleID). Handle ID should be validated with STRS_ValidateHandleID to determine if successful.
Precondition	None
Postcondition	The file is open unless an error occurs. On error, the return value should contain an error indication that can be tested by STRS_ValidateHandleID.
Applicable to	OE developer: usually platform provider

(STRS-75) The STRS infrastructure shall contain a callable STRS_FileRemove method as described in Table 56, STRS_FileRemove().

Table 56—STRS_FileRemove()

STRS_FileRemove()	
Description	Remove the closed file.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• oldName - (in char *) name of file to be removed.
Return	status (STRS_Result)
Precondition	None
Postcondition	The file is no longer available, and the space where it was stored becomes available.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-76) The STRS infrastructure shall contain a callable STRS_FileRename method as described in Table 57, STRS_FileRename().

Table 57—STRS_FileRename()

STRS_FileRename()	
Description	Rename the closed file where the new file name does not exist prior to the call.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) current name of file.newName - (in char *) new name of file after rename.
Return	status (STRS_Result)
Precondition	None
Postcondition	The contents of the old file are now associated with the new file name.
Applicable to	OE developer: usually platform provider

7.3.9 STRS Infrastructure Messaging API

The STRS applications use the STRS Infrastructure Messaging methods to establish messages passing facilities to send messages between components using a single handle ID. The ability for applications, services, devices, or files to communicate with other STRS applications, services, devices, or files is crucial for the separation of radio functionality among independent asynchronous components. For example, the receive and transmit telecommunication functionalities can be separated between two applications. Another example is when commands or log messages come from several independent sources and have to be merged appropriately. Some examples of independent components that probably need to interact with others could be for navigation, GPS, file upload, file download, and computations (even nonradio). The STRS radio is essentially a computer, and it has capabilities that make the whole spacecraft system more robust. The final destination of a message is not necessarily known to the producer of the message.

There are two models for passing messages: STRS queue and Pub/Sub. In an STRS queue, messages are written to a First-In First-Out (FIFO) queue by one entity and read from the FIFO queue by another entity. In a Pub/Sub, messages written to the message passing facility by one application are delivered to all subscribers of that publisher. Therefore, the Pub/Sub messaging API should be implemented using a form of the Publish-Subscribe design pattern. To read from a queue, STRS_Read is used. To write to a message passing facility, STRS_Write is used. STRS_Read and STRS_Write provide a portable means for the applications to use message passing facility.

Specific predefined message passing facility handle identification/identifiers (IDs) denoted by STRS_ERROR_QUEUE, STRS_FATAL_QUEUE, and STRS_WARNING_QUEUE are required. The STRS_Log method uses these special-purpose handle IDs to log errors.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

The message passing facility names are global so that the ones with the same name refer to the same target across all applications. The same handle name refers to the same application, device, file, queue, Pub/Sub, timer, or service across all applications. For information about errors, see section 7.3.12.

(STRS-77) The STRS applications shall use the STRS Infrastructure Messaging, STRS Infrastructure Data Source, and STRS Infrastructure Data Sink methods to send messages between components.

(STRS-126) The STRS infrastructure shall contain a callable STRS_MessageQueueCreate method as described in Table 58, STRS_MessageQueueCreate().

Table 58—STRS_MessageQueueCreate()

STRS_MessageQueueCreate()	
Description	Create a FIFO message queue if a handle does not already exist having the given name
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• queueName - (in char *) unique name of the queue.• nb – (STRS_Buffer_Size) maximum size of buffer containing messages.• nmax – (STRS_Queue_Max_Messages) maximum number of messages in queue.
Return	handle ID of queue or error status (STRS_HandleID)
Precondition	None
Postcondition	Queue is created unless an error occurs.
Applicable to	OE developer: usually platform provider

(STRS-127) The STRS infrastructure shall contain a callable STRS_MessageQueueDelete method as described in Table 59, STRS_MessageQueueDelete().

Table 59—STRS_MessageQueueDelete()

STRS_MessageQueueDelete()	
Description	Delete a queue if it exists.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toQueue - (inout STRS_HandleID) handle ID of queue to delete.
Return	status (STRS_Result)
Precondition	None
Postcondition	Queue is deleted.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-128) The STRS infrastructure shall contain a callable STRS_PubSubCreate method as described in Table 60, STRS_PubSubCreate().

Table 60—STRS_PubSubCreate()

STRS_PubSubCreate()	
Description	Create a Pub/Sub handle ID that is a proxy used to receive and redistribute messages using STRS_Write unless the handle name already is used somewhere else.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.pubsubName - (in char *) unique name of the Pub/Sub.
Return	handle ID of Pub/Sub or error status (STRS_HandleID)
Precondition	None
Postcondition	Pub/Sub is created unless an error occurs.
Applicable to	OE developer: usually platform provider

(STRS-129) The STRS infrastructure shall contain a callable STRS_PubSubDelete method as described in Table 61, STRS_PubSubDelete().

Table 61—STRS_PubSubDelete()

STRS_PubSubDelete()	
Description	Delete a Pub/Sub if it exists. Any association between a publisher and subscriber that references the Pub/Sub is removed.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toPubSub - (inout STRS_HandleID) handle ID of Pub/Sub to delete.
Return	status (STRS_Result)
Precondition	None
Postcondition	Specified Pub/Sub is deleted and any associations are removed.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-80) The STRS infrastructure shall contain a callable STRS_Register method as described in Table 62, STRS_Register().

Table 62—STRS_Register()

STRS_Register()	
Description	Register an association between a publisher and subscriber where both exist. Disallow duplicates between the same publisher and subscriber.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• useQID - (in STRS_HandleID) handle ID of Pub/Sub that will be used as a sink; the publisher.• actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should respond to the request as a sink; the subscriber.
Return	status (STRS_Result)
Precondition	None
Postcondition	Association between publisher and subscriber is registered, if allowed.
Applicable to	OE developer: usually platform provider

(STRS-81) The STRS infrastructure shall contain a callable STRS_Unregister method as described in Table 63, STRS_Unregister().

Table 63—STRS_Unregister()

STRS_Unregister()	
Description	Remove an association between a publisher and subscriber, if the association exists.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• useQID - (in STRS_HandleID) handle ID of Pub/Sub that was used as a sink; the publisher.• actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should no longer respond to the request as a sink; usually the subscriber.
Return	status (STRS_Result)
Precondition	None
Postcondition	Association between publisher and subscriber is removed.
Applicable to	OE developer: usually platform provider

7.3.10 STRS Infrastructure Time Control API

The STRS Infrastructure Time Control methods are used to access the hardware and software timers. If timers require synchronization with external clocks, a dedicated service should handle the communication between the STRS radio and the external clock source, adjusting the time or offset for distance and velocity, before using these methods to adjust a corresponding internal

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

timer. These methods also include conversion of time between seconds and nanoseconds, taken individually, and some implementation-specific object containing both. Although nanoseconds are the units obtained by STRS_GetNanoseconds, that does not imply that the resolution is nanoseconds or that the underlying STRS_TimeWarp object contains its data in nanoseconds. For example, the underlying STRS_TimeWarp object could count ticks from some epoch and then STRS_GetSeconds and STRS_GetNanoseconds compute the seconds and nanoseconds from the same or a different epoch. These timers are expected to be used for relatively low accuracy timing such as time stamps, timed events, and time constraints. The timers are expected to be used for signal processing in the GPP if the GPP becomes fast enough.

(STRS-82) Any portion of the STRS Applications on the GPP needing time control shall use the STRS Infrastructure Time Control methods to access the hardware and software timers.

(STRS-130) The implementer of an STRS clock/timer software component for use with STRS_GetTime shall document it to include handle name, kind, epoch, resolution, use of leap seconds, and whether it should match a time somewhere else, as described further in Table 64, Document STRS Clock/Timer.

Table 64—Document STRS Clock/Timer

Applicable to	STRS clock/timer developer, which may be platform provider or application developer.
----------------------	--

(STRS-83) The STRS infrastructure shall contain a callable STRS_GetNanoseconds method as described in Table 65, STRS_GetNanoseconds().

Table 65—STRS_GetNanoseconds()

STRS_GetNanoseconds()	
Description	Get the number of nanoseconds from the STRS_TimeWarp object.
Parameters	<ul style="list-style-type: none">twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the nanoseconds portion of the time increment is extracted.
Return	Integer number of nanoseconds in the STRS_TimeWarp object representing a time interval. (STRS_Nanoseconds)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-84) The STRS infrastructure shall contain a callable STRS_GetSeconds method as described in Table 66, STRS_GetSeconds().

Table 66—STRS_GetSeconds()

STRS_GetSeconds()	
Description	Get the number of seconds from the STRS_TimeWarp object.
Parameters	<ul style="list-style-type: none">twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the nanoseconds portion of the time increment is extracted.
Return	integer number of seconds in the STRS_TimeWarp object representing a time interval. (STRS_Seconds)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-85) The STRS infrastructure shall contain a callable STRS_GetTime method as described in Table 67, STRS_GetTime().

Table 67—STRS_GetTime()

STRS_GetTime()	
Description	Get the current base time and the corresponding time of a specified type (kind). The base clock/timer is usually a hardware timer. The variable kind is used to obtain a nonbase time at a specified offset from the base time. An offset is usually specified to ensure that the clock is monotonically increasing after a power reset or synchronized with another clock/timer. To compute the time interval between two nonbase times of different kinds, the function is called twice and the interval is modified by the difference between the two base times.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.baseTime - (inout STRS_TimeWarp) current time of the base timer.kind - (in STRS_Clock_Kind) type of clock/timer.kindTime - (inout STRS_TimeWarp) current time of the specified timer.
Return	status (STRS_Result)
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-131) The STRS infrastructure shall contain a callable STRS_GetTimeAdjust method as described in Table 68, STRS_GetTimeAdjust().

Table 68—STRS_GetTimeAdjust()

STRS_GetTimeAdjust()	
Description	Get the current time rate for the specified clock/timer which, when applied to the clock specified by <i>its handle ID</i> , will more closely synchronize it with another.
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.
Return	iRate (STRS_TimeRate) an integer time rate. Units are specific to the clock/timer.
Precondition	None
Postcondition	Time rate is obtained or computed.
Applicable to	OE developer: usually platform provider

(STRS-86) The STRS infrastructure shall contain a callable STRS_GetTimeWarp method as described in Table 69, STRS_GetTimeWarp().

Table 69—STRS_GetTimeWarp()

STRS_GetTimeWarp()	
Description	Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.
Parameters	<ul style="list-style-type: none">isec - (in STRS_Seconds) number of seconds in the time intervalnsec - (in STRS_Nanoseconds) number of nanoseconds in the fractional portion of the time interval
Return	STRS_TimeWarp object representing the time interval.
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-87) The STRS infrastructure shall contain a callable STRS_SetTime method as described in Table 70, STRS_SetTime().

Table 70—STRS_SetTime()

STRS_SetTime()	
Description	Set the current time in the specified clock/timer by adjusting the time offset.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toDev - (in STRS_HandleID) handle ID of device that should respond to the request.• kind - (in STRS_Clock_Kind) type of clock/timer.• delta - (in STRS_TimeWarp) increment to add to specified clock/timer.
Return	status (STRS_Result)
Precondition	None
Postcondition	Time is adjusted.
Applicable to	OE developer: usually platform provider

(STRS-132) The STRS infrastructure shall contain a callable STRS_SetTimeAdjust method as described in Table 71, STRS_SetTimeAdjust().

Table 71—STRS_SetTimeAdjust()

STRS_SetTimeAdjust()	
Description	Set the current time rate in the specified clock/timer.
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toDev - (in STRS_HandleID) handle ID of device that should respond to the request.• iRate - (in STRS_TimeRate) a rate applied to the specified clock/timer to set the clock/timer relative time. Units are specific to the clock/timer.
Return	status (STRS_Result)
Precondition	None
Postcondition	Time rate is adjusted.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

(STRS-133) The STRS infrastructure shall contain a callable STRS_Sleep method as described in Table 72, STRS_Sleep().

Table 72—STRS_Sleep()

STRS_Sleep()	
Description	Delays the execution of the application for at least the time specified in the STRS_TimeWarp argument that contains the number of seconds and nanoseconds in the time interval. The time interval may still not be accurate depending on the underlying timer resolution and thread interaction.
Parameters	<ul style="list-style-type: none"> clockID – (STRS_HandleID) the handle ID of the timer/clock. twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the time is extracted. absOrRel- (Boolean) true, if absolute time is specified; false, if relative time is specified.
Return	(STRS_Result) STRS_OK when successful. STRS_ERROR for error. STRS_WARNING if interrupted.
Precondition	None
Postcondition	None
Applicable to	OE developer: usually platform provider

(STRS-88) The STRS infrastructure shall contain a callable STRS_TimeSynch method as described in Table 73, STRS_TimeSynch().

Table 73—STRS_TimeSynch()

STRS_TimeSynch()	
Description	Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external, or whether the clocks differ by amounts that exceed the maximum step size allowed.
Parameters	<ul style="list-style-type: none"> fromWF - (in STRS_HandleID) handle ID of current component making the request. refDev - (in STRS_HandleID) handle ID of reference device containing the reference clock/timer. ref - (in STRS_Clock_Kind) type of reference clock/timer. targetDev - (in STRS_HandleID) handle ID of target device to synchronize. target - (in STRS_Clock_Kind) type of clock/timer to synchronize with reference clock/timer. stepMax – (in STRS_TimeWarp) maximum step size to allow at a time, which can be used for gradual time adjustment. Zero implies no limit in step size.
Return	status (STRS_Result) where a positive value indicates the number of steps left to adjust at the maximum step size.
Precondition	None
Postcondition	Clocks are more synchronized.
Applicable to	OE developer: usually platform provider

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

7.3.11 STRS Predefined Data

For portability, standard names are defined for various constants and data types, but the implementation of these definitions is mission dependent. The common symbols and data types defined to support the STRS infrastructure APIs are shown in Table 74, STRS Predefined Data.

(STRS-89) The STRS platform provider shall provide an STRS.h file containing the STRS predefined data shown in Table 74, STRS Predefined Data.

(STRS-106) An STRS application shall use the appropriate constant, typedef, or struct defined in Table 74, STRS Predefined Data, when the data are used to interact with the STRS APIs.

NASA-STD-4009A

Table 74—STRS Predefined Data

Typedefs		
Name	Type	Description
STRS_Access	A number	Used to indicate how reading and/or writing of a file or queue is done. See also constants STRS_ACCESS_APPEND, STRS_ACCESS_BOTH, STRS_ACCESS_READ, and STRS_ACCESS_WRITE.
STRS_Buffer_Size	A number	Used to represent a buffer size in bytes. The type of the number is to be long enough to contain the maximum number of bytes to reserve or to transfer with a read or write.
STRS_Clock_Kind	A number	Used to represent a kind of clock or timer. The type of the number is to be long enough to contain the maximum number of kinds of clocks and timers.
STRS_File_Size	A number	Used to represent a size in bytes. The type of the number is to be long enough to contain the number of bytes in GPP storage. Specific negative error values returned indicate an error.
STRS_HandleID	A number	Used to represent an STRS application, device, file, or queue. Specific error value(s) returned indicate an error.
STRS_int8	A number	Used for an 8-bit signed integer
STRS_int16	A number	Used for a 16-bit signed integer
STRS_int32	A number	Used for a 32-bit signed integer
STRS_int64	A number	Used for a 64-bit signed integer
STRS_ISR_Function	A static C-style function pointer	Used to define static C-style function pointers passed to the STRS SetISR() method. The function passed to the STRS_SetISR() method is defined with any arguments needed by the OE for its underlying system calls. The OE-specific documentation contains the description of any arguments.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Typedefs		
Name	Type	Description
STRS_Message	A char array pointer	Used for messages.
STRS_Nanoseconds*	A number	Used to hold the number of nanoseconds in the STRS_TimeWarp object, at least 32 bits for a signed integer. Using 32 bits would allow a maximum of 2,147,483,647 nanoseconds = 2.147483647 seconds that would allow the sum/difference of the nanosecond counter in 2 normalized STRS_TimeWarp objects. Each additional bit multiplies the 2.1475 seconds in the nanosecond counter by 2.
STRS_Property_Name		Used to hold a property name, usually a set of characters (char).
STRS_Property_Value		Used to hold a property value, usually a set of characters (char).
STRS_Queue_Max_Messages	A number	Used to represent the maximum number of messages allowed in the queue.
STRS_Result	A number	Used to represent a return value, where there are specific values that indicate an error.
STRS_Seconds*	A number	Used to hold the number of seconds in the STRS_TimeWarp object, at least 32 bit signed integer. Using 32 bits would allow a maximum of 2,147,483,647 seconds = 68.05 years. For an epoch of 1970, the 32-bit second counter runs out in 2038. Each additional bit multiplies the 68.05 years in the second counter by 2.
STRS_TestID	A number	Used to represent the built-in test or ground test to be performed by APP_RunTest or APP_GroundTest, respectively. See also STRS_TEST_STATUS and STRS_TEST_USER_BASE.
STRS_TimeWarp	A representation of a time delay	The representation of a time delay able to hold the number of seconds and nanoseconds in the time delay so that the corresponding macros can extract them. The time delay is meant to be used for recurrent processes such as in health management. The implementation is mission and/or platform specific and is most likely a

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Typedefs		
Name	Type	Description
		struct. The maximum number of seconds in a time delay cannot be greater than $2^{(\text{no. of bits in STRS_Seconds} - 1)}$ seconds. Divide the maximum number of seconds by 31557600 ($60*60*24*365.25$) to get the approximate number of years.
STRS_TimeRate	A number	Integer indicating time rate factor used to adjust time relative to clock accuracy defined by STRS_TimeRatePPS.
STRS_Type	A number	Used to indicate whether a file is text or binary. See also constants STRS_TYPE_BINARY and STRS_TYPE_TEXT.
STRS_uint8	A number	Used for an 8-bit unsigned integer
STRS_uint16	A number	Used for a 16-bit unsigned integer
STRS_uint32	A number	Used for a 32-bit unsigned integer
STRS_uint64	A number	Used for a 64-bit unsigned integer

Constants		
Name	Type	Description
STRS_ACCESS_APPEND	STRS_Access	Indicates that writing is allowed such that previous data written are preserved and new data are written following any previous data. Corresponds to ISO C fopen mode “a”.
STRS_ACCESS_BOTH	STRS_Access	Indicates that both reading and writing are allowed. Corresponds to ISO C fopen mode “r+” used for update.
STRS_ACCESS_READ	STRS_Access	Indicates that reading is allowed. Corresponds to ISO C fopen mode “r”.
STRS_ACCESS_WRITE	STRS_Access	Indicates that writing is allowed. Corresponds to ISO C fopen mode “w”.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Constants		
Name	Type	Description
STRS_OK	STRS_Result	Indicates that the STRS_Result is valid. See also STRS_IsOK().
STRS_ERROR	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that the application or other component is still usable. See also STRS_IsOK() and STRS_GetErrorQueue().
STRS_ERROR_QUEUE	STRS_HandleID	Indicates that the log queue is for error messages. See also STRS_GetErrorQueue().
STRS_FATAL	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating a serious error such that the application or other component is not usable. See also STRS_IsOK() and STRS_GetErrorQueue().
STRS_FATAL_QUEUE	STRS_HandleID	Indicates that the log queue is for fatal messages. The fatal queue is used for messages that the fault monitoring and recovery functions are to deal with immediately. The messages are sent to the Flight Computer for further handling. See also STRS_GetErrorQueue().
STRS_TELEMETRY_QUEUE	STRS_HandleID	Indicates that the log queue is for telemetry data.
STRS_TYPE_BINARY	STRS_Type	Indicates that a file is a binary file.
STRS_TYPE_TEXT	STRS_Type	Indicates that a file is a text file.
STRS_WARNING	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that there may be little or no effect on the operation of the application or other component. See also STRS_IsOK() and STRS_GetErrorQueue().

