NASA TECHNICAL HANDBOOK **National Aeronautics and Space Administration** **Washington, DC 20546-0001**	**NASA-HDBK-4009** **Approved: 06-05-2014**

SPACE TELECOMMUNICATIONS RADIO SYSTEM (STRS) ARCHITECTURE STANDARD RATIONALE

MEASUREMENT SYSTEM IDENTIFICATION:
None.

APPROVED FOR PUBLIC RELEASE—DISTRIBUTION IS UNLIMITED

DOCUMENT HISTORY LOG

Status	Document Revision	Approval Date	Description
Baseline		06-05-2014	Initial Release

FOREWORD

This Handbook is published by the National Aeronautics and Space Administration (NASA) as a guidance document to provide engineering information; lessons learned; possible options to address technical issues; classification of similar items, materials, or processes; interpretative direction and techniques; and any other type of guidance information that may help the Government or its contractors in the design, construction, selection, management, support, or operation of systems, products, processes, or services.

This Handbook is approved for use by NASA Headquarters and NASA Centers, including Component Facilities and Technical and Service Support Centers.

This Handbook establishes the key rationale, explanatory material, and additional information to support NASA-STD-4009, Space Telecommunications Radio System (STRS) Architecture Standard. This architecture is a standard for reconfigurable communication transceiver developments among NASA missions.

NASA-STD-4009 strives to provide commonality among NASA radio developments to take full advantage of emerging software-defined radio (SDR) 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, corrections, or additions to this Handbook should be submitted via "Feedback" in the NASA Standards and Technical Assistance Resource Tool at https://standards.nasa.gov.

Original Signed By: ***06-05-2014***

Ralph R. Roe, Jr.
NASA Chief Engineer

Approval Date

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION **PAGE**

TABLE OF CONTENTS (Continued)

SECTION **PAGE**

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

SPACE TELECOMMUNICATIONS RADIO SYSTEM (STRS) ARCHITECTURE STANDARD RATIONALE

1. SCOPE

1.1 Purpose

The purpose of this Handbook is to present the rationale which underlays the requirements contained in NASA-STD-4009, Space Telecommunications Radio System (STRS) Architecture Standard, the companion document to this Handbook. Supporting examples and further descriptions for clarification of portions of NASA-STD-4009 are also provided. Answers prompted by questions from the Space Communications and Navigation (SCaN) Testbed partners, who created the first space implementation of STRS, are also included. As the Standard evolves, minor corrections and updates to obsolete information will be added to the Handbook. The Handbook is aimed at helping readers and implementers of NASA-STD-4009 understand the Standard.

NASA-STD-4009 provides an STRS overview, background, and detailed descriptions that might be useful to the reader not familiar with the STRS architecture.

1.2 Applicability

This Handbook is applicable to providing the rationale as well as additional information to NASA-STD-4009, which is a standard for reconfigurable communication transceiver developments among NASA missions.

This Handbook is approved for use by NASA Headquarters and NASA Centers, including Component Facilities and Technical and Service Support Centers. This Handbook may also apply to the Jet Propulsion Laboratory or to other contractors, grant recipients, or parties to agreements only to the extent specified or referenced in their contracts, grants, or agreements.

This Handbook, or portions thereof, may be referenced in contract, program, and other Agency documents for guidance. When this Handbook contains procedural or process requirements, they may be cited in contract, program, and other Agency documents for guidance.

2. APPLICABLE DOCUMENTS

2.1 General

The documents listed in this section are applicable to the guidance in this Handbook.

2.1.1 The latest issuances of cited documents shall apply unless specific versions are designated.

2.1.2 Non-use of specific versions as designated shall be approved by the responsible Technical Authority.

The applicable documents are accessible via the NASA Standards and Technical Assistance Resource Tool at https://standards.nasa.gov or may be obtained directly from the Standards Developing Organizations or other document distributors.

2.2 Government Documents

NASA

NASA-STD-4009	Space Telecommunications Radio System (STRS) Architecture Standard
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—2009-215478	Case Study: Using the OMG SWRADIO Profile and SDR Forum Input for NASA's Space Telecommunications Radio System
NASA/TM—2011-216948	Symbol Tables and Branch Tables: Linking Applications Together
NASA/TM—2011-217266	Space Telecommunications Radio System (STRS) Compliance Testing
NPR-7150.2	NASA Software Engineering Requirements

2.3 Non-Government Documents

RTCA, Incorporated

DO-178B	Software Considerations in Airborne Systems and Equipment Certification

2.4 