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Constants		
Name	Type	Description
STRS_WARNING_QUEUE	STRS_HandleID	Indicates that the log queue is for warning messages. See also STRS_GetErrorQueue().
STRS_OE_HANDLE_NAME	char *	A handle name used to find handle ID that may be used to query the OE.
OEClockAppName	char *	The handle name used to find handle ID that may be used to access time using STRS_GetTime used in a timestamp.
OEClockKind	STRS_Clock_Kind	The number used to indicate the type of clock/timer used in a timestamp.
STRS_TimeRatePPS	STRS_TimeRate	Integer accuracy of time rate in number of parts per second.
STRS_MAX_PROPERTY_NAME_SIZE	#define	The maximum number of characters in the name in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.
STRS_MAX_PROPERTY_VALUE_SIZE	#define	The maximum number of characters in the value in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.
STRS_MAX_PATH_NAME_SIZE	#define	The maximum number of characters in a path name for the OE, not including the final '\0'. Any use of this as a dimension should be increased by one.
STRS_MAX_HANDLE_NAME_SIZE	#define	The maximum number of characters in a handle name for the OE, not including the final '\0'. Any use of this as a dimension should be increased by one.
STRS_MAX_LOG_MESSAGE_SIZE	#define	The maximum number of characters in each message submitted to the log, not including the final '\0'. Any use of this as a dimension should be increased by one.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Constants		
Name	Type	Description
STRS_MAX_QUEUE_MESSAGES	STRS_Queue_Max_Messages	The maximum number of messages that can be stored in a queue. Not normally used except for testing.

The STRS Predefined Data is used internally and the STRS Queryable Data is used externally. The STRS Queryable Data may be used to verify the version of the Application or OE, and to be able to notice when it changes.

(STRS-134) The STRS infrastructure shall have the queryable parameter names in Table 75, Queryable Platform Parameter Names, for which values may be obtained using STRS_Query with the handle ID corresponding to the handle name STRS_OE_HANDLE_NAME.

Table 75—Queryable Platform Parameter Names

Parameter Name	Description	Notes
PLATFORM_PROVIDER	Unique name of STRS platform provider	This is usually a company name or university, followed by a subsidiary, division, or department name.
PLATFORM_OE_VERSION	Unique version number for platform STRS infrastructure software	

(STRS-135) An STRS application shall have the queryable parameter names in Table 76, Queryable Application Parameter Names, for which values may be obtained using STRS_Query with the handle ID of the application.

Table 76—Queryable Application Parameter Names

Parameter Name	Description	Notes
WAVEFORM_DEVELOPER	Unique name of application developer	This is usually a company name or university, followed by a subsidiary, division, or department name.
WAVEFORM_VERSION	Unique version number for STRS application software	
WAVEFORM_STATE	Current application state	Documented per STRS-12(3)

7.3.12 Error Handling

Special-purpose handle IDs for errors include the following: STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, and STRS_FATAL_QUEUE. The STRS_Log method uses these special-purpose handle IDs to log errors. A nonfatal error is a correctable condition such that the application is usable when the error is corrected. This nonfatal error is denoted by the STRS return value of STRS_ERROR and is logged using the STRS handle ID of

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

STRS_ERROR_QUEUE. A warning is an indication of an impending error that is correctable if action is taken. This warning is denoted by the STRS return value of STRS_WARNING and is logged using the STRS handle ID of STRS_WARNING_QUEUE. A fatal error is a condition where the application is subsequently not usable and a reboot or reload is often necessary. This fatal error is denoted by the STRS return value of STRS_FATAL and is logged using the STRS handle ID of STRS_FATAL_QUEUE.

7.4 Portable Operating System Interface

STRS applications and services can use a subset of the POSIX® API as shown in Figure 9 and discussed in more detail in this section. POSIX® refers to a family of IEEE standards 1003.n that describe the fundamental services and functions necessary to provide a UNIX®-like kernel interface to applications. POSIX® itself is not an OS but is instead the guaranteed programming interfaces available to the application programmer.

POSIX® specifies a set of OS interfaces and services. POSIX® is not specifically bound to a specific OS, and has in fact been implemented on top of OS such as Digital Equipment Corporation's (DEC's) OpenVMS™ (Virtual Memory System) and Microsoft Windows NT®. However, the creation of POSIX® is closely coupled to the UNIX® OS and its evolution. The goal was to create a standard set of interfaces that all of the UNIX® flavors would support in order to facilitate software portability. Even though POSIX® technically refers to the family of specifications, it is more commonly used to refer specifically to IEEE 1003.1, Information Technology - Portable Operating System Interface (POSIX®), which is the core POSIX® specification.

Characteristics of POSIX® include the following:

- a. Application-oriented.*
- b. Interface, not implementation.*
- c. Source, not object, portability.*
- d. The C-language/system interfaces written in terms of the ISO C standard.*
- e. No superuser, no system administration.*
- f. Minimal interface, minimally defined—core facilities of this NASA Technical Standard have been kept as minimal as possible.*
- g. Broadly implementable.*
- h. Minimal changes to historical implementations.*
- i. Minimal changes to existing application code.*

The original POSIX® specification was based on a general purpose computing platform, but a series of amendments addressed the unique requirements of real-time computing. These amendments follow:

- a. IEEE 1003.1B-Realtime Extension.*
- b. IEEE 1003.1C-Threads Extension.*

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

- c. *IEEE 1003.1D-Additional Realtime Extensions.*
- d. *IEEE 1003.1J-Advanced Realtime Extensions.*
- e. *IEEE 1003.1Q-Tracing.*

These amendments were rolled into the base specification in version IEEE 1003.1-1996. IEEE 1003.13 provides a standards-based option for an STRS AEP.

7.4.1 STRS Application Environment Profile

The subset of the POSIX® API described below is used by STRS applications to access platform services when no STRS Infrastructure-provided API is available. POSIX® was chosen as part of this NASA Technical Standard because it defines an open-standard OS interface and environment to support application portability. However, because of the limited resources on a space-based platform, it was not practical to support the entire IEEE 1003.1 specification.

The IEEE 1003.1 standard provides a means to implement a subset of the interfaces by using “Subprofiling Option Groups.” These option groups specify “Units of Functionality” that can be removed from the base POSIX® specification.

IEEE 1003.13 created four AEPs that specified subsets of 1003.1 more suitable to embedded applications. These profiles follow:

- *PSE51—Minimal Realtime Systems Profile.*
- *PSE52—Realtime Controller System Profile.*
- *PSE53—Dedicated Realtime System Profile.*
- *PSE54—Multi-Purpose Realtime System Profile.*

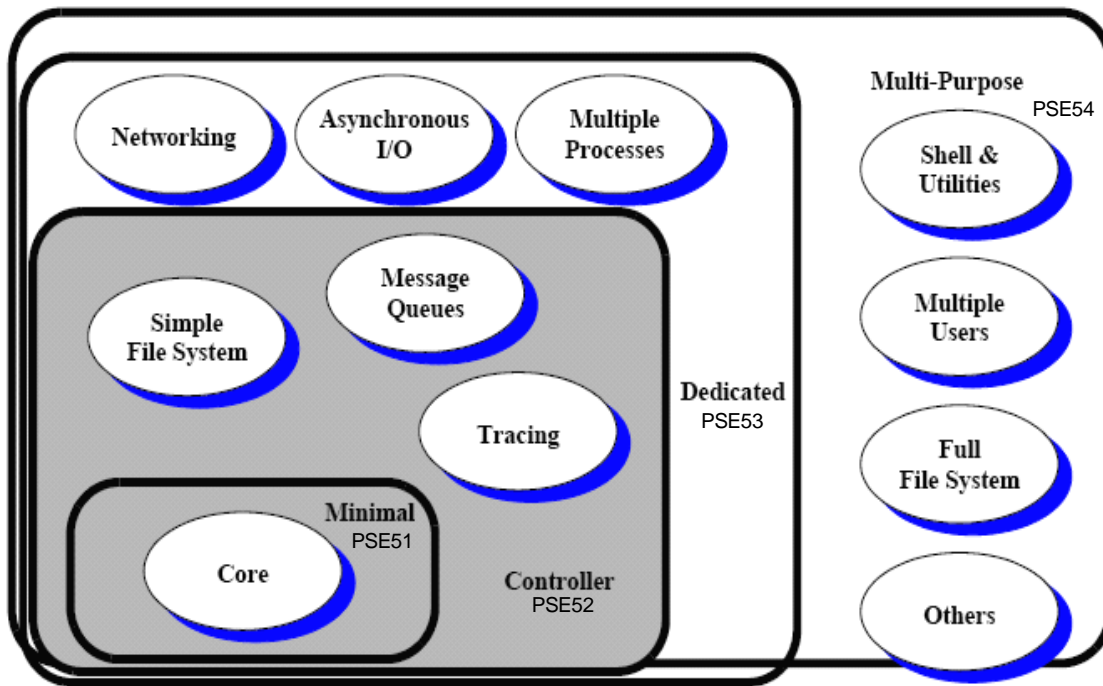


Figure 15—Profile Building Blocks

The profiles are each upwardly compatible and consist of the basic building blocks shown in Figure 15,¹ Profile Building Blocks.

Each of these profiles has increasing capabilities, which increase requirements on resources. Profiles 51 and 52 runs on a single processor with no Memory Management Unit (MMU), and thus imply a single process containing one or more threads. Profile 52 adds a file system interface and asynchronous I/O. Profile 53 adds support for multiple processes, thus requiring an MMU. The last and largest profile 54 adds support for interactive users, and is almost a full-blown POSIX® 1003.1 environment. The higher numbered profiles are supersets of the lower numbered profiles, such that PSE52 includes all the features of a PSE51.

Upward portability between profiles is supported by requiring certain APIs, such as memory locking, for profiles PSE51 and PSE52. Even though there is no MMU support on the PSE51 and PSE52 profiles, code written as if there is an MMU present will be portable among all four profiles by requiring such APIs to be defined in all four profiles. The signature of these APIs will be identical on all profiles, but the functionality will differ according to the capabilities. For example, calling a memory-locking API on a PSE51 platform with no MMU will always return success. When this example application is ported to a PSE53 platform, the memory locking will work as intended without modification to the source code.

Currently, this NASA Technical Standard supports platforms based on profiles PSE51 through PSE54, although PSE54 will only be used for development platforms and ground stations.

¹ IEEE 1003.13-2003

NASA-STD-4009A

Allowing multiple profiles allows the architecture to scale with mission class. Applications developed for a specific profile are compatible with higher profiles; that is, a profile 52 application could be ported to profile PSE53 and PSE54 platform, but not vice versa. This upward scalability anticipates that smaller platforms will desire smaller profiles and will not have the resources to run larger applications that comply with the larger profiles. Appendix A provides a table comparing the POSIX® profile functionality for subset PSE51 through PSE53.

(STRS-90) The STRS OE shall provide the interfaces described in POSIX® standard IEEE 1003.13 profile PSE51.

For constrained-resource platforms with limited software evolutionary capability, where the waveform signal processing is implemented in specialized hardware, the supplier may request a waiver to only implement a subset of POSIX® PSE51 as required by the portion of the waveforms residing on the GPP. The applications created for this platform are to be upward-compatible to a larger platform containing POSIX® PSE51. The POSIX® API is grouped into units of functionality. If none of the applications for a constrained-resource platform use any of the interfaces in a unit of functionality, then the supplier may request a waiver to eliminate that entire unit of functionality.

Regardless of the POSIX® profile implemented, applications are not to use any restricted functions or their equivalent, such as abort(), atexit(), exit(), calloc(), free(), malloc(), or realloc(). For portability of application code to multithreaded STRS platforms, STRS applications are to use the thread-safe versions of the POSIX® methods listed in Table 77, Replacements for Unsafe Functions.

(STRS-91) STRS applications shall use POSIX® methods except for the unsafe functions listed in Table 77, Replacements for Unsafe Functions.

Table 77—Replacements for Unsafe Functions

Unsafe Function Do Not Use!	Reentrant Counterpart OK to Use
abort	STRS_AbortApp
asctime	asctime_r
atexit	-
ctermid	ctermid_r
ctime	ctime_r
exit	STRS_AbortApp
getlogin	getlogin_r
gmtime	gmtime_r
localtime	localtime_r
rand	rand_r
readdir	readdir_r
strtok	strtok_r
tmpnam	tmpnam_r

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

7.5 Network Stack

A network stack is the part of the OS used for networking, usually Transmission Control Protocol/Internet Property (TCP/IP). Communications over a network use a layered network model. TCP/IP is the protocol that is used to transport information over the Internet, and the TCP/IP network model consists of five layers: the application layer, the transport layer, the network layer, the data link layer, and the physical network.

7.6 Operating System

The OS is an integral part of the OE for the STRS software architecture. Modern communication systems perform simultaneous application processing in dedicated hardware at the very fast speeds to which users have become accustomed. Any change in this environment is to equal or exceed previous performance for it to be considered for usage. As such, the proposal to perform application processing via software modules executing on a GPP involves careful consideration of both the necessary OS characteristics and the application processing requirements. In a simplistic sense, a computer OS manages the usage and sharing of resources between competing users (i.e., tasks) to perform work. In this case, each task is performing a specific instance of application processing. When the OS decides to stop the execution of one task and start another, the current context of the machine (register values, instruction pointers, etc.) is to be saved and then switched to accommodate the requirements of the new task. On a desktop computer system, context switching between competing tasks is performed on an ad-hoc basis with no guarantee of task execution. For most missions, this is unacceptable because context switching between execution threads and deterministic thread execution are the driving characteristics for an OS.

To support these requirements, most STRS platforms will use an RTOS instead of a general-purpose OS. An RTOS provides the capabilities of fast, low overhead for context switching, and a deterministic scheduling mechanism so that processing constraints can be achieved when required.

Fundamental to STRS application development is the existence of an OS kernel that can be configured and scaled down to fit into the executable image of the STRS system. A modern RTOS is primarily designed for either performance (monolithic kernel) or extensibility (microkernel). Monolithic kernels have tightly integrated services and less run-time overhead but are not easily extensible. Microkernels have somewhat high run-time overheads but are highly extensible. Most modern RTOSs are microkernels, and although modern microkernels have more overhead than monolithic kernels, they have less overhead than traditional microkernels. The run-time overhead of modern RTOSs is decreased by reducing the unnecessary context switch. Important timings such as context switch time, interrupt latency, and semaphore get and release latency is to be kept to a minimum.

7.7 Hardware Abstraction Layer

The HAL is the library of software functions in the STRS OE that provides a platform-vendor-specific view of the specialized hardware by abstracting the underlying physical hardware

NASA-STD-4009A

interfaces. The HAL allows specialized hardware to be integrated with the GPM so that the STRS OE can access functions implemented on the specialized hardware of the STRS platform.

Two examples of specialized hardware currently in use on SDRs are FPGAs and DSPs. Examples of functionality that a HAL might need to support include boot code for initializing the hardware and loading the OS image, context switch code, configuration and access to hardware resources. The HAL is commonly referred to by platform vendors as drivers or BSPs. Most companies already provide such libraries to allow use of specialized hardware. This layer enables the STRS infrastructure to have a direct interface to the hardware drivers on the platform.

There are two requirements concerning the HAL in the STRS architecture:

a. STRS-11 requires a HAL software API, which defines the physical and logical interfaces for intermodule and intramodule integration. The HAL is required for communicating data and control information between the GPP and the specialized hardware. The HAL API is not currently defined in this NASA Technical Standard but is left for the STRS platform provider to specify.

b. STRS-92 requires HAL documentation that includes a description of each method, its calling sequence, the return values, an explanation of the functionality, preconditions for using the method, postconditions after using the method, and examples where helpful. Note that the delivery of the HAL source code is not required.

The electrical interfaces, connector requirements, and physical requirements are specified by the STRS platform provider in the HID. Information on a module's use of data in the HID will be made available to STRS application developers, either directly from the manufacturer (for specific types of components) or from the STRS platform provider (for memory maps based on electrical connections). The infrastructure or HAL may use this information to appropriately initialize hardware drivers such that control and data messages are delivered to the module.

Even though there is not a requirement for the STRS OE to be portable, the HAL is expected to foster portability and reusability of the STRS infrastructure and specialized hardware in different combinations from that originally designed. It can reduce the design efforts otherwise necessary to adapt the software to a new hardware platform. The goal with the HAL is to make it easier to change or add new hardware and to minimize the impact to the software. It does this by localizing the differences in software so that most of the STRS OE code does not need to be changed to run on a new platform or a platform with a new module.

Table 78, Sample HAL Documentation, shows an example of the HAL API for the function OPEN.

(STRS-92) The STRS platform provider shall provide the STRS platform HAL documentation that includes the following:

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

- (1) For each method or function, its calling sequence, return values, an explanation of its functionality, any preconditions for using the method or function, and the postconditions after using the method or function.
- (2) Information required to address the underlying hardware, including the interrupt input and output, the memory mapping, and the configuration data necessary to operate in the STRS platform environment.

Table 78—Sample HAL Documentation

HAL API	RESULT OPEN(HANDLE* resourceHandle, RESOURCE_NAME resourceName)
Description	Open a resource by name. If no errors are encountered, use the resourceHandle to access the resource.
Parameters	<ul style="list-style-type: none"> • resourceHandle - [out] A pointer to place the opened handle into • resourceName - [in] The name of the resource to open
Return	<p>A 32-bit signed integer used to determine whether an error has occurred. Use TEST_ERROR to obtain a printable message.</p> <ul style="list-style-type: none"> • Zero - No errors or warnings. • Positive – Warning. • Negative – Error.
Precondition	Resource is not open before executing this command.
Postcondition	Resource will be open and ready for further access if no error was encountered.
See Also	READ, WRITE, CLOSE, TEST_ERROR
Example	<pre>#include <HALResources.h> ... RESULT result; HANDLE resourceHandle; RESOURCE_NAME resourceName = "FPGA"; result = OPEN(&resourceHandle, resourceName) if (result < 0) { cout << "Error: " << TEST_ERROR(result) << endl; } else if (result > 0) { cout << "Warning: " << TEST_ERROR(result) << endl; }</pre>

8. EXTERNAL COMMAND AND TELEMETRY INTERFACES

An STRS radio cannot perform the necessary application and platform functions without an external system providing commands, accepting responses, and monitoring the radio's health and status. The STRS radio implements an external interface to receive and act on the commands from the external system, translates the commands into the format expected by the application, and provides the information for monitoring the health and status of the radio. If the STRS radio has the capability for new or modified OE, application software, or configurable hardware design, the external command and telemetry interfaces should be able to accept and store new files. The interface in the STRS radio and in the external system, which is to provide the control, via a command sequence, to the STRS radio and receive responses from an STRS radio, is referred to as the STRS command and telemetry interfaces. The external STRS command and telemetry functionality illustrated in Figure 16, Command and Telemetry Interfaces, typically resides on the spacecraft's flight computer, and/or it may reside on a ground station or another spacecraft.

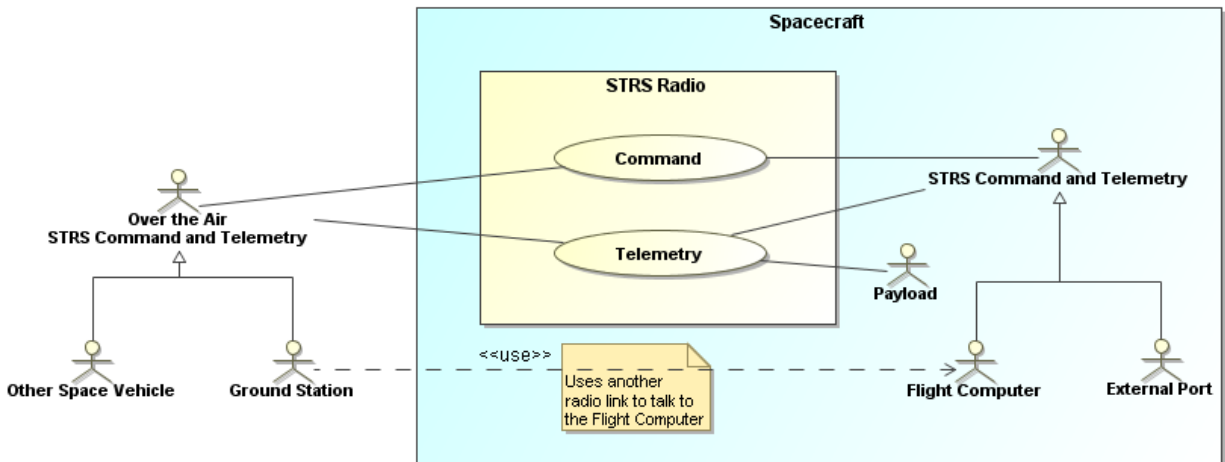


Figure 16—Command and Telemetry Interfaces

This shared capability implies that the STRS radio is capable of performing the interface functions. Within the STRS radio, if there are data stored on the radio that are to be transferred to an external system, the capability is to exist to send data using a mission-specific protocol to the receiver (flight computer, ground station, or other spacecraft) and capability in the receiver to process those data or write those data to a file or download service or to a storage area that is accessible from both. The reverse capability for STRS radio control is also necessary: The external system is capable of sending commands using a mission-specific protocol and the STRS radio is capable of validating, deciphering, and processing those commands. For example, data coming over the Flight Computer Interface are interpreted by the Command and Control Manager as shown in Figure 13 and are processed by the STRS infrastructure.

Within the STRS radio, components of the command and telemetry interfaces are necessary to provide the interfaces between the STRS OE and the STRS command and telemetry functionality on the external system. The command and telemetry interfaces may include a standard type of

NASA-STD-4009A

mechanical, electrical, and functional spacecraft bus interface, such as MIL-STD-1553, Digital Time Division Command/Response Multiplex Data Bus; command and telemetry interpretation; and translation of the command set to the STRS standard necessary for application control. The protocol, command set, and telemetry set for the STRS command and telemetry interfaces are NOT part of the STRS standard but can be unique to each mission. A number of interface and behavior requirements are part of the standard to support the mission-specific protocols.

The requirements related to the external command and telemetry interfaces follow:

(STRS-94) An STRS platform shall accept, validate, and respond to external commands.

(STRS-95) An STRS platform shall execute external application control commands using the standardized STRS APIs.

(STRS-107) An STRS platform provider shall document the external commands describing their format, function, and any STRS methods invoked.

(STRS-96) The STRS infrastructure shall use the STRS_Query method to service external system requests for information and to provide telemetry data about an STRS application.

The STRS telemetry set will be mission-specific but will likely contain some or all of the following parameters:

- a. Power values.*
 - (1) Voltage, current, and power readings.*
- b. Environment values.*
 - (1) Temperature.*
 - (2) Pressure.*
- c. Power on reset test result status.*
 - (1) RAM test.*
 - (2) Read-only memory (ROM) test.*
 - (3) File management test.*
 - (4) PROM software revision.*
 - (5) Maximum memory configuration.*
 - (6) Individual module self-test status (GO/NO GO).*
- d. Module configuration.*
 - (1) Module type.*
 - (2) Module location.*
 - (3) Hardware revision.*
- e. Application-specific parameters.*
- f. Language support (C and/or C++).*

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

- g. *STRS Architecture Standard version.*
- h. *STRS OE release version.*
- i. *Available memory and free space for data and files.*

A suggested set of services that may be implemented by the STRS command and telemetry interfaces on the external system (flight computer, ground station, or other spacecraft) is shown in Table 79, Suggested Services Implemented by the STRS Command and Telemetry Interfaces. These services are NOT required for the STRS Architecture Standard at this time, but are likely needed for commanding and controlling an SDR and are expected to be part of the external system set of required functions.

**Table 79—Suggested Services Implemented by the
STRS Command and Telemetry Interfaces**

Function	Description
Application Control	
Application Selection	This command requests that the STRS radio instantiate the application and facilitate the installation of devices and resources requested by the application. This service should not impact existing applications. The command arguments will include the application ASCII handle name and string used to identify the application for instantiation.
Application Configuration	This command requests a customization of the application by specifying parameters the application will use.
Application Query	This command requests the current parameters and operational values of the application.
Application Start	This command requests that an initialized application begin processing application data. If the application has not been selected or completed initialization, the command will be rejected.
Application Stop	This command requests that a running application halt processing of application data. The application resources are not deallocated.
Application Unload	This command requests that the STRS infrastructure unload the identified application and release all resources associated with the application.
File Control Interface	
Upload File Request	This request will initiate an upload of a file to the STRS radio and place it in a specified location. If the command gets an error, the reason will be made available.
Delete File Request	This is a request for the deletion of a specified file from an STRS platform.
Download File Request	This request is complementary to the Upload File Request. This command will initiate a download of a specified file from the STRS platform.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Radio Control Interface	
Built-in-test	This request will perform a commanded built-in-test used to monitor the health of the radio and diagnose any problems.
Telemetry Control Interface	
Telemetry Control	Several different telemetry structure definitions may exist for different classes of STRS radios. Many systems will employ a polling technique where the data are provided only upon request. Other systems may desire a grouping of telemetry that can be identified to be sent at some periodic rate.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

APPENDIX A

POSIX® API PROFILE

Appendix A provides a list of the POSIX® profile recommended as part of the application abstraction.

Table 80, POSIX® Subset Profiles PSE51, PSE52, and PSE53 provides the POSIX® subset in profiles PSE51, PSE52, and PSE53.

Table 80—POSIX® Subset Profiles PSE51, PSE52, and PSE53

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_C_LANG_JUMP	longjmp(), setjmp()	X	X	X
POSIX_C_LANG_MATH	acos(), acosf(), acosh(), acoshf(), acoshl(), acosl(), asin(), asinf(), asinh(), asinhf(), asinhl(), asinl(), catan(), atan2(), atan2f(), atan2l(), atanf(), atanh(), atanhf(), atanh(), atanl(), cabs(), cabsf(), cabsl(), cacos(), cacosf(), cacosh(), cacoshf(), cacoshl(), cacosl(), carg(), cargf(), cargl(), casin(), casinf(), casinh(), casinhf(), casinhl(), casinl(), catan(), catanf(), catanh(), catanhf(), catanh(), catanl(), cbrt(), cbrtf(), cbrtl(), ccos(), ccosf(), ccosh(), ccoshf(), ccoshl(),		X	X
POSIX_C_LANG_MATH	ccosl(), ceil(), ceilf(), ceil(), cexp(), cexpf(), cexpl(), cimag(), cimagf(), cimagl(), clog(), clogf(), clogl(), conj(), conjf(), conjl(), copysign(), copysignf(), copysignl(), cos(), cosf(), cosh(), coshf(), coshl(), cosl(), cpow(), cpowf(), cpowl(), cproj(), cprojf(), cprojl(), creal(), crealf(), creall(), csin(), csinf(), csinh(), csinhf(), csinhl(), csinl(), csqrt(), csqrtf(), csqrtl(), ctan(), ctanf(), ctanh(), ctanhf(), ctanh(), ctanl(), erf(), erfc(), erfcf(), erfcl(), erff(), erfl(), exp(), exp2(), exp2f(), exp2l(), expf(), expl(), expm1(), expm1f(), expm1l(), fabs(), fabsf(), fabsl(), fdim(), fdimf(), fdiml(), floor(), floorf(), floorl(), fma(), fmaf(), fmal(),		X	X

NASA-STD-4009A

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_C_LANG_MATH	fmax(), fmaxf(), fmaxl(), fmin(), fminf(), fminl(), fmod(), fmodf(), fmodl(), fpclassify(), frexp(), frexpf(), frexpl(), hypot(), hypotf(), hypotl(), ilogb(), ilogbf(), ilogbl(), isfinite(), isgreater(), isgreaterequal(), isinf(), isless(), islessequal(), islessgreater(), isnan(), isnormal(), isunordered(), ldexp(), ldexpf(), ldexpl(), lgamma(), lgammaf(), lgammal(), llrint(), llrintf(), llrintl(), llround(), llroundf(), llroundl(), log(), log10(), log10f(), log10l(), log1p(), log1pf(), log1pl(), log2(), log2f(), log2l(), logb(), logbf(), logbl(), logf(), logl(), lrint(), lrintf(), lrintl(), lround(), lroundf(), lroundl(), modf(), modff(), modfl(), nan(), nanf(), nanl(), nearbyint(), nearbyintf(), nearbyintl(), nextafter(), nextafterf(), nextafterl(), nexttoward(), nexttowardf(), nexttowardl(), pow(), powf(), powl(), remainder(), remainderf(), remainderl(), remquo(), remquoof(), remquoofl(), rint(), rintf(), rintl(), round(), roundf(), roundl(), scalbln(), scalblnf(), scalblnl(), scalbn(), scalbnf(), scalbnl(), signbit(), sin(), sinf(), sinh(), sinhlf(), sinhl(), sinl(), sqrt(), sqrtf(), sqrtl(), tan(), tanf(), tanh(), tanhf(), tanhl(), tanl(), tgamma(), tgammaf(), tgamma(), trunc(), truncf(), trunc()		X	X

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_C_LANG_SUPPORT	abs(), asctime(), asctime_r(), atof(), atoi(), atol(),atoll(), bsearch(), calloc(), ctime(), ctime_r(),difftime(), div(), feclearexcept(), fegetenv(), fegetexceptflag(), fegetround(), fholdexcept(),feraiseexcept(), fesetenv(), fesetexceptflag(), fesetround(), fetestexcept(), feupdateenv(), free(),gmtime(), gmtime_r(), imaxabs(), imaxdiv(), isalnum(), isalpha(), isblank(), iscntrl(), isdigit(),isgraph(), islower(), isprint(), ispunct(), isspace(),isupper(), isxdigit(), labs(), ldiv(), llabs(), lldiv(), localeconv(), localtime(), localtime_r(), malloc(),memchr(), memcmp(), memcpy(), memmove(),memset(), mktime(), qsort(), rand(), rand_r(), realloc(), setlocale(), snprintf(), sprintf(), srand(),sscanf(), strcat(), strchr(), strcmp(), strcoll(), strcpy(),strcspn(), strerror(), strerror_r(), strptime(), strlen(), strncat(), strncmp(), strncpy(), strpbrk(), strrchr(), strspn(), strstr(), strtod(), strtod(), strtoimax(),strtok(), strtok_r(), strtol(), strtold(), strtoll(), strtoul(), strtoull(), strtoumax(), strxfrm(), time(),tolower(), toupper(), tzname, tzset(), va_arg(),va_copy(), va_end(), va_start(), vsnprintf(), vsprintf(), vsscanf()	X	X	X
POSIX_DEVICE_IO	clearerr(), close(), fclose(), fdopen(), feof(), ferror(),fflush(), fgetc(), fgets(), fileno(), fopen(), fprintf(), fputc(), fputs(), fread(), freopen(), fscanf(), fwrite(),getc(), getchar(), gets(), open(), perror(), printf(), putc(), putchar(), puts(), read(), scanf(), setbuf(),setvbuf(), stderr, stdin, stdout, ungetc(), vfprintf(), vfscanf(), vprintf(), vscanf(), write()	X	X	X

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_EVENT_MGMT	FD_CLR(), FD_ISSET(), FD_SET(), FD_ZERO(), pselect(), select()			X
POSIX_FD_MGMT	dup(), dup2(), fcntl(), fgetpos(), fseek(), fseeko(), fsetpos(), ftell(), ftello(), ftruncate(), lseek(), rewind()		X	X
POSIX_FILE_LOCKING	flockfile(), ftrylockfile(), funlockfile(), getc_unlocked(), getchar_unlocked(), putc_unlocked(), putchar_unlocked()	X	X	X
POSIX_FILE_SYSTEM	access(), chdir(), closedir(), creat(), fpathconf(), fstat(), getcwd(), link(), mkdir(), opendir(), pathconf(), readdir(), readdir_r(), remove(), rename(), rewinddir(), rmdir(), stat(), tmpfile(), tmpnam(), unlink(), utime()		X	X
POSIX_MULTI_PROCESS	_Exit(), _exit(), assert(), atexit(), clock(), execl(), execle(), execlp(), execv(), execve(), execvp(), exit(), fork(), getpgrp(), getpid(), getppid(), setsid(), sleep(), times(), wait(), waitpid()			X

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_NETWORKING	accept(), bind(), connect(), endhostent(), endnetent(), endprotoent(), endservent(), freeaddrinfo(), gai_strerror(), getaddrinfo(), gethostbyaddr(), gethostbyname(), gethostent(), gethostname(), getnameinfo(), getnetbyaddr(), getnetbyname(), getnetent(), getpeername(), getprotobyname(), getprotobynumber(), getprotoent(), getservbyname(), getservbyport(), getservent(), getsockname(), getsockopt(), h_errno, htonl(), htons(), if_freenameindex(), if_indextoname(), if_nameindex(), if_nametoindex(), inet_addr(), inet_ntoa(), inet_ntop(), inet_pton(), listen(), ntohl(), ntohs(), recv(), recvfrom(), recvmsg(), send(), sendmsg(), sendto(), sethostent(), setnetent(), setprotoent(), setservent(), setsockopt(), shutdown(), socket(), socketatmark(), socketpair()			X
POSIX_PIPE	pipe()			X
POSIX_SIGNALS	abort(), alarm(), kill(), pause(), raise(), sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), signal(), sigpending(), sigprocmask(), sigsuspend(), sigwait()	X	X	X
POSIX_SIGNAL_JUMP	siglongjmp(), sigsetjmp()			X
POSIX_SINGLE_PROCESS	confstr(), environ, errno, getenv(), setenv(), sysconf(), uname(), unsetenv()	X	X	X

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Unit of Functionality	Interfaces	PSE51	PSE52	PSE53
POSIX_THREADS_BASE	pthread_atfork(), pthread_attr_destroy(), pthread_attr_getdetachstate(), pthread_attr_getschedparam(), pthread_attr_init(), pthread_attr_setdetachstate(), pthread_attr_setschedparam(), pthread_cancel(), pthread_cleanup_pop(), pthread_cleanup_push(), pthread_cond_broadcast(), pthread_cond_destroy(), pthread_cond_init(), pthread_cond_signal(), pthread_cond_timedwait(), pthread_cond_wait(), pthread_condattr_destroy(), pthread_condattr_init(), pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(), pthread_getspecific(), pthread_join(), pthread_key_create(), pthread_key_delete(), pthread_kill(), pthread_mutex_destroy(), pthread_mutex_init(), pthread_mutex_lock(), pthread_mutex_trylock(), pthread_mutex_unlock(), pthread_mutexattr_destroy(), pthread_mutexattr_init(), pthread_once(), pthread_self(), pthread_setcancelstate(), pthread_setcanceltype(), pthread_setspecific(), pthread_sigmask(), pthread_testcancel()	X	X	X
POSIX_THREAD_MUTEX_EXT	pthread_mutexattr_gettype(), pthread_mutexattr_settype()	X	X	X
XSI_THREADS_EXT	pthread_attr_getguardsize(), pthread_attr_getstack(), pthread_attr_setguardsize(), pthread_attr_setstack(), pthread_getconcurrency(), pthread_setconcurrency()	X	X	X

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

APPENDIX B

REFERENCE DOCUMENTS

The following reference documents are recommended for further guidance.

Department of Defense

Document Number	Document Title
MIL-STD-1553	Digital Time Division Command/Response Multiplex Data Bus
SCA Version 2.2.2	Software Communications Architecture Specification http://www.wirelessinnovation.org/assets/work_products/sca_version_2_2_2.pdf

National Aeronautics and Space Administration

Document Number	Document Title
NASA-HDBK-4009	Space Telecommunications Radio Systems (STRS) Architecture Standard Rationale
NASA/TM—2007-215042	Space Telecommunications Radio System (STRS) Architecture Goals/Objectives and Level 1 Requirements
NASA/TP—2008-214813	Space Telecommunications Radio System Software Architecture Concepts and Analysis
NASA/TM—2008-215445	Space Telecommunications Radio System (STRS) Definitions and Acronyms
NASA/TM—2010-216809	Space Telecommunications Radio System (STRS) Architecture Standard. Release 1.02.1
NPR 2210.1	Release of NASA Software
STRS Website	Space Telecommunications Radio System (STRS) Website

NASA-STD-4009A

National Institute of Standards and Technology

Document Number	Document Title
FIPS PUB 140-2	Security Requirements for Cryptographic Modules

Non-Government Documents

Institute of Electrical and Electronics Engineers (IEEE)

Document Number	Document Title
IEEE 1003.1™	IEEE Standard for Information Technology—Portable Operating System Interface (POSIX®)

Use the latest version of IEEE 1003.1 for your platform; the following amendments were rolled into the base specification in the 1996 version:

IEEE 1003.1B™	IEEE Standard for Information Technology - Portable Operating System Interfaces (POSIX®) - Part 1: System Application Program Interface (API) - Amendment 1: Realtime Extension (C Language)
IEEE 1003.1C™	IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®) – Part 1: System Application Program Interface (API) Amendment 2: Threads Extension (C Language)
IEEE 1003.1D™	IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®)- Part 1: System Application Program Interface (API) - Amendment 4: Additional Realtime Extensions (C Language)
IEEE 1003.1J™	IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®)- Part 1: System Application Program Interface (API) - Amendment 5: Advanced Realtime Extensions (C Language)
IEEE 1003.1Q™	IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®)- Part 1: System Application Program Interface (API) - Amendment 7: Tracing (C Language)
IEEE 1394	A High-Performance Serial Bus

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

Object Management Group (OMG)

Document Number	Document Title
ORMSC/14-06-01	Model-Driven Architecture (MDA), Object Management Group (OMG) Architecture Board

American National Standards Institute (ANSI), International Standards Organization (ISO), and International Electrotechnical Commission (IEC) Joint Technical Committee (JTC) 1 Working Group

Document Number	Document Title
C99RationaleV5.10	Rationale for International Standard—Programming Languages—C
ISO/IEC 9899 (In USA this is: INCITS/ISO/IEC 9899:year)	Information technology—Programming languages—C
ISO 9945	Information technology—Portable Operating System Interface (POSIX®) Base Specifications
ISO/IEC 14882 (In USA this is: INCITS/ISO/IEC 14882:year)	Information technology—Programming languages—C++

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

APPENDIX C

**ACKNOWLEDGEMENTS
(for Original Version)**

Principal Authors

Richard C. Reinhart, Thomas J. Kacpura, Louis M. Handler, Sandra K. Johnson, Janette C. Briones, Jennifer M. Nappier, and Joseph A. Downey Glenn Research Center, Cleveland, OH	C. Steve Hall Analex Corporation, Cleveland, OH James P. Lux Jet Propulsion Laboratory, Pasadena, CA
Dale J. Mortensen ASRC Aerospace Corporation, Cleveland, OH	

Key Industry Participants

Carl Smith, John Liebetreu General Dynamics Corporation, C4-I	Vince Kovarik Harris Corporation, Melbourne, FL
Mark Scoville L-3 Communications Salt Lake City, UT	Jerry Bickle Prism Tech Woburn, MA

Key Reviewers and Contributors

David J. Israel Goddard Space Flight Center, Greenbelt, MD	Allen Farrington, Yong Chong, Kenneth J. Peters Jet Propulsion Laboratory, Pasadena, CA
Andrew L. Benjamin Johnson Space Center, Houston, TX	Eric A. Eberly, Terry M. Luttrell Marshall Space Flight Center, Huntsville, AL

SDR Forum Contributing Member Companies

General Dynamics Prism Tech Boeing Corporation Cincinnati Electronics	Harris Corporation L-3 Communications Lockheed Martin
--	---

NASA-STD-4009A

APPENDIX D

REQUIREMENTS COMPLIANCE MATRIX

E.1 Purpose

This Appendix provides a listing of requirements contained in this NASA Technical Standard for selection and verification of requirements by programs and projects. (*Note: Enter “Yes” to describe the requirement’s applicability to the program or project; or enter “No” if the intent is to tailor, and enter how tailoring is to be applied in the “Rationale” column.*)

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
2.1.1	Applicable Documents, General	[NTS-1] The latest issuances of cited documents shall apply unless specific versions are designated.		
2.1.2	Applicable Documents, General	[NTS-2] Non-use of specifically designated versions shall be approved by the responsible Technical Authority.		
2.4.2	Order of Precedence	[NTS-3] Conflicts between this NASA Technical Standard and other requirements documents shall be resolved by the responsible Technical Authority.		
4. Hardware Architecture				
4.	Hardware Architecture	[STRS-1] An STRS platform shall have a known state after completion of the power-up process.		
4.2.1.4	GPM Requirements	[STRS-2] A module’s diagnostic information shall be available via the STRS APIs.		
4.2.1.4	GPM Requirements	(STRS-109) An STRS platform shall have a GPM that contains and executes the STRS OE and the control portions of the STRS applications and services software.		
4.2.3.4	RFM Requirements	[STRS-6] The STRS platform provider shall describe, in the HID document, the behavior and performance of the RF modular component(s).		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
4.3	Hardware Interface Description	[STRS-4] The STRS platform provider shall describe, in the HID document, the behavior and capability of each major module or component available for use by waveform, service, or other application (e.g., FPGA, GPP, DSP, or memory), noting any operational limitations.		
4.3	Hardware Interface Description	[STRS-5] The STRS platform provider shall describe, in the HID document, the reconfigurability behavior and capability of each reconfigurable component.		
4.3	Hardware Interface Description	[STRS-7] The STRS platform provider shall describe, in the HID document, the interfaces that are provided to and from each modular component of the STRS platform.		
4.3.1	Control and Data Interface	[STRS-8] The STRS platform provider shall describe, in the HID document, the control, telemetry, and data mechanisms of each modular component (i.e., how to program or control each modular component of the platform, and how to use or access each device or software component, noting any proprietary and nonstandard aspects).		
4.3.2	Operating Power Interface	[STRS-9] The STRS platform provider shall describe, in the HID document, the behavior and performance of any power supply or power converter modular component(s).		
4.3.3	Thermal Interface and Power Consumption	[STRS-108] The STRS platform provider shall describe, in the HID document, the thermal and power limits of the hardware at the smallest modular level to which power is controlled.		
5. Applications				
5.1	Application Implementation	[STRS-10] An STRS application shall use the STRS infrastructure-provided APIs and POSIX® API for access to platform resources.		
5.1	Application Implementation	[STRS-11] The STRS infrastructure shall use the STRS platform HAL APIs to communicate with application components on the platform specialized hardware via the physical interface defined by the STRS platform provider.		
5.3	Application Repository Submissions	<p>[STRS-12] The following application or OE development artifacts shall be submitted to the NASA STRS Application Repository:</p> <ul style="list-style-type: none"> (1) Application (or OE) or system component software and configurable hardware design simulation model(s) and/or documentation. (Design Description Document) (2) Documentation of external interfaces for STRS application, devices, or configurable hardware design (e.g., signal names, descriptions, polarity, format, data type, and timing constraints). (HID) (3) Documentation of STRS application or OE behavior and adaptability (e.g., configurable and queryable data items). (Design Description Document, User's Guide) 		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
		<p>(4) Application or OE function sources (e.g., C, C++, header files, very-high-speed integrated circuit HDL (VHDL), and Verilog). (Artifacts)</p> <p>(5) Application or OE libraries, if applicable (e.g., electronic design interchange format (EDIF), dynamic link library (DLL)). (Artifacts)</p> <p>(6) Documentation of application (or OE) development environment and/or tool suite as follows: (Design Description Document)</p> <p style="padding-left: 40px;">A. Include the development environment and/or tool suite name, purpose, developer, version, and configuration specifics (e.g., ISE Design Suite System, Xilinx, 14.4, EDK and SDK; MATLAB®, Model base design support automatic code generation, MathWorks, R2016a)</p> <p style="padding-left: 40px;">B. Include a description of the hardware on which the development environment and/or tool suite is executed, its OS, OS developer, OS version, and OS configuration specifics (e.g., Microsoft® Windows 7, Service pack 2; Linux® Ubuntu, (Xenial Xerus) 16.04)</p> <p style="padding-left: 40px;">C. Include a description of the output of the development environment and/or tool suite, its STRS infrastructure/OE description, developer, version, and unique implementation items (e.g., Type of file, .mdl, .slx; GRC's STRS Reference Implementation; IP generated from Xilinx).</p> <p style="padding-left: 40px;">D. Include a description of licensing agreements for development environment and/or tool suite.</p> <p>(7) Test plans, procedures, and results documentation. (V&V Plan, V&V Procedure, and V&V Results)</p> <p>(8) Identification of software development standards used. (Version Description Document (VDD)/Metadata)</p> <p>(9) Version of this NASA Technical Standard used. (VDD/Metadata)</p> <p>(10) Information, along with supporting documentation, required to make the appropriate decisions regarding ownership, distribution rights, and release (technology transfer) of the application or OE and associated artifacts. (Transfer Rights/Agreements)</p> <p>(11) Version Description Document, if available, or other document containing the version number of each separable artifact in the release, defined down to the lowest level components. (VDD)</p> <p>(12) Documentation of the platform component hardware used by the application or OE, its function and the interconnections. If the component executes an operating system, document the OS, OS developer, OS version, and OS configuration. (HID)</p> <p>(13) Documentation when an OE is submitted to the STRS Application Repository, providing guidelines to aid a waveform/application developer and integrator in the task of developing an STRS compliant</p>		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
		waveform/application. (OE-Specific Developer's Guide)		
6. Configurable Hardware Design Architecture				
6.1	Specialized Hardware Interfaces	[STRS-13] If the STRS application has a component resident outside the GPM (e.g., in configurable hardware design), then the component shall be controllable from the STRS OE.		
6.1	Specialized Hardware Interfaces	<p>[STRS-14] The STRS SPM developer shall provide a platform-specific wrapper for each user-programmable FPGA, which performs the following functions:</p> <ul style="list-style-type: none"> (1) Provides an interface for command and data from the GPM to the waveform application. (2) Provides the platform-specific pinout for the STRS application developer. 		
6.1	Specialized Hardware Interfaces	<p>[STRS-15] The STRS SPM developer shall provide documentation on the configurable hardware design interfaces of the platform-specific wrapper for each user-programmable FPGA, which describes the following:</p> <ul style="list-style-type: none"> (1) Signal names and descriptions. (2) Signal polarity, format, and data type. (3) Signal direction. (4) Signal-timing constraints. (5) Clock generation and synchronization methods. (6) Signal-registering methods. (7) Identification of development tool set used. (8) Any included noninterface functionality. 		
7. Software Architecture				
7.3	STRS APIs	[STRS-105] The STRS infrastructure APIs shall have an ISO/IEC C language compatible interface.		
7.3.1	STRS Application-Provided Application Control API	[STRS-16] The STRS Application-provided Application Control API shall be implemented using ISO/IEC C or C++.		
7.3.1	STRS Application-	[STRS-17] The STRS infrastructure shall use the STRS Application-provided Application Control API to control STRS applications.		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
	Provided Application Control API			
7.3.1	STRS Application- Provided Application Control API	[STRS-18] The STRS OE shall support ISO/IEC C or C++, or both, language interfaces for the STRS Application-provided Application Control API at compile-time.		
7.3.1	STRS Application- Provided Application Control API	[STRS-19] The STRS OE shall support ISO/IEC C or C++, or both, language interfaces for the STRS Application-provided Application Control API at run-time.		
7.3.1	STRS Application- Provided Application Control API	[STRS-20] Each STRS application shall contain #include "STRS_ApplicationControl.h"		
7.3.1	STRS Application- Provided Application Control API	[STRS-21] The STRS platform provider shall provide an "STRS_ApplicationControl.h" that contains the method prototypes for each STRS application and, for C++, the class definition for the base class STRS_ApplicationControl.		
7.3.1	STRS Application- Provided Application Control API	[STRS-22] If the STRS Application-provided Application Control API is implemented in C++, the STRS application class shall be derived from the STRS_ApplicationControl base class.		
7.3.1	STRS Application- Provided Application Control API	[STRS-23] If the STRS application provides the APP_Write method, the STRS application shall contain #include "STRS_Sink.h"		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A				
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale
7.3.1	STRS Application- Provided Application Control API	[STRS-24] The STRS platform provider shall provide an “STRS_Sink.h” that contains the method prototypes for APP_Write and, for C++, the class definition for the base class STRS_Sink.		
7.3.1	STRS Application- Provided Application Control API	[STRS-25] If the STRS Application-provided Application Control API is implemented in C++ and the STRS application provides the APP_Write method, the STRS application class shall be derived from the STRS_Sink base class.		
7.3.1	STRS Application- Provided Application Control API	[STRS-26] If the STRS application provides the APP_Read method, the STRS application shall contain #include "STRS_Source.h"		
7.3.1	STRS Application- Provided Application Control API	[STRS-27] The STRS platform provider shall provide an “STRS_Source.h” that contains the method prototypes for APP_Read and, for C++, the class definition for the base class STRS_Source.		
7.3.1	STRS Application- Provided Application Control API	(STRS-110) The STRS platform provider shall provide an “STRS_APIs.h” that contains the method prototypes for the STRS infrastructure APIs.		
7.3.1	STRS Application- Provided Application Control API	[STRS-28] If the STRS Application-provided Application Control API is implemented in C++ and the STRS application provides the APP_Read method, the STRS application class shall be derived from the STRS_Source base class.		
7.3.1	STRS Application- Provided	(STRS-111) Each STRS Device shall contain #include "STRS_DeviceControl.h"		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
	Application Control API																	
7.3.1	STRS Application-Provided Application Control API	(STRS-112) The STRS platform provider shall provide an “STRS_DeviceControl.h” that contains the method prototypes for each STRS Device and, for C++, the class definition for the base class STRS_DeviceControl, which inherits from the base class STRS_ApplicationControl.																
7.3.1	STRS Application-Provided Application Control API	(STRS-113) If the STRS Device-provided Device Control API is implemented in C++, the STRS Device class shall be derived from the STRS_DeviceControl base class.																
7.3.1	STRS Application-Provided Application Control API	<div>[STRS-29] Each STRS application shall contain a callable APP_Configure method as described in Table 5, APP_Configure().</div> <div><table><tr><th colspan="2">Table 5—APP_Configure()</th></tr><tr><th colspan="2">APP_Configure()</th></tr><tr><td>Description</td><td>Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states. The API is defined in STRS_PropertySet.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value</td></tr><tr><td>Return</td><td>status (STRS_Result) actual size stored unless error</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td></td><td></td></tr></table></div>	Table 5—APP_Configure()		APP_Configure()		Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states. The API is defined in STRS_PropertySet.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value	Return	status (STRS_Result) actual size stored unless error	Precondition	None				
Table 5—APP_Configure()																		
APP_Configure()																		
Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states. The API is defined in STRS_PropertySet.																	
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value																	
Return	status (STRS_Result) actual size stored unless error																	
Precondition	None																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Postcondition	The appropriately named value is configured. When an error is returned, see the logs for more information.		
		Applicable to	Application developer		
7.3.1	STRS Application-Provided Application Control API	(STRS-114) Each STRS application shall contain a callable APP_Destroy method as described in Table 6, APP_Destroy().			
		Table 6—APP_Destroy()			
		APP_Destroy()			
		Description	Call the destructor for the specified target component. APP_Destroy is not a class method.		
		Parameters	<ul style="list-style-type: none">pApp – (STRS_Instance *) pointer to application instance.		
		Return	None		
		Precondition	Application must be stopped and resources released.		
		Postcondition	STRS instance object is no longer valid.		
		Applicable to	Application developer		
7.3.1	STRS Application-Provided Application Control API	(STRS-115) The STRS infrastructure shall define a callable APP_GetHandleID method in each application as described in Table 7, APP_GetHandleID().			
		Table 7—APP_GetHandleID()			
		APP_GetHandleID()			
		Description	Obtain the handle ID for the application, stored by the OE.		
		Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.		
		Return	handle ID of the current application (STRS_HandleID)		
		Precondition	Application is instantiated.		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.1	STRS Application-	(STRS-116) The STRS infrastructure shall define a callable APP_GetHandleName method in each application as described in Table 8, APP_GetHandleName().			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
	Provided Application Control API	<div>Table 8—APP_GetHandleName()</div> <table><tr><td colspan="2">APP_GetHandleName()</td></tr><tr><td>Description</td><td>Obtain the handle name for the application, stored by the OE.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.</td></tr><tr><td>Return</td><td>handle name of the current application (char *)</td></tr><tr><td>Precondition</td><td>Application is instantiated.</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	APP_GetHandleName()		Description	Obtain the handle name for the application, stored by the OE.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.	Return	handle name of the current application (char *)	Precondition	Application is instantiated.	Postcondition	None	Applicable to	OE developer: usually platform provider		
APP_GetHandleName()																		
Description	Obtain the handle name for the application, stored by the OE.																	
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.																	
Return	handle name of the current application (char *)																	
Precondition	Application is instantiated.																	
Postcondition	None																	
Applicable to	OE developer: usually platform provider																	
7.3.1	STRS Application-Provided Application Control API	<div>[STRS-30] Each STRS application shall contain a callable APP_GroundTest method as described in Table 9, APP_GroundTest().</div> <div>Table 9—APP_GroundTest()</div> <table><tr><td colspan="2">APP_GroundTest()</td></tr><tr><td>Description</td><td>Perform unit and system testing, which is usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject. The method is similar to APP_RunTest except that it contains more extensive testing used especially before deployment to satisfy any additional project requirements. This method may be invalid upon deployment and if so, it may be eliminated.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The test is performed. The state is unchanged unless specifically required by mission.</td></tr></table>	APP_GroundTest()		Description	Perform unit and system testing, which is usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject. The method is similar to APP_RunTest except that it contains more extensive testing used especially before deployment to satisfy any additional project requirements. This method may be invalid upon deployment and if so, it may be eliminated.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed	Return	status (STRS_Result)	Precondition	None	Postcondition	The test is performed. The state is unchanged unless specifically required by mission.				
APP_GroundTest()																		
Description	Perform unit and system testing, which is usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject. The method is similar to APP_RunTest except that it contains more extensive testing used especially before deployment to satisfy any additional project requirements. This method may be invalid upon deployment and if so, it may be eliminated.																	
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The test is performed. The state is unchanged unless specifically required by mission.																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Applicable to	Application developer														
7.3.1	STRS Application- Provided Application Control API	[STRS-31] Each STRS application shall contain a callable APP_Initialize method as described in Table 10, APP_Initialize(). Table 10—APP_Initialize() APP_Initialize() <table><tr><td>Description</td><td>Initialize the target component (application, device). The API is defined in STRS_LifeCycle. The purpose is to set or reset the component to a known initial state. If no fault is detected, this method changes the internal state as appropriate.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>Application developer</td></tr></table>		Description	Initialize the target component (application, device). The API is defined in STRS_LifeCycle. The purpose is to set or reset the component to a known initial state. If no fault is detected, this method changes the internal state as appropriate.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.	Return	status (STRS_Result)	Precondition	None	Postcondition	None	Applicable to	Application developer		
Description	Initialize the target component (application, device). The API is defined in STRS_LifeCycle. The purpose is to set or reset the component to a known initial state. If no fault is detected, this method changes the internal state as appropriate.																
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	None																
Applicable to	Application developer																
7.3.1	STRS Application- Provided Application Control API	[STRS-32] Each STRS application shall contain a callable APP_Instance method as described in Table 11, APP_Instance(). Table 11—APP_Instance() APP_Instance() <table><tr><td>Description</td><td>Store the two parameters passed in the calling sequence, so that they are available to the constructor. In C++, APP_Instance is a static method used to call the class constructor for C++. If no fault is detected, this method returns an instance pointer and initializes the internal state.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">id - (in STRS_HandleID) handle ID of this STRS application.name – (in STRS_HandleName *) handle name of this STRS application.</td></tr><tr><td>Return</td><td>Pointer to STRS_Instance, instance of class, in C or C++.</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr></table>		Description	Store the two parameters passed in the calling sequence, so that they are available to the constructor. In C++, APP_Instance is a static method used to call the class constructor for C++. If no fault is detected, this method returns an instance pointer and initializes the internal state.	Parameters	<ul style="list-style-type: none">id - (in STRS_HandleID) handle ID of this STRS application.name – (in STRS_HandleName *) handle name of this STRS application.	Return	Pointer to STRS_Instance, instance of class, in C or C++.	Precondition	None	Postcondition	None				
Description	Store the two parameters passed in the calling sequence, so that they are available to the constructor. In C++, APP_Instance is a static method used to call the class constructor for C++. If no fault is detected, this method returns an instance pointer and initializes the internal state.																
Parameters	<ul style="list-style-type: none">id - (in STRS_HandleID) handle ID of this STRS application.name – (in STRS_HandleName *) handle name of this STRS application.																
Return	Pointer to STRS_Instance, instance of class, in C or C++.																
Precondition	None																
Postcondition	None																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Applicable to	Application developer														
7.3.1	STRS Application- Provided Application Control API	[STRS-33] Each STRS application shall contain a callable APP_Query method as described in Table 12, APP_Query(). <div>Table 12—APP_Query()</div> <div>APP_Query()</div> <table><tr><td>Description</td><td>Obtain values for one or more properties in the target component (application, device). It is the responsibility of the target component to determine which properties can be interrogated in which internal states. The API is defined in STRS_PropertySet.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location to store data corresponding to namelenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value</td></tr><tr><td>Return</td><td>status (STRS_Result) actual size unless error</td></tr><tr><td>Precondition</td><td>The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).</td></tr><tr><td>Postcondition</td><td>Value is populated with data, as appropriate. When an error is returned, see the logs for more information.</td></tr><tr><td>Applicable to</td><td>Application developer</td></tr></table>		Description	Obtain values for one or more properties in the target component (application, device). It is the responsibility of the target component to determine which properties can be interrogated in which internal states. The API is defined in STRS_PropertySet.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location to store data corresponding to namelenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value	Return	status (STRS_Result) actual size unless error	Precondition	The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).	Postcondition	Value is populated with data, as appropriate. When an error is returned, see the logs for more information.	Applicable to	Application developer		
Description	Obtain values for one or more properties in the target component (application, device). It is the responsibility of the target component to determine which properties can be interrogated in which internal states. The API is defined in STRS_PropertySet.																
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location to store data corresponding to namelenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value																
Return	status (STRS_Result) actual size unless error																
Precondition	The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).																
Postcondition	Value is populated with data, as appropriate. When an error is returned, see the logs for more information.																
Applicable to	Application developer																
7.3.1	STRS Application- Provided Application Control API	[STRS-34] If the STRS application provides data to the infrastructure, then the STRS application shall contain a callable APP_Read method as described in Table 13, APP_Read(). <div>Table 13—APP_Read()</div> <div>APP_Read()</div> <table><tr><td>Description</td><td>Method used to obtain data from the target component (application, device). This is optional. The API is defined in STRS_Source. The caller manages the</td></tr></table>		Description	Method used to obtain data from the target component (application, device). This is optional. The API is defined in STRS_Source. The caller manages the												
Description	Method used to obtain data from the target component (application, device). This is optional. The API is defined in STRS_Source. The caller manages the																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
			buffer area, preallocating the buffer before calling APP_Read and processing the returned data without any effects on the data source application. The character data type (STRS_Message) does not have to contain valid characters. If the application is not in the appropriate internal state, then nothing is done and an error is returned.		
		Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementationbuffer - (out STRS_Message) a pointer to an area in which the application stores the requested datanb - (in STRS_Buffer_Size) number of bytes requested		
		Return	Error status (negative) or actual number of bytes (non-negative) obtained (STRS_Result)		
		Precondition	Storage for the buffer with space for nb bytes is allocated before calling APP_Read. If used for a C-style character string, the size should include space for a final '\0'.		
		Postcondition	The data from the application is stored in the buffer area.		
		Applicable to	Application developer		
7.3.1	STRS Application-Provided Application Control API	[STRS-35] Each STRS application shall contain a callable APP_ReleaseObject method as described in Table 14, APP_ReleaseObject(). Table 14—APP_ReleaseObject() APP_ReleaseObject()			
		Description	Free any resources that the target component (application, device) has acquired. An example would be to allow the target component to close any open files or devices. It is the responsibility of the target component to determine whether any release is done in which internal states. The API is defined in STRS_LifeCycle. The purpose of APP_ReleaseObject is to prepare the target component for removal.		
		Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.		
		Return	status (STRS_Result)		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Precondition	None														
		Postcondition	All resources acquired by the target component are released. The application may not be usable unless reinstantiated or reinitialized.														
		Applicable to	Application developer														
7.3.1	STRS Application-Provided Application Control API	[STRS-36] Each STRS application shall contain a callable APP_RunTest method as described in Table 15, APP_RunTest(). Table 15—APP_RunTest() APP_RunTest() <table><tr><td>Description</td><td>Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the application. Application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed. Values of testID are mission dependent.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The test is performed.</td></tr><tr><td>Applicable to</td><td>Application developer</td></tr></table>		Description	Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the application. Application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed. Values of testID are mission dependent.	Return	status (STRS_Result)	Precondition	None	Postcondition	The test is performed.	Applicable to	Application developer		
Description	Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the application. Application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. The API is defined in STRS_TestableObject.																
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.testID - (in STRS_TestID) number of the test to be performed. Values of testID are mission dependent.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	The test is performed.																
Applicable to	Application developer																
7.3.1	STRS Application-Provided Application Control API	[STRS-37] Each STRS application shall contain a callable APP_Start method as described in Table 16, APP_Start(). Table 16—APP_Start() APP_Start() <table><tr><td>Description</td><td>Begin normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent.</td></tr></table>		Description	Begin normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent.												
Description	Begin normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale	
		Parameters	• inst – (STRS_Instance *) instance pointer, only for C implementation.			
		Return	status (STRS_Result)			
		Precondition	None			
		Postcondition	None			
		Applicable to	Application developer			
7.3.1	STRS Application- Provided Application Control API	[STRS-38] Each STRS application shall contain a callable APP_Stop method as described in Table 17, APP_Stop(). Table 17—APP_Stop() APP_Stop() Description End normal target component (application, device) processing. If the application is not in the appropriate internal state, then nothing is done and an error is returned. The API is defined in STRS_ControllableComponent. Parameters • inst – (STRS_Instance *) instance pointer, only for C implementation. Return status (STRS_Result) Precondition None Postcondition None Applicable to Application developer				
7.3.1	STRS Application- Provided Application Control API	[STRS-39] If the STRS application receives data from the infrastructure, then the STRS application shall contain a callable APP_Write method as described in Table 18, APP_Write(). Table 18—APP_Write() APP_Write() Description Method used to send data to the target component (application, device). This is optional. The API is defined in STRS_Sink. The caller manages the buffer area, preallocating and filling the buffer before calling APP_Write. The character data type (STRS_Message) does not have to contain valid characters. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Parameters • inst – (STRS_Instance *) instance pointer, only for C implementation.				

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A														
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale									
			<ul style="list-style-type: none">buffer - (in STRS_Message) pointer to the data for the application to processnb - (in STRS_Buffer_Size) number of bytes in buffer											
		Return	Error status (negative) or number of bytes (non-negative) written (STRS_Result)											
		Precondition	Storage for the buffer with space for nb bytes is allocated before calling APP_Write. If used for a C-style character string, the size should include space for a final '\0'.											
		Postcondition	The data has been captured by the application for its processing.											
		Applicable to	Application developer											
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-40] The STRS infrastructure shall contain a callable STRS_Configure method as described in Table 19, STRS_Configure(). Table 19—STRS_Configure() STRS_Configure() <table><tr><td>Description</td><td>Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value</td></tr><tr><td>Return</td><td>status (STRS_Result) actual size stored unless error</td></tr><tr><td>Precondition</td><td>None</td></tr></table>			Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value	Return	status (STRS_Result) actual size stored unless error	Precondition	None		
Description	Set value for one property in the target component (application, device). It is the responsibility of the target component to determine which properties can be changed in which internal states.													
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location of data to process/store in application corresponding to namelenValue – (in STRS_Buffer_Size) actual length of data in value													
Return	status (STRS_Result) actual size stored unless error													
Precondition	None													

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Postcondition	The appropriate named value is configured. When an error is returned, see the logs for more information.														
		Applicable to	OE developer: usually platform provider														
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-41] The STRS infrastructure shall contain a callable STRS_GroundTest method as described in Table 20, STRS_GroundTest(). Table 20—STRS_GroundTest() STRS_GroundTest() <table><tr><td>Description</td><td>Perform unit and system testing—usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. This method provides more exhaustive testing to satisfy any additional project requirements. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. This method may be invalid upon deployment.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.testID - (in STRS_TestID) number of the test to be performed. Values are mission dependent.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The test is performed.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>		Description	Perform unit and system testing—usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. This method provides more exhaustive testing to satisfy any additional project requirements. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. This method may be invalid upon deployment.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.testID - (in STRS_TestID) number of the test to be performed. Values are mission dependent.	Return	status (STRS_Result)	Precondition	None	Postcondition	The test is performed.	Applicable to	OE developer: usually platform provider		
Description	Perform unit and system testing—usually done before deployment. The testing may include calibration. The tests aid in isolating faults within the target component. This method provides more exhaustive testing to satisfy any additional project requirements. A responding component may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed. This method may be invalid upon deployment.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.testID - (in STRS_TestID) number of the test to be performed. Values are mission dependent.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	The test is performed.																
Applicable to	OE developer: usually platform provider																
7.3.2	STRS Infrastructure-Provided Application	[STRS-42] The STRS infrastructure shall contain a callable STRS_Initialize method as described in Table 21, STRS_Initialize().															

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale	
	Control API	Table 21—STRS_Initialize()				
		STRS_Initialize()				
		Description	Initialize the target component (application, device). The purpose is to set or reset the component to a known initial state.			
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.			
		Return	status (STRS_Result)			
		Precondition	None			
		Postcondition	None			
		Applicable to	OE developer: usually platform provider			
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-43] The STRS infrastructure shall contain a callable STRS_Query method as described in Table 22, STRS_Query().				
Table 22—STRS_Query()						
STRS_Query()						
Description	Obtain values for one or more properties in the target component (application, device).					
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.name - (in STRS_Property_Name) name or other identification of data to be obtainedvalue - (in STRS_Property_Value *) location to store data corresponding to namelenValue – (in STRS_Buffer_Size) maximum length of data in to be stored in value					

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Return	status (STRS_Result) actual size unless error		
		Precondition	The value is to have space allotted for the maximum size of the property whose value is to be returned (not bigger than lenValue).		
		Postcondition	Value is populated with data as appropriate. When an error is returned, see the logs for more information.		
		Applicable to	OE developer: usually platform provider		
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-44] The STRS infrastructure shall contain a callable STRS_ReleaseObject method as described in Table 23, STRS_ReleaseObject(). Table 23—STRS_ReleaseObject() STRS_ReleaseObject() Description Free any resources that the target component (application, device) has acquired. An example would be to allow the target component to close any open files or devices. It is the responsibility of the target component to determine whether any release is done in which internal states. The purpose of STRS_ReleaseObject is to prepare the target component for removal. Parameters <ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request. Return status (STRS_Result) Precondition None Postcondition All resources acquired by the target component are released. The target component may not be usable unless reinstantiated or reinitialized. Applicable to OE developer: usually platform provider			
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-45] The STRS infrastructure shall contain a callable STRS_RunTest method as described in Table 24, STRS_RunTest().			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Table 24—STRS_RunTest()			
		STRS_RunTest()			
		Description	Test specific functionality within the target component (application, device). The tests provide aid in isolating faults within the target component. A responding application may be in any internal state, but certain tests may be restricted to specific states. If the application is not in the appropriate internal state, then nothing is done and an error is returned. Property values may be used, if needed.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.testID - (in STRS_TestID) number of the test to be performed. Values of testID are mission-dependent.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	The test is performed.		
		Applicable to	OE developer: usually platform provider		
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-46] The STRS infrastructure shall contain a callable STRS_Start method as described in Table 25, STRS_Start().			
Table 25—STRS_Start()					
STRS_Start()					
Description	Begin normal target component (application, device) processing. Nothing is done if the application (or device) is not in the appropriate internal state.				
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request.				
Return	status (STRS_Result)				

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.2	STRS Infrastructure-Provided Application Control API	[STRS-47] The STRS infrastructure shall contain a callable STRS_Stop method as described in Table 26, STRS_Stop().			
		Table 26—STRS_Stop()			
		STRS_Stop()			
		Description	End target component (application, device) processing. Nothing is done unless the application (or device) is in the appropriate internal state.		
		Parameters	<ul style="list-style-type: none">fromWF - in STRS_HandleID) handle ID of current component making the request.toWF - in STRS_HandleID) handle ID of target component that should respond to the request.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	[STRS-48] The STRS infrastructure shall contain a callable STRS_AbortApp method as described in Table 27, STRS_AbortApp().			
		Table 27—STRS_AbortApp()			
		STRS_AbortApp()			
		Description	Abort an application or service.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toWF - (in STRS_HandleID) handle ID of target component that should respond to the request		
		Return	Status (STRS_Result)		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Precondition	None		
		Postcondition	The target component is aborted, and application is stopped, resources released, and unloaded, if allowed by OE.		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	[STRS-49] The STRS infrastructure shall contain a callable STRS_GetErrorQueue method as described in Table 28, STRS_GetErrorQueue().			
		Table 28—STRS_GetErrorQueue()			
		STRS_GetErrorQueue()			
		Description	Transform an error status into an error queue.		
		Parameters	<ul style="list-style-type: none">result - (in STRS_Result) return value of previous call.		
		Return	Handle ID (STRS_HandleID) corresponding to invalid STRS_Result; that is, return STRS_ERROR_QUEUE for STRS_ERROR, STRS_WARNING_QUEUE for STRS_WARNING, and STRS_FATAL_QUEUE for STRS_FATAL; otherwise, implementation defined.		
		Precondition	None		
		Postcondition	The corresponding error queue handle ID is returned.		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	(STRS-117) The STRS infrastructure shall contain a callable STRS_GetHandleName method as described in Table 29, STRS_GetHandleName().			
		Table 29—STRS_GetHandleName()			
		STRS_GetHandleName()			
		Description	The handle name is obtained for the given handle ID. The handle ID of the current component (fromWF) is used for any error message. Using STRS_GetHandleName to determine the handle name of the current component while it is being instantiated gives undefined results.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
			<ul style="list-style-type: none">toID - (in STRS_HandleID) handle ID of the resource (application, device, file, queue) for which the handle name is to be obtained.toResourceName - (out char *) handle name of the desired resource.														
		Return	status (STRS_Result)														
		Precondition	Space must be allocated for handle name.														
		Postcondition	Handle name is filled in. On error, the first character of the handle name is filled with a zero unless the toResourceName variable is NULL.														
		Applicable to	OE developer: usually platform provider														
7.3.3	STRS Infrastructure Application Setup API	[STRS-50] The STRS infrastructure shall contain a callable STRS_HandleRequest method as described in Table 30, STRS_HandleRequest(). <div>Table 30—STRS_HandleRequest() STRS_HandleRequest() <table><tr><td>Description</td><td>The handle ID is obtained for the given handle name. The handle ID of the current component (fromWF) is used for any error message. Using STRS_HandleRequest to determine the handle ID of the current component while it is being instantiated gives undefined results.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toResourceName - (in char *) name of desired resource (application, device, file, queue).</td></tr><tr><td>Return</td><td>Handle ID of the entity or error status. (STRS_HandleID)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table></div>		Description	The handle ID is obtained for the given handle name. The handle ID of the current component (fromWF) is used for any error message. Using STRS_HandleRequest to determine the handle ID of the current component while it is being instantiated gives undefined results.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toResourceName - (in char *) name of desired resource (application, device, file, queue).	Return	Handle ID of the entity or error status. (STRS_HandleID)	Precondition	None	Postcondition	None	Applicable to	OE developer: usually platform provider		
Description	The handle ID is obtained for the given handle name. The handle ID of the current component (fromWF) is used for any error message. Using STRS_HandleRequest to determine the handle ID of the current component while it is being instantiated gives undefined results.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toResourceName - (in char *) name of desired resource (application, device, file, queue).																
Return	Handle ID of the entity or error status. (STRS_HandleID)																
Precondition	None																
Postcondition	None																
Applicable to	OE developer: usually platform provider																
7.3.3	STRS Infrastructure Application Setup API	[STRS-51] The STRS infrastructure shall contain a callable STRS_InstantiateApp method as described in Table 31, STRS_InstantiateApp().															

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Table 31—STRS_InstantiateApp()			
		STRS_InstantiateApp()			
		Description	Instantiate an application, service, or device. The handle name specified for the application, service, or device is to be unique. The OE-specific name is used to identify the application for instantiation and may impose additional operations to be performed as documented by the platform provider. It is up to the OE to determine whether any resources are to be loaded to accomplish the instantiation.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.handleName - (in char *) unique handle name for the application (or device) that should be instantiated.startName - (in char *) OE-specific name used to instantiate and configure the application (or device) into a known state.		
		Return	handle ID (STRS_HandleID) of the application (or device) instantiated or the error status		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	[STRS-52] The STRS infrastructure shall contain a callable STRS_IsOK method as described in Table 32, STRS_IsOK().			
Table 32—STRS_IsOK()					
STRS_IsOK()					
Description	Return true, if return value of argument obtained from previous call is not an error status.				
Parameters	<ul style="list-style-type: none">result - (in STRS_Result) return value of previous call.				
Return	true, if STRS_Result is not STRS_WARNING, STRS_ERROR, or STRS_FATAL: that is, non-negative (bool)				
	Precondition	Previous call returns a status result.			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																				
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale															
		Postcondition	None																	
		Applicable to	OE developer: usually platform provider																	
7.3.3	STRS Infrastructure Application Setup API	[STRS-53] The STRS infrastructure shall contain a callable STRS_Log method as described in Table 33, STRS_Log(). Table 33—STRS_Log() <table><tr><th colspan="2">STRS_Log()</th></tr><tr><td>Description</td><td>Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Log message is distributed.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>			STRS_Log()		Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer	Return	status (STRS_Result)	Precondition	None	Postcondition	Log message is distributed.	Applicable to	OE developer: usually platform provider		
STRS_Log()																				
Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.																			
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer																			
Return	status (STRS_Result)																			
Precondition	None																			
Postcondition	Log message is distributed.																			
Applicable to	OE developer: usually platform provider																			
7.3.3	STRS Infrastructure Application Setup API	[STRS-54] When an STRS application has a nonfatal error, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_ERROR_QUEUE.																		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
		<div>Table 33—STRS_Log()</div> <table><tr><td colspan="2">STRS_Log()</td></tr><tr><td>Description</td><td>Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Log message is distributed.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_Log()		Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer	Return	status (STRS_Result)	Precondition	None	Postcondition	Log message is distributed.	Applicable to	OE developer: usually platform provider		
STRS_Log()																		
Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	Log message is distributed.																	
Applicable to	OE developer: usually platform provider																	
7.3.3	STRS Infrastructure Application Setup API	<div>[STRS-55] When an STRS application has a fatal error, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_FATAL_QUEUE.</div> <div>Table 33—STRS_Log()</div> <table><tr><td colspan="2">STRS_Log()</td></tr><tr><td>Description</td><td>Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE,</td></tr></table>	STRS_Log()		Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE,												
STRS_Log()																		
Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE,																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A									
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale				
			STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.						
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer						
		Return	status (STRS_Result)						
		Precondition	None						
		Postcondition	Log message is distributed.						
		Applicable to	OE developer: usually platform provider						
7.3.3	STRS Infrastructure Application Setup API	[STRS-56] When an STRS application has a warning condition, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_WARNING_QUEUE. <div>Table 33—STRS_Log()</div> <div>STRS_Log()</div> <table><tr><td>Description</td><td>Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE,</td></tr></table>		Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE,		
Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.								
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE,								

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
			STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors. <ul style="list-style-type: none">msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	Log message is distributed.		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	[STRS-57] When an STRS application needs to send telemetry, the STRS application shall use the callable STRS_Log method as described in Table 33, STRS_Log(), with a target handle ID of constant STRS_TELEMETRY_QUEUE.			
		Table 33—STRS_Log()			
		STRS_Log()			
		Description	Send log message for distribution as appropriate. The time stamp and an indication of the from and target handles are added automatically. STRS_Log may be used to inform the infrastructure that the STRS component is in the FAULT state when a target handle ID of STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE is used. The character data type (STRS_Message) does not have to contain valid characters.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.logTarget - (in STRS_HandleID) handle ID of target (e.g., STRS_TELEMETRY_QUEUE, STRS_ERROR_QUEUE, STRS_WARNING_QUEUE, or STRS_FATAL_QUEUE). The last three special-purpose handle IDs may be used to log errors.msg - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer		
		Return	status (STRS_Result)		
		Precondition	None		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Postcondition	Log message is distributed.		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	(STRS-118) The STRS infrastructure shall contain a callable STRS_ValidateHandleID method as described in Table 34, STRS_ValidateHandleID().			
		Table 34—STRS_ValidateHandleID()			
		STRS_ValidateHandleID()			
		Description	Determines if a handle ID is STRS_OK or in error. After calling any STRS method that returns a handle ID, it is recommended that STRS_ValidateHandleID be called before any other STRS method.		
		Parameters	<ul style="list-style-type: none">tstID - (in STRS_HandleID) the STRS_HandleID object from which the handle ID is extracted.		
		Return	(STRS_Result) STRS_OK when successful; otherwise, for error, STRS_WARNING, STRS_ERROR, or STRS_FATAL.		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.3	STRS Infrastructure Application Setup API	(STRS-119) The STRS infrastructure shall contain a callable STRS_ValidateSize method as described in Table 35, STRS_ValidateSize().			
		Table 35—STRS_ValidateSize()			
		STRS_ValidateSize()			
		Description	Determines if a STRS_File_Size is STRS_OK or in error. STRS_FileGetFreeSpace and STRS_FileGetSize return a type STRS_File_Size number. After calling any STRS method that returns an STRS_File_Size, it is recommended that STRS_ValidateSize be called before calling any other STRS method.		
		Parameters	<ul style="list-style-type: none">tstSize - (in STRS_File_Size) the file size object from which the file size is extracted.		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Return	(STRS_Result) STRS_OK when successful; otherwise, for error, STRS_WARNING, STRS_ERROR, or STRS_FATAL.		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.4	STRS Infrastructure Data Sink	[STRS-58] The STRS infrastructure shall contain a callable STRS_Write method as described in Table 36, STRS_Write().			
		Table 36—STRS_Write()			
		STRS_Write()			
		Description	Method used to send data to a target component (application, device, file, or queue) acting as a sink. The caller manages the buffer area, preallocating and filling the buffer before calling STRS_Write. The character data type (STRS_Message) does not have to contain valid characters.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toID - (in STRS_HandleID) handle ID of target component that should respond to the request and that implemented STRS_Sink.buffer - (in STRS_Message) a pointer to the data to processnb - (in STRS_Buffer_Size) number of bytes in buffer		
		Return	Error status (negative) or number of bytes (non-negative) written (STRS_Result)		
		Precondition	Storage for the buffer is allocated before calling STRS_Write having space for at least nb bytes. If used for a C-style character string, the size should include space for a final '\0'.		
		Postcondition	The data has been captured by the target component for its processing.		
		Applicable to	OE developer: usually platform provider		
7.3.5	STRS Infrastructure Data Source	[STRS-59] The STRS infrastructure shall contain a callable STRS_Read method as described in Table 37, STRS_Read().			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Table 37—STRS_Read()			
		STRS_Read()			
		Description	Method used to obtain data from a target component (application, device, file, or queue) acting as a source or supplier. The caller manages the buffer area, preallocating the buffer before calling STRS_Read and processing the returned data without any effects on the data source application. The character data type (STRS_Message) does not have to contain valid characters.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.pullID - (in STRS_HandleID) handle ID of target component that should respond to the request and that implemented STRS_Source.buffer - (out STRS_Message) a pointer to an area in which to store the data requestednb - (in STRS_Buffer_Size) number of bytes requested		
		Return	Error status (negative) or actual number of bytes (non-negative) obtained (STRS_Result)		
		Precondition	Storage for the buffer is allocated before calling STRS_Read, having space for at least nb bytes. If used for a C-style character string, the size should include space for a final '\0'.		
		Postcondition	The data from the target component is stored in the buffer area.		
		Applicable to	OE developer: usually platform provider		
7.3.6	STRS Infrastructure-Provided Device Control API	[STRS-61] The STRS infrastructure shall contain a callable STRS_DeviceClose method as described in Table 38, STRS_DeviceClose().			
Table 38—STRS_DeviceClose()					
STRS_DeviceClose()					
Description	Close the open device.				
	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
			<ul style="list-style-type: none">toDev - (in STRS_HandleID) handle ID of device that should respond to the request.														
		Return	status (STRS_Result)														
		Precondition	None														
		Postcondition	The device is closed.														
		Applicable to	OE developer: usually platform provider														
7.3.6	STRS Infrastructure-Provided Device Control API	[STRS-62] The STRS infrastructure shall contain a callable STRS_DeviceFlush method as described in Table 39, STRS_DeviceFlush(). Table 39—STRS_DeviceFlush() STRS_DeviceFlush() <table><tr><td>Description</td><td>Used the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>		Description	Used the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.	Return	status (STRS_Result)	Precondition	None	Postcondition	None	Applicable to	OE developer: usually platform provider		
Description	Used the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	None																
Applicable to	OE developer: usually platform provider																
7.3.6	STRS Infrastructure-Provided Device Control API	[STRS-63] The STRS infrastructure shall contain a callable STRS_DeviceLoad method as described in Table 40, STRS_DeviceLoad(). Table 40—STRS_DeviceLoad() STRS_DeviceLoad() <table><tr><td>Description</td><td>Load a binary image to the open device.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.</td></tr></table>		Description	Load a binary image to the open device.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.										
Description	Load a binary image to the open device.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
			<ul style="list-style-type: none">toDev - (in STRS_HandleID) handle ID of device that should respond to the request.fileName - (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.														
		Return	status (STRS_Result)														
		Precondition	None														
		Postcondition	The binary image is stored in the target device.														
		Applicable to	OE developer: usually platform provider														
7.3.6	STRS Infrastructure-Provided Device Control API	[STRS-64] The STRS infrastructure shall contain a callable STRS_DeviceOpen method as described in Table 41, STRS_DeviceOpen(). Table 41—STRS_DeviceOpen() STRS_DeviceOpen() <table><tr><td>Description</td><td>Open the device.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The device is opened.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>		Description	Open the device.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.	Return	status (STRS_Result)	Precondition	None	Postcondition	The device is opened.	Applicable to	OE developer: usually platform provider		
Description	Open the device.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	The device is opened.																
Applicable to	OE developer: usually platform provider																
7.3.6	STRS Infrastructure-Provided Device Control API	[STRS-65] The STRS infrastructure shall contain a callable STRS_DeviceReset method as described in Table 42, STRS_DeviceReset().															

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Table 42—STRS_DeviceReset()			
		STRS_DeviceReset()			
		Description	Reinitialize the device, if possible. Reset is normally used after the corresponding device has been started and stopped, and before the device is started again to bring the hardware device to its power-on state.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.6	STRS Infrastructure-Provided Device Control API	(STRS-68) The STRS infrastructure shall contain a callable STRS_DeviceUnload method as described in Table 43, STRS_DeviceUnload().			
		Table 43—STRS_DeviceUnload()			
		STRS_DeviceUnload()			
		Description	Unload the open device.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	The device is unloaded.		
		Applicable to	OE developer: usually platform provider		
7.3.6	STRS Infrastructure-	(STRS-69) The STRS infrastructure shall contain a callable STRS_SetISR method as described in Table 44, STRS_SetISR().			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
	Provided Device Control API	<div>Table 44—STRS_SetISR()</div> <table><tr><td colspan="2">STRS_SetISR()</td></tr><tr><td>Description</td><td>Set the Interrupt Service Routine for the device.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of the current component making the request.toDev - (in STRS_HandleID) handle ID of the device that should respond to the request.pfun – (in STRS_ISR_Function) function pointer to a static or non-class function to be called to service the interrupt</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>ISR function is activated.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_SetISR()		Description	Set the Interrupt Service Routine for the device.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of the current component making the request.toDev - (in STRS_HandleID) handle ID of the device that should respond to the request.pfun – (in STRS_ISR_Function) function pointer to a static or non-class function to be called to service the interrupt	Return	status (STRS_Result)	Precondition	None	Postcondition	ISR function is activated.	Applicable to	OE developer: usually platform provider		
STRS_SetISR()																		
Description	Set the Interrupt Service Routine for the device.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of the current component making the request.toDev - (in STRS_HandleID) handle ID of the device that should respond to the request.pfun – (in STRS_ISR_Function) function pointer to a static or non-class function to be called to service the interrupt																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	ISR function is activated.																	
Applicable to	OE developer: usually platform provider																	
7.3.7	STRS Device-Provided Device Control API	<p>(STRS-120) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Close method as described in Table 45, DEV_Close().</p> <div>Table 45—DEV_Close()</div> <table><tr><td colspan="2">DEV_Close()</td></tr><tr><td>Description</td><td>Close the open device.</td></tr><tr><td>Parameters</td><td>inst – (STRS_Instance *) instance pointer, only for C implementation.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The device is closed.</td></tr><tr><td>Applicable to</td><td>Device developer: usually platform provider</td></tr></table>	DEV_Close()		Description	Close the open device.	Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.	Return	status (STRS_Result)	Precondition	None	Postcondition	The device is closed.	Applicable to	Device developer: usually platform provider		
DEV_Close()																		
Description	Close the open device.																	
Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The device is closed.																	
Applicable to	Device developer: usually platform provider																	
7.3.7	STRS Device-Provided Device Control API	<p>(STRS-121) If the hardware is to be flushed by the STRS Device, the STRS Device shall contain a callable DEV_Flush method as described in Table 46, DEV_Flush().</p>																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
		<div>Table 46—DEV_Flush()</div> <table><tr><td colspan="2">DEV_Flush()</td></tr><tr><td>Description</td><td>Use the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.</td></tr><tr><td>Parameters</td><td>inst – (STRS_Instance *) instance pointer, only for C implementation.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The device’s buffered data is flushed.</td></tr><tr><td>Applicable to</td><td>Device developer: usually platform provider</td></tr></table>	DEV_Flush()		Description	Use the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.	Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.	Return	status (STRS_Result)	Precondition	None	Postcondition	The device’s buffered data is flushed.	Applicable to	Device developer: usually platform provider		
DEV_Flush()																		
Description	Use the opened device to send any buffered data immediately to the underlying hardware and clear the buffers.																	
Parameters	inst – (STRS_Instance *) instance pointer, only for C implementation.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The device’s buffered data is flushed.																	
Applicable to	Device developer: usually platform provider																	
7.3.7	STRS Device- Provided Device Control API	<div>(STRS-122) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Load method as described in Table 47, DEV_Load().</div> <div>Table 47—DEV_Load()</div> <table><tr><td colspan="2">DEV_Load()</td></tr><tr><td>Description</td><td>Load a binary image to the open device.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.fileName - (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The binary image is stored in the target device.</td></tr><tr><td>Applicable to</td><td>Device developer: usually platform provider</td></tr></table>	DEV_Load()		Description	Load a binary image to the open device.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.fileName - (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.	Return	status (STRS_Result)	Precondition	None	Postcondition	The binary image is stored in the target device.	Applicable to	Device developer: usually platform provider		
DEV_Load()																		
Description	Load a binary image to the open device.																	
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.fileName - (in char *) storage area name or fully qualified file name of the binary image to load onto the hardware device.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The binary image is stored in the target device.																	
Applicable to	Device developer: usually platform provider																	
7.3.7	STRS Device- Provided Device Control API	<div>(STRS-123) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Open method as described in Table 48, DEV_Open().</div> <div>Table 48—DEV_Open()</div> <table><tr><td colspan="2">DEV_Open()</td></tr><tr><td>Description</td><td>Open the device.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.</td></tr></table>	DEV_Open()		Description	Open the device.	Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.										
DEV_Open()																		
Description	Open the device.																	
Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	The device is opened.		
		Applicable to	Device developer: usually platform provider		
7.3.7	STRS Device- Provided Device Control API	(STRS-124) If the hardware is to be reset by the STRS Device, the STRS Device shall contain a callable DEV_Reset method as described in Table 49, DEV_Reset().			
		Table 49—DEV_Reset()			
		DEV_Reset()			
		Description	Reinitialize the device, if possible. Reset is normally used after the corresponding device has been started and stopped, and before the device is started again to bring the hardware device to its power-on state.		
		Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	The device is reset to an initial state.		
		Applicable to	Device developer: usually platform provider		
7.3.7	STRS Device- Provided Device Control API	(STRS-125) If the hardware is to be loaded by the STRS Device, the STRS Device shall contain a callable DEV_Unload method as described in Table 50, DEV_Unload().			
		Table 50—DEV_Unload()			
		DEV_Unload()			
		Description	Unload the open device.		
		Parameters	<ul style="list-style-type: none">inst – (STRS_Instance *) instance pointer, only for C implementation.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	The device is unloaded.		
		Applicable to	Device developer: usually platform provider		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
7.3.8	STRS Infrastructure File Control API	<p>(STRS-70) The STRS infrastructure shall contain a callable STRS_FileClose method as described in Table 51, STRS_FileClose().</p> <p style="text-align: center;">Table 51—STRS_FileClose()</p> <table><tr><th colspan="2">STRS_FileClose()</th></tr><tr><td>Description</td><td>Close the open file. STRS_FileClose is used to close a file that has been opened by STRS_FileOpen.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toFile - (in STRS_HandleID) handle ID of file to be closed.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The file is closed and the handle ID is released.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_FileClose()		Description	Close the open file. STRS_FileClose is used to close a file that has been opened by STRS_FileOpen.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toFile - (in STRS_HandleID) handle ID of file to be closed.	Return	status (STRS_Result)	Precondition	None	Postcondition	The file is closed and the handle ID is released.	Applicable to	OE developer: usually platform provider		
STRS_FileClose()																		
Description	Close the open file. STRS_FileClose is used to close a file that has been opened by STRS_FileOpen.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toFile - (in STRS_HandleID) handle ID of file to be closed.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The file is closed and the handle ID is released.																	
Applicable to	OE developer: usually platform provider																	
7.3.8	STRS Infrastructure File Control API	<p>(STRS-71) The STRS infrastructure shall contain a callable STRS_FileGetFreeSpace method as described in Table 52, STRS_FileGetFreeSpace().</p> <p style="text-align: center;">Table 52—STRS_FileGetFreeSpace()</p> <table><tr><th colspan="2">STRS_FileGetFreeSpace()</th></tr><tr><td>Description</td><td>Get total size of free space available for file storage.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.fileSystem - (in char *) used when more than one file system exists.</td></tr><tr><td>Return</td><td>Total size in bytes (STRS_File_Size).</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_FileGetFreeSpace()		Description	Get total size of free space available for file storage.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.fileSystem - (in char *) used when more than one file system exists.	Return	Total size in bytes (STRS_File_Size).	Precondition	None	Postcondition	None	Applicable to	OE developer: usually platform provider		
STRS_FileGetFreeSpace()																		
Description	Get total size of free space available for file storage.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.fileSystem - (in char *) used when more than one file system exists.																	
Return	Total size in bytes (STRS_File_Size).																	
Precondition	None																	
Postcondition	None																	
Applicable to	OE developer: usually platform provider																	
7.3.8	STRS Infrastructure File Control	<p>(STRS-72) The STRS infrastructure shall contain a callable STRS_FileGetSize method as described in Table 53, STRS_FileGetSize().</p>																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale	
	API	Table 53—STRS_FileGetSize()				
		STRS_FileGetSize()				
		Description	Get the size of the specified file.			
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.fileName - (in char *) storage area name or fully qualified file name of the file for which the size is obtained.			
		Return	File size in bytes (STRS_File_Size).			
		Precondition	None			
		Postcondition	None			
		Applicable to	OE developer: usually platform provider			
7.3.8	STRS Infrastructure File Control API	(STRS-73) The STRS infrastructure shall contain a callable STRS_FileGetStreamPointer method as described in Table 54, STRS_FileGetStreamPointer().				
Table 54—STRS_FileGetStreamPointer()						
STRS_FileGetStreamPointer()						
Description	Get the file stream pointer for the open file associated with the STRS handle ID. This is normally not used because either the common functions are built into the STRS architecture or the entire file manipulation is local to one application or device. This method may be needed for certain file operations not built into the STRS architecture and distributed over more than one application or device or the STRS infrastructure. For example, the file stream pointer may be used when multiple applications write to the same file using a queue or need features not found in STRS_Write. Having a file system is optional; if no file system is present, NULL will be returned. A NULL will also be returned if another error condition is detected.					
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toFile - (in STRS_HandleID) file handle ID.					

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Return	File stream pointer (FILE *) or NULL for error condition.		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.8	STRS Infrastructure File Control API	(STRS-74) The STRS infrastructure shall contain a callable STRS_FileOpen method as described in Table 55, STRS_FileOpen().			
		Table 55—STRS_FileOpen()			
		STRS_FileOpen()			
		Description	Open the file. This method is used to obtain an STRS handle ID when the file manipulation is either built into the STRS architecture or distributed over more than one application or device or the STRS infrastructure.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.filename - (in char *) file name of the file to be opened.file access - (in STRS_Access) indicates if file is to be opened for reading, writing, both, or appending.file type - (in STRS_Type) indicator whether file is text or binary.		
		Return	a handle ID used to read or write data from or to the file (STRS_HandleID). Handle ID should be validated with STRS_ValidateHandleID to determine if successful.		
		Precondition	None		
		Postcondition	The file is open unless an error occurs. On error, the return value should contain an error indication that can be tested by STRS_ValidateHandleID.		
		Applicable to	OE developer: usually platform provider		
7.3.8	STRS Infrastructure File Control API	(STRS-75) The STRS infrastructure shall contain a callable STRS_FileRemove method as described in Table 56, STRS_FileRemove().			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
		<p style="text-align: center;">Table 56—STRS_FileRemove()</p> <table><tr><td colspan="2">STRS_FileRemove()</td></tr><tr><td>Description</td><td>Remove the closed file.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) name of file to be removed.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The file is no longer available, and the space where it was stored becomes available.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_FileRemove()		Description	Remove the closed file.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) name of file to be removed.	Return	status (STRS_Result)	Precondition	None	Postcondition	The file is no longer available, and the space where it was stored becomes available.	Applicable to	OE developer: usually platform provider		
STRS_FileRemove()																		
Description	Remove the closed file.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) name of file to be removed.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The file is no longer available, and the space where it was stored becomes available.																	
Applicable to	OE developer: usually platform provider																	
7.3.8	STRS Infrastructure File Control API	<p>(STRS-76) The STRS infrastructure shall contain a callable STRS_FileRename method as described in Table 57, STRS_FileRename().</p> <p style="text-align: center;">Table 57—STRS_FileRename()</p> <table><tr><td colspan="2">STRS_FileRename()</td></tr><tr><td>Description</td><td>Rename the closed file where the new file name does not exist prior to the call.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) current name of file.newName - (in char *) new name of file after rename.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>The contents of the old file are now associated with the new file name.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_FileRename()		Description	Rename the closed file where the new file name does not exist prior to the call.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) current name of file.newName - (in char *) new name of file after rename.	Return	status (STRS_Result)	Precondition	None	Postcondition	The contents of the old file are now associated with the new file name.	Applicable to	OE developer: usually platform provider		
STRS_FileRename()																		
Description	Rename the closed file where the new file name does not exist prior to the call.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.oldName - (in char *) current name of file.newName - (in char *) new name of file after rename.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	The contents of the old file are now associated with the new file name.																	
Applicable to	OE developer: usually platform provider																	
7.3.9	STRS Infrastructure Messaging API	(STRS-77) The STRS applications shall use the STRS Infrastructure Messaging, STRS Infrastructure Data Source, and STRS Infrastructure Data Sink methods to send messages between components.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
7.3.9	STRS Infrastructure Messaging API	<p>(STRS-126) The STRS infrastructure shall contain a callable STRS_MessageQueueCreate method as described in Table 58, STRS_MessageQueueCreate().</p> <p style="text-align: center;">Table 58—STRS_MessageQueueCreate()</p> <table><tr><th colspan="2">STRS_MessageQueueCreate()</th></tr><tr><td>Description</td><td>Create a FIFO message queue if a handle does not already exist having the given name</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.queueName - (in char *) unique name of the queue.nb – (STRS_Buffer_Size) maximum size of buffer containing messages.nmax – (STRS_Queue_Max_Messages) maximum number of messages in queue.</td></tr><tr><td>Return</td><td>handle ID of queue or error status (STRS_HandleID)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Queue is created unless an error occurs.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_MessageQueueCreate()		Description	Create a FIFO message queue if a handle does not already exist having the given name	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.queueName - (in char *) unique name of the queue.nb – (STRS_Buffer_Size) maximum size of buffer containing messages.nmax – (STRS_Queue_Max_Messages) maximum number of messages in queue.	Return	handle ID of queue or error status (STRS_HandleID)	Precondition	None	Postcondition	Queue is created unless an error occurs.	Applicable to	OE developer: usually platform provider		
STRS_MessageQueueCreate()																		
Description	Create a FIFO message queue if a handle does not already exist having the given name																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.queueName - (in char *) unique name of the queue.nb – (STRS_Buffer_Size) maximum size of buffer containing messages.nmax – (STRS_Queue_Max_Messages) maximum number of messages in queue.																	
Return	handle ID of queue or error status (STRS_HandleID)																	
Precondition	None																	
Postcondition	Queue is created unless an error occurs.																	
Applicable to	OE developer: usually platform provider																	
7.3.9	STRS Infrastructure Messaging API	<p>(STRS-127) The STRS infrastructure shall contain a callable STRS_MessageQueueDelete method as described in Table 59, STRS_MessageQueueDelete().</p> <p style="text-align: center;">Table 59—STRS_MessageQueueDelete()</p> <table><tr><th colspan="2">STRS_MessageQueueDelete()</th></tr><tr><td>Description</td><td>Delete a queue if it exists.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toQueue - (inout STRS_HandleID) handle ID of queue to delete.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Queue is deleted.</td></tr></table>	STRS_MessageQueueDelete()		Description	Delete a queue if it exists.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toQueue - (inout STRS_HandleID) handle ID of queue to delete.	Return	status (STRS_Result)	Precondition	None	Postcondition	Queue is deleted.				
STRS_MessageQueueDelete()																		
Description	Delete a queue if it exists.																	
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toQueue - (inout STRS_HandleID) handle ID of queue to delete.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	Queue is deleted.																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Applicable to	OE developer: usually platform provider		
7.3.9	STRS Infrastructure Messaging API	(STRS-128) The STRS infrastructure shall contain a callable STRS_PubSubCreate method as described in Table 60, STRS_PubSubCreate(). Table 60—STRS_PubSubCreate() STRS_PubSubCreate() DescriptionCreate a Pub/Sub handle ID that is a proxy used to receive and redistribute messages using STRS_Write unless the handle name already is used somewhere else. Parameters <ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.pubsubName - (in char *) unique name of the Pub/Sub. Returnhandle ID of Pub/Sub or error status (STRS_HandleID) PreconditionNone PostconditionPub/Sub is created unless an error occurs. Applicable toOE developer: usually platform provider			
7.3.9	STRS Infrastructure Messaging API	(STRS-129) The STRS infrastructure shall contain a callable STRS_PubSubDelete method as described in Table 61, STRS_PubSubDelete(). Table 61—STRS_PubSubDelete() STRS_PubSubDelete() DescriptionDelete a Pub/Sub if it exists. Any association between a publisher and subscriber that references the Pub/Sub is removed. Parameters <ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toPubSub - (inout STRS_HandleID) handle ID of Pub/Sub to delete. Returnstatus (STRS_Result) PreconditionNone PostconditionSpecified Pub/Sub is deleted and any associations are removed.			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Applicable to	OE developer: usually platform provider														
7.3.9	STRS Infrastructure Messaging API	(STRS-80) The STRS infrastructure shall contain a callable STRS_Register method as described in Table 62, STRS_Register(). Table 62—STRS_Register() STRS_Register() <table><tr><td>Description</td><td>Register an association between a publisher and subscriber where both exist. Disallow duplicates between the same publisher and subscriber.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.useQID - (in STRS_HandleID) handle ID of Pub/Sub that will be used as a sink; the publisher.actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should respond to the request as a sink; the subscriber.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Association between publisher and subscriber is registered, if allowed.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>		Description	Register an association between a publisher and subscriber where both exist. Disallow duplicates between the same publisher and subscriber.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.useQID - (in STRS_HandleID) handle ID of Pub/Sub that will be used as a sink; the publisher.actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should respond to the request as a sink; the subscriber.	Return	status (STRS_Result)	Precondition	None	Postcondition	Association between publisher and subscriber is registered, if allowed.	Applicable to	OE developer: usually platform provider		
Description	Register an association between a publisher and subscriber where both exist. Disallow duplicates between the same publisher and subscriber.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.useQID - (in STRS_HandleID) handle ID of Pub/Sub that will be used as a sink; the publisher.actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should respond to the request as a sink; the subscriber.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	Association between publisher and subscriber is registered, if allowed.																
Applicable to	OE developer: usually platform provider																
7.3.9	STRS Infrastructure Messaging API	(STRS-81) The STRS infrastructure shall contain a callable STRS_Unregister method as described in Table 63, STRS_Unregister(). Table 63—STRS_Unregister() STRS_Unregister() <table><tr><td>Description</td><td>Remove an association between a publisher and subscriber, if the association exists.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.</td></tr></table>		Description	Remove an association between a publisher and subscriber, if the association exists.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.										
Description	Remove an association between a publisher and subscriber, if the association exists.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A							
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale		
			<ul style="list-style-type: none">useQID - (in STRS_HandleID) handle ID of Pub/Sub that was used as a sink; the publisher.actQID - (in STRS_HandleID) handle ID of Pub/Sub, file, device, or target component that should no longer respond to the request as a sink; usually the subscriber.				
		Return	status (STRS_Result)				
		Precondition	None				
		Postcondition	Association between publisher and subscriber is removed.				
		Applicable to	OE developer: usually platform provider				
7.3.10	STRS Infrastructure Time Control API	(STRS-82) Any portion of the STRS Applications on the GPP needing time control shall use the STRS Infrastructure Time Control methods to access the hardware and software timers.					
7.3.10	STRS Infrastructure Time Control API	(STRS-130) The implementer of an STRS clock/timer software component for use with STRS_GetTime shall document it to include handle name, kind, epoch, resolution, use of leap seconds, and whether it should match a time somewhere else, as described further in Table 64, Document STRS Clock/Timer. Table 64—Document STRS Clock/Timer					
		Applicable to	STRS clock/timer developer, which may be platform provider or application developer.				
7.3.10	STRS Infrastructure Time Control API	(STRS-83) The STRS infrastructure shall contain a callable STRS_GetNanoseconds method as described in Table 65, STRS_GetNanoseconds(). Table 65—STRS_GetNanoseconds()					
		STRS_GetNanoseconds()					
		Description	Get the number of nanoseconds from the STRS_TimeWarp object.				
		Parameters	<ul style="list-style-type: none">twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the nanoseconds portion of the time increment is extracted.				
		Return	Integer number of nanoseconds in the STRS_TimeWarp object representing a time interval. (STRS_Nanoseconds)				

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.10	STRS Infrastructure Time Control API	(STRS-84) The STRS infrastructure shall contain a callable STRS_GetSeconds method as described in Table 66, STRS_GetSeconds().			
		Table 66—STRS_GetSeconds()			
		STRS_GetSeconds()			
		Description	Get the number of seconds from the STRS_TimeWarp object.		
		Parameters	<ul style="list-style-type: none">twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the nanoseconds portion of the time increment is extracted.		
		Return	integer number of seconds in the STRS_TimeWarp object representing a time interval. (STRS_Seconds)		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.10	STRS Infrastructure Time Control API	(STRS-85) The STRS infrastructure shall contain a callable STRS_GetTime method as described in Table 67, STRS_GetTime().			
		Table 67—STRS_GetTime()			
		STRS_GetTime()			
		Description	Get the current base time and the corresponding time of a specified type (kind). The base clock/timer is usually a hardware timer. The variable kind is used to obtain a nonbase time at a specified offset from the base time. An offset is usually specified to ensure that the clock is monotonically increasing after a power reset or synchronized with another clock/timer. To compute the time interval between two nonbase times of different kinds, the function is called twice and the interval is modified by the difference between the two base times.		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.baseTime - (inout STRS_TimeWarp) current time of the base timer.kind - (in STRS_Clock_Kind) type of clock/timer.kindTime - (inout STRS_TimeWarp) current time of the specified timer.		
		Return	status (STRS_Result)		
		Precondition	None		
		Postcondition	None		
		Applicable to	OE developer: usually platform provider		
7.3.10	STRS Infrastructure Time Control API	(STRS-131) The STRS infrastructure shall contain a callable STRS_GetTimeAdjust method as described in Table 68, STRS_GetTimeAdjust(). Table 68—STRS_GetTimeAdjust() STRS_GetTimeRate()			
		Description	Get the current time rate for the specified clock/timer which, when applied to the clock specified by <i>its handle ID</i> , will more closely synchronize it with another.		
		Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.		
		Return	iRate (STRS_TimeRate) an integer time rate. Units are specific to the clock/timer.		
		Precondition	None		
		Postcondition	Time rate is obtained or computed.		
		Applicable to	OE developer: usually platform provider		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale														
7.3.10	STRS Infrastructure Time Control API	<p>(STRS-86) The STRS infrastructure shall contain a callable STRS_GetTimeWarp method as described in Table 69, STRS_GetTimeWarp().</p> <p style="text-align: center;">Table 69—STRS_GetTimeWarp()</p> <table><tr><th colspan="2">STRS_GetTimeWarp()</th></tr><tr><td>Description</td><td>Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">• isec - (in STRS_Seconds) number of seconds in the time interval• nsec - (in STRS_Nanoseconds) number of nanoseconds in the fractional portion of the time interval</td></tr><tr><td>Return</td><td>STRS_TimeWarp object representing the time interval.</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>None</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>	STRS_GetTimeWarp()		Description	Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.	Parameters	<ul style="list-style-type: none">• isec - (in STRS_Seconds) number of seconds in the time interval• nsec - (in STRS_Nanoseconds) number of nanoseconds in the fractional portion of the time interval	Return	STRS_TimeWarp object representing the time interval.	Precondition	None	Postcondition	None	Applicable to	OE developer: usually platform provider		
STRS_GetTimeWarp()																		
Description	Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.																	
Parameters	<ul style="list-style-type: none">• isec - (in STRS_Seconds) number of seconds in the time interval• nsec - (in STRS_Nanoseconds) number of nanoseconds in the fractional portion of the time interval																	
Return	STRS_TimeWarp object representing the time interval.																	
Precondition	None																	
Postcondition	None																	
Applicable to	OE developer: usually platform provider																	
7.3.10	STRS Infrastructure Time Control API	<p>[STRS-87] The STRS infrastructure shall contain a callable STRS_SetTime method as described in Table 70, STRS_SetTime().</p> <p style="text-align: center;">Table 70—STRS_SetTime()</p> <table><tr><th colspan="2">STRS_SetTime()</th></tr><tr><td>Description</td><td>Set the current time in the specified clock/timer by adjusting the time offset.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toDev - (in STRS_HandleID) handle ID of device that should respond to the request.• kind - (in STRS_Clock_Kind) type of clock/timer.• delta - (in STRS_TimeWarp) increment to add to specified clock/timer.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Time is adjusted.</td></tr></table>	STRS_SetTime()		Description	Set the current time in the specified clock/timer by adjusting the time offset.	Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toDev - (in STRS_HandleID) handle ID of device that should respond to the request.• kind - (in STRS_Clock_Kind) type of clock/timer.• delta - (in STRS_TimeWarp) increment to add to specified clock/timer.	Return	status (STRS_Result)	Precondition	None	Postcondition	Time is adjusted.				
STRS_SetTime()																		
Description	Set the current time in the specified clock/timer by adjusting the time offset.																	
Parameters	<ul style="list-style-type: none">• fromWF - (in STRS_HandleID) handle ID of current component making the request.• toDev - (in STRS_HandleID) handle ID of device that should respond to the request.• kind - (in STRS_Clock_Kind) type of clock/timer.• delta - (in STRS_TimeWarp) increment to add to specified clock/timer.																	
Return	status (STRS_Result)																	
Precondition	None																	
Postcondition	Time is adjusted.																	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																	
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale												
		Applicable to	OE developer: usually platform provider														
7.3.10	STRS Infrastructure Time Control API	(STRS-132) The STRS infrastructure shall contain a callable STRS_SetTimeAdjust method as described in Table 71, STRS_SetTimeAdjust(). Table 71—STRS_SetTimeAdjust() STRS_SetTimeAdjust() <table><tr><td>Description</td><td>Set the current time rate in the specified clock/timer.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.iRate - (in STRS_TimeRate) a rate applied to the specified clock/timer to set the clock/timer relative time. Units are specific to the clock/timer.</td></tr><tr><td>Return</td><td>status (STRS_Result)</td></tr><tr><td>Precondition</td><td>None</td></tr><tr><td>Postcondition</td><td>Time rate is adjusted.</td></tr><tr><td>Applicable to</td><td>OE developer: usually platform provider</td></tr></table>		Description	Set the current time rate in the specified clock/timer.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.iRate - (in STRS_TimeRate) a rate applied to the specified clock/timer to set the clock/timer relative time. Units are specific to the clock/timer.	Return	status (STRS_Result)	Precondition	None	Postcondition	Time rate is adjusted.	Applicable to	OE developer: usually platform provider		
Description	Set the current time rate in the specified clock/timer.																
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.toDev - (in STRS_HandleID) handle ID of device that should respond to the request.iRate - (in STRS_TimeRate) a rate applied to the specified clock/timer to set the clock/timer relative time. Units are specific to the clock/timer.																
Return	status (STRS_Result)																
Precondition	None																
Postcondition	Time rate is adjusted.																
Applicable to	OE developer: usually platform provider																
7.3.10	STRS Infrastructure Time Control API	(STRS-133) The STRS infrastructure shall contain a callable STRS_Sleep method as described in Table 72, STRS_Sleep(). Table 72—STRS_Sleep() STRS_Sleep() <table><tr><td>Description</td><td>Delays the execution of the application for at least the time specified in the STRS_TimeWarp argument that contains the number of seconds and nanoseconds in the time interval. The time interval may still not be accurate depending on the underlying timer resolution and thread interaction.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">clockID – (STRS_HandleID) the handle ID of the timer/clock.twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the time is extracted.</td></tr></table>		Description	Delays the execution of the application for at least the time specified in the STRS_TimeWarp argument that contains the number of seconds and nanoseconds in the time interval. The time interval may still not be accurate depending on the underlying timer resolution and thread interaction.	Parameters	<ul style="list-style-type: none">clockID – (STRS_HandleID) the handle ID of the timer/clock.twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the time is extracted.										
Description	Delays the execution of the application for at least the time specified in the STRS_TimeWarp argument that contains the number of seconds and nanoseconds in the time interval. The time interval may still not be accurate depending on the underlying timer resolution and thread interaction.																
Parameters	<ul style="list-style-type: none">clockID – (STRS_HandleID) the handle ID of the timer/clock.twObj - (in STRS_TimeWarp) the STRS_TimeWarp object from which the time is extracted.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A												
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale							
			<ul style="list-style-type: none">absOrRel- (Boolean) true, if absolute time is specified; false, if relative time is specified.									
		Return	(STRS_Result) STRS_OK when successful. STRS_ERROR for error. STRS_WARNING if interrupted.									
		Precondition	None									
		Postcondition	None									
		Applicable to	OE developer: usually platform provider									
7.3.10	STRS Infrastructure Time Control API	[STRS-88] The STRS infrastructure shall contain a callable STRS_TimeSynch method as described in Table 73, STRS_TimeSynch(). Table 73—STRS_TimeSynch() STRS_TimeSynch() <table><tr><td>Description</td><td>Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external, or whether the clocks differ by amounts that exceed the maximum step size allowed.</td></tr><tr><td>Parameters</td><td><ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.refDev - (in STRS_HandleID) handle ID of reference device containing the reference clock/timer.ref - (in STRS_Clock_Kind) type of reference clock/timer.targetDev - (in STRS_HandleID) handle ID of target device to synchronize.target - (in STRS_Clock_Kind) type of clock/timer to synchronize with reference clock/timer.stepMax – (in STRS_TimeWarp) maximum step size to allow at a time, which can be used for gradual time adjustment. Zero implies no limit in step size.</td></tr><tr><td>Return</td><td>status (STRS_Result) where a positive value indicates the number of steps left to adjust at the maximum step size.</td></tr></table>			Description	Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external, or whether the clocks differ by amounts that exceed the maximum step size allowed.	Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.refDev - (in STRS_HandleID) handle ID of reference device containing the reference clock/timer.ref - (in STRS_Clock_Kind) type of reference clock/timer.targetDev - (in STRS_HandleID) handle ID of target device to synchronize.target - (in STRS_Clock_Kind) type of clock/timer to synchronize with reference clock/timer.stepMax – (in STRS_TimeWarp) maximum step size to allow at a time, which can be used for gradual time adjustment. Zero implies no limit in step size.	Return	status (STRS_Result) where a positive value indicates the number of steps left to adjust at the maximum step size.		
Description	Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external, or whether the clocks differ by amounts that exceed the maximum step size allowed.											
Parameters	<ul style="list-style-type: none">fromWF - (in STRS_HandleID) handle ID of current component making the request.refDev - (in STRS_HandleID) handle ID of reference device containing the reference clock/timer.ref - (in STRS_Clock_Kind) type of reference clock/timer.targetDev - (in STRS_HandleID) handle ID of target device to synchronize.target - (in STRS_Clock_Kind) type of clock/timer to synchronize with reference clock/timer.stepMax – (in STRS_TimeWarp) maximum step size to allow at a time, which can be used for gradual time adjustment. Zero implies no limit in step size.											
Return	status (STRS_Result) where a positive value indicates the number of steps left to adjust at the maximum step size.											

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A							
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale	
		Precondition	None				
		Postcondition	Clocks are more synchronized.				
		Applicable to	OE developer: usually platform provider				
7.3.11	STRS Predefined Data	(STRS-89) The STRS platform provider shall provide an STRS.h file containing the STRS predefined data shown in Table 74, STRS Predefined Data.					
		Table 74—STRS Predefined Data					
		Typedefs					
		Name	Type	Description			
		STRS_Access	A number	Used to indicate how reading and/or writing of a file or queue is done. See also constants STRS_ACCESS_APPEND, STRS_ACCESS_BOTH, STRS_ACCESS_READ, and STRS_ACCESS_WRITE.			
		STRS_Buffer_Size	A number	Used to represent a buffer size in bytes. The type of the number is to be long enough to contain the maximum number of bytes to reserve or to transfer with a read or write.			
		STRS_Clock_Kind	A number	Used to represent a kind of clock or timer. The type of the number is to be long enough to contain the maximum number of kinds of clocks and timers.			
		STRS_File_Size	A number	Used to represent a size in bytes. The type of the number is to be long enough to contain the number of bytes in GPP storage. Specific negative error values returned indicate an error.			

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		STRS_HandleID	A number	Used to represent an STRS application, device, file, or queue. Specific error value(s) returned indicate an error.	
		STRS_int8	A number	Used for an 8-bit signed integer	
		STRS_int16	A number	Used for a 16-bit signed integer	
		STRS_int32	A number	Used for a 32-bit signed integer	
		STRS_int64	A number	Used for a 64-bit signed integer	
		STRS_ISR_Function	A static C-style function pointer	Used to define static C-style function pointers passed to the STRS SetISR() method. The function passed to the STRS_SetISR() method is defined with any arguments needed by the OE for its underlying system calls. The OE-specific documentation contains the description of any arguments.	
		STRS_Message	A char array pointer	Used for messages.	
		STRS_Nanoseconds*	A number	Used to hold the number of nanoseconds in the STRS_TimeWarp object, at least 32 bits for a signed integer. Using 32 bits would allow a maximum of 2,147,483,647 nanoseconds = 2.147483647 seconds that would allow the sum/difference of the nanosecond counter in 2 normalized STRS_TimeWarp objects. Each additional bit multiplies the 2.1475 seconds in the nanosecond counter by 2.	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		STRS_Property_Name		Used to hold a property name, usually a set of characters (char).	
		STRS_Property_Value		Used to hold a property value, usually a set of characters (char).	
		STRS_Queue_Max_Messages	A number	Used to represent the maximum number of messages allowed in the queue.	
		STRS_Result	A number	Used to represent a return value, where there are specific values that indicate an error.	
		STRS_Seconds*	A number	Used to hold the number of seconds in the STRS_TimeWarp object, at least 32 bit signed integer. Using 32 bits would allow a maximum of 2,147,483,647 seconds = 68.05 years. For an epoch of 1970, the 32-bit second counter runs out in 2038. Each additional bit multiplies the 68.05 years in the second counter by 2.	
		STRS_TestID	A number	Used to represent the built-in test or ground test to be performed by APP_RunTest or APP_GroundTest, respectively. See also STRS_TEST_STATUS and STRS_TEST_USER_BASE.	
		STRS_TimeWarp	A representation of a time delay	The representation of a time delay able to hold the number of seconds and nanoseconds in the time delay so that the corresponding macros can extract them. The time delay is meant to be used for recurrent processes such as in health management. The implementation is mission and/or platform specific and is most likely a struct. The maximum	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				number of seconds in a time delay cannot be greater than $2^{(\text{no. of bits in STRS_Seconds} - 1)}$ seconds. Divide the maximum number of seconds by 31557600 (60*60*24*365.25) to get the approximate number of years.		
		STRS_TimeRate	A number	Integer indicating time rate factor used to adjust time relative to clock accuracy defined by STRS_TimeRatePPS.		
		STRS_Type	A number	Used to indicate whether a file is text or binary. See also constants STRS_TYPE_BINARY and STRS_TYPE_TEXT.		
		STRS_uint8	A number	Used for an 8-bit unsigned integer		
		STRS_uint16	A number	Used for a 16-bit unsigned integer		
		STRS_uint32	A number	Used for a 32-bit unsigned integer		
		STRS_uint64	A number	Used for a 64-bit unsigned integer		
		Constants				
		Name	Type	Description		
		STRS_ACCESS_APPEND	STRS_Access	Indicates that writing is allowed such that previous data written are preserved and new data are written following any previous data. Corresponds to ISO C fopen		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				mode “a”.		
		STRS_ACCESS_BOTH	STRS_Access	Indicates that both reading and writing are allowed. Corresponds to ISO C fopen mode “r+” used for update.		
		STRS_ACCESS_READ	STRS_Access	Indicates that reading is allowed. Corresponds to ISO C fopen mode “r”.		
		STRS_ACCESS_WRITE	STRS_Access	Indicates that writing is allowed. Corresponds to ISO C fopen mode “w”.		
		STRS_OK	STRS_Result	Indicates that the STRS_Result is valid. See also STRS_IsOK().		
		STRS_ERROR	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that the application or other component is still usable. See also STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_ERROR_QUEUE	STRS_HandleID	Indicates that the log queue is for error messages. See also STRS_GetErrorQueue().		
		STRS_FATAL	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating a serious error such that the application or other component is not usable. See also		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_FATAL_QUEUE	STRS_HandleID	Indicates that the log queue is for fatal messages. The fatal queue is used for messages that the fault monitoring and recovery functions are to deal with immediately. The messages are sent to the Flight Computer for further handling. See also STRS_GetErrorQueue().		
		STRS_TELEMETRY_QUEUE	STRS_HandleID	Indicates that the log queue is for telemetry data.		
		STRS_TYPE_BINARY	STRS_Type	Indicates that a file is a binary file.		
		STRS_TYPE_TEXT	STRS_Type	Indicates that a file is a text file.		
		STRS_WARNING	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that there may be little or no effect on the operation of the application or other component. See also STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_WARNING_QUEUE	STRS_HandleID	Indicates that the log queue is for warning messages. See also STRS_GetErrorQueue().		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		STRS_OE_HANDLE_NAME	char *	A handle name used to find handle ID that may be used to query the OE	
		OEClockAppName	char *	The handle name used to find handle ID that may be used to access time using STRS_GetTime used in a timestamp.	
		OEClockKind	STRS_Clock_Kind	The number used to indicate the type of clock/timer used in a timestamp.	
		STRS_TimeRatePPS	STRS_TimeRate	Integer accuracy of time rate in number of parts per second.	
		STRS_MAX_PROPERTY_NAME_SIZE	#define	The maximum number of characters in the name in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.	
		STRS_MAX_PROPERTY_VALUE_SIZE	#define	The maximum number of characters in the value in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.	
		STRS_MAX_PATH_NAME_SIZE	#define	The maximum number of characters in a path name for the	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				OE, not including the final '\0'. Any use of this as a dimension should be increased by one.		
		STRS_MAX_HANDLE_NAME_SIZE	#define	The maximum number of characters in a handle name for the OE, not including the final '\0'. Any use of this as a dimension should be increased by one.		
		STRS_MAX_LOG_MESSAGE_SIZE	#define	The maximum number of characters in each message submitted to the log, not including the final '\0'. Any use of this as a dimension should be increased by one.		
		STRS_MAX_QUEUE_MESSAGES	STRS_Queue_Max_Messages	The maximum number of messages that can be stored in a queue. Not normally used except for testing.		
7.3.11	STRS Predefined Data	(STRS-106) An STRS application shall use the appropriate constant, typedef, or struct defined in Table 74, STRS Predefined Data, when the data are used to interact with the STRS APIs.				

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)		Applicable (Yes or No)	If No, Enter Rationale
		Table 74—STRS Predefined Data			
		Typedefs			
		Name	Type	Description	
		STRS_Access	A number	Used to indicate how reading and/or writing of a file or queue is done. See also constants STRS_ACCESS_APPEND, STRS_ACCESS_BOTH, STRS_ACCESS_READ, and STRS_ACCESS_WRITE.	
		STRS_Buffer_Size	A number	Used to represent a buffer size in bytes. The type of the number is to be long enough to contain the maximum number of bytes to reserve or to transfer with a read or write.	
		STRS_Clock_Kind	A number	Used to represent a kind of clock or timer. The type of the number is to be long enough to contain the maximum number of kinds of clocks and timers.	
		STRS_File_Size	A number	Used to represent a size in bytes. The type of the number is to be long enough to contain the number of bytes in GPP storage. Specific negative error values returned indicate an error.	
		STRS_HandleID	A number	Used to represent an STRS application, device, file, or queue. Specific error value(s) returned indicate an error.	
		STRS_int8	A number	Used for an 8-bit signed integer	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		STRS_int16	A number	Used for a 16-bit signed integer	
		STRS_int32	A number	Used for a 32-bit signed integer	
		STRS_int64	A number	Used for a 64-bit signed integer	
		STRS_ISR_Function	A static C-style function pointer	Used to define static C-style function pointers passed to the STRS_SetISR() method. The function passed to the STRS_SetISR() method is defined with any arguments needed by the OE for its underlying system calls. The OE-specific documentation contains the description of any arguments.	
		STRS_Message	A char array pointer	Used for messages.	
		STRS_Nanoseconds*	A number	Used to hold the number of nanoseconds in the STRS_TimeWarp object, at least 32 bits for a signed integer. Using 32 bits would allow a maximum of 2,147,483,647 nanoseconds = 2.147483647 seconds that would allow the sum/difference of the nanosecond counter in 2 normalized STRS_TimeWarp objects. Each additional bit multiplies the 2.1475 seconds in the nanosecond counter by 2.	
		STRS_Property_Name		Used to hold a property name, usually a set of characters (char).	
		STRS_Property_Value		Used to hold a property value, usually a set of characters (char).	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		STRS_Queue_Max_Messages	A number	Used to represent the maximum number of messages allowed in the queue.	
		STRS_Result	A number	Used to represent a return value, where there are specific values that indicate an error.	
		STRS_Seconds*	A number	Used to hold the number of seconds in the STRS_TimeWarp object, at least 32 bit signed integer. Using 32 bits would allow a maximum of 2,147,483,647 seconds = 68.05 years. For an epoch of 1970, the 32-bit second counter runs out in 2038. Each additional bit multiplies the 68.05 years in the second counter by 2.	
		STRS_TestID	A number	Used to represent the built-in test or ground test to be performed by APP_RunTest or APP_GroundTest, respectively. See also STRS_TEST_STATUS and STRS_TEST_USER_BASE.	
		STRS_TimeWarp	A representation of a time delay	The representation of a time delay able to hold the number of seconds and nanoseconds in the time delay so that the corresponding macros can extract them. The time delay is meant to be used for recurrent processes such as in health management. The implementation is mission and/or platform specific and is most likely a struct. The maximum number of seconds in a time delay cannot be greater than $2^{(\text{no. of bits in STRS_Seconds} - 1)}$ seconds. Divide the maximum number of seconds by 31557600 ($60*60*24*365.25$) to get the approximate number of years.	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
		STRS_TimeRate	A number	Integer indicating time rate factor used to adjust time relative to clock accuracy defined by STRS_TimeRatePPS.		
		STRS_Type	A number	Used to indicate whether a file is text or binary. See also constants STRS_TYPE_BINARY and STRS_TYPE_TEXT.		
		STRS_uint8	A number	Used for an 8-bit unsigned integer		
		STRS_uint16	A number	Used for a 16-bit unsigned integer		
		STRS_uint32	A number	Used for a 32-bit unsigned integer		
		STRS_uint64	A number	Used for a 64-bit unsigned integer		
		Constants				
		Name	Type	Description		
		STRS_ACCESS_APPEND	STRS_Access	Indicates that writing is allowed such that previous data written are preserved and new data are written following any previous data. Corresponds to ISO C fopen mode “a”.		
		STRS_ACCESS_BOTH	STRS_Access	Indicates that both reading and writing are allowed. Corresponds to ISO C fopen mode “r+” used		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				for update.		
		STRS_ACCESS_READ	STRS_Access	Indicates that reading is allowed. Corresponds to ISO C fopen mode “r”.		
		STRS_ACCESS_WRITE	STRS_Access	Indicates that writing is allowed. Corresponds to ISO C fopen mode “w”.		
		STRS_OK	STRS_Result	Indicates that the STRS_Result is valid. See also STRS_IsOK().		
		STRS_ERROR	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that the application or other component is still usable. See also STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_ERROR_QUEUE	STRS_HandleID	Indicates that the log queue is for error messages. See also STRS_GetErrorQueue().		
		STRS_FATAL	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating a serious error such that the application or other component is not usable. See also STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_FATAL_QUEUE	STRS_HandleID	Indicates that the log queue is for fatal messages. The fatal queue is		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A						
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale
				used for messages that the fault monitoring and recovery functions are to deal with immediately. The messages are sent to the Flight Computer for further handling. See also STRS_GetErrorQueue().		
		STRS_TELEMETRY_QUEUE	STRS_HandleID	Indicates that the log queue is for telemetry data.		
		STRS_TYPE_BINARY	STRS_Type	Indicates that a file is a binary file.		
		STRS_TYPE_TEXT	STRS_Type	Indicates that a file is a text file.		
		STRS_WARNING	STRS_Result	Indicates that the STRS_Result is invalid. Specific value indicating an error such that there may be little or no effect on the operation of the application or other component. See also STRS_IsOK() and STRS_GetErrorQueue().		
		STRS_WARNING_QUEUE	STRS_HandleID	Indicates that the log queue is for warning messages. See also STRS_GetErrorQueue().		
		STRS_OE_HANDLE_NAME*	char *	A handle name used to find handle ID that may be used to query the OE		

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A					
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)
		OEClockAppName*	char *	The handle name used to find handle ID that may be used to access time using STRS_GetTime used in a timestamp.	
		OEClockKind*	STRS_Clock_Kind	The number used to indicate the type of clock/timer used in a timestamp.	
		STRS_TimeRatePPS	STRS_TimeRate	Integer accuracy of time rate in number of parts per second.	
		STRS_MAX_PROPERTY_NAME_SIZE	#define	The maximum number of characters in the name in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.	
		STRS_MAX_PROPERTY_VALUE_SIZE	#define	The maximum number of characters in the value in a Property object, not including the final '\0'. Any use of this as a dimension should be increased by one.	
		STRS_MAX_PATH_NAME_SIZE	#define	The maximum number of characters in a path name for the OE, not including the final '\0'. Any use of this as a dimension should be increased by one.	

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																		
Section	Description	Requirement in this NASA Technical Standard (NTS)			Applicable (Yes or No)	If No, Enter Rationale												
		STRS_MAX_HANDLE_NAME_SIZE	#define	The maximum number of characters in a handle name for the OE, not including the final ‘\0’. Any use of this as a dimension should be increased by one.														
		STRS_MAX_LOG_MESSAGE_SIZE	#define	The maximum number of characters in each message submitted to the log, not including the final ‘\0’. Any use of this as a dimension should be increased by one.														
		STRS_MAX_QUEUE_MESSAGES	STRS_Queue_Max_Messages	The maximum number of messages that can be stored in a queue. Not normally used except for testing.														
7.3.11	STRS Predefined Data	(STRS-134) The STRS infrastructure shall have the queryable parameter names in Table 75, Queryable Platform Parameter Names, for which values may be obtained using STRS_Query with the handle ID corresponding to the handle name STRS_OE_HANDLE_NAME. <table><tr><th colspan="3">Table 75—Queryable Platform Parameter Names</th></tr><tr><th>Parameter Name</th><th>Description</th><th>Notes</th></tr><tr><td>PLATFORM_PROVIDER</td><td>Unique name of STRS platform provider</td><td>This is usually a company name or university, followed by a subsidiary, division, or department name.</td></tr><tr><td>PLATFORM_OE_VERSION</td><td>Unique version number for platform STRS infrastructure software</td><td></td></tr></table>			Table 75—Queryable Platform Parameter Names			Parameter Name	Description	Notes	PLATFORM_PROVIDER	Unique name of STRS platform provider	This is usually a company name or university, followed by a subsidiary, division, or department name.	PLATFORM_OE_VERSION	Unique version number for platform STRS infrastructure software			
Table 75—Queryable Platform Parameter Names																		
Parameter Name	Description	Notes																
PLATFORM_PROVIDER	Unique name of STRS platform provider	This is usually a company name or university, followed by a subsidiary, division, or department name.																
PLATFORM_OE_VERSION	Unique version number for platform STRS infrastructure software																	
7.3.11	STRS Predefined	STRS-135) An STRS application shall have the queryable parameter names in Table 76, Queryable Application Parameter Names, for which values may be obtained using STRS_Query with the handle ID of the application.																

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A																														
Section	Description	Requirement in this NASA Technical Standard (NTS)	Applicable (Yes or No)	If No, Enter Rationale																										
	Data	<table><tr><th colspan="3">Table 76—Queryable Application Parameter Names</th></tr><tr><th>Parameter Name</th><th>Description</th><th>Notes</th></tr><tr><td>WAVEFORM_DEVELOPER</td><td>Unique name of application developer</td><td>This is usually a company name or university, followed by a subsidiary, division, or department name.</td></tr><tr><td>WAVEFORM_VERSION</td><td>Unique version number for STRS application software</td><td></td></tr><tr><td>WAVEFORM_STATE</td><td>Current application state</td><td>Documented per STRS-12(3)</td></tr></table>	Table 76—Queryable Application Parameter Names			Parameter Name	Description	Notes	WAVEFORM_DEVELOPER	Unique name of application developer	This is usually a company name or university, followed by a subsidiary, division, or department name.	WAVEFORM_VERSION	Unique version number for STRS application software		WAVEFORM_STATE	Current application state	Documented per STRS-12(3)													
Table 76—Queryable Application Parameter Names																														
Parameter Name	Description	Notes																												
WAVEFORM_DEVELOPER	Unique name of application developer	This is usually a company name or university, followed by a subsidiary, division, or department name.																												
WAVEFORM_VERSION	Unique version number for STRS application software																													
WAVEFORM_STATE	Current application state	Documented per STRS-12(3)																												
7.4.1	STRS Application Environment Profile	[STRS-90] The STRS OE shall provide the interfaces described in POSIX® standard IEEE 1003.13 profile PSE51.																												
7.4.1	STRS Application Environment Profile	<p>[STRS-91] STRS applications shall use POSIX® methods except for the unsafe functions listed in Table 77, Replacements for Unsafe Functions.</p> <table><tr><th colspan="2">Table 77—Replacements for Unsafe Functions</th></tr><tr><th>Unsafe Function Do Not Use!</th><th>Reentrant Counterpart OK to Use</th></tr><tr><td>abort</td><td>STRS_AbortApp</td></tr><tr><td>asctime</td><td>asctime_r</td></tr><tr><td>atexit</td><td>-</td></tr><tr><td>ctermid</td><td>ctermid_r</td></tr><tr><td>ctime</td><td>ctime_r</td></tr><tr><td>exit</td><td>STRS_AbortApp</td></tr><tr><td>getlogin</td><td>getlogin_r</td></tr><tr><td>gmtime</td><td>gmtime_r</td></tr><tr><td>localtime</td><td>localtime_r</td></tr><tr><td>rand</td><td>rand_r</td></tr><tr><td>readdir</td><td>readdir_r</td></tr></table>	Table 77—Replacements for Unsafe Functions		Unsafe Function Do Not Use!	Reentrant Counterpart OK to Use	abort	STRS_AbortApp	asctime	asctime_r	atexit	-	ctermid	ctermid_r	ctime	ctime_r	exit	STRS_AbortApp	getlogin	getlogin_r	gmtime	gmtime_r	localtime	localtime_r	rand	rand_r	readdir	readdir_r		
Table 77—Replacements for Unsafe Functions																														
Unsafe Function Do Not Use!	Reentrant Counterpart OK to Use																													
abort	STRS_AbortApp																													
asctime	asctime_r																													
atexit	-																													
ctermid	ctermid_r																													
ctime	ctime_r																													
exit	STRS_AbortApp																													
getlogin	getlogin_r																													
gmtime	gmtime_r																													
localtime	localtime_r																													
rand	rand_r																													
readdir	readdir_r																													

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

NASA-STD-4009A

NASA-STD-4009A							
Section	Description	Requirement in this NASA Technical Standard (NTS)				Applicable (Yes or No)	If No, Enter Rationale
			strtok	strtok_r			
			tmpnam	tmpnam_r			
7.7	Hardware Abstraction Layer	[STRS-92] The STRS platform provider shall provide the STRS platform HAL documentation that includes the following: (1) For each method or function, its calling sequence, return values, an explanation of its functionality, any preconditions for using the method or function, and the postconditions after using the method or function. (2) Information required to address the underlying hardware, including the interrupt input and output, the memory mapping, and the configuration data necessary to operate in the STRS platform environment.					
8. External Command and Telemetry Interfaces							
8.	External Command and Telemetry Interfaces	[STRS-94] An STRS platform shall accept, validate, and respond to external commands.					
8.	External Command and Telemetry Interfaces	[STRS-95] An STRS platform shall execute external application control commands using the standardized STRS APIs.					
8.	External Command and Telemetry Interfaces	[STRS-107] An STRS platform provider shall document the external commands describing their format, function, and any STRS methods invoked.					
8.	External Command and Telemetry Interfaces	(STRS-96) The STRS infrastructure shall use the STRS_Query method to service external system requests for information and to provide telemetry data about an STRS application.					

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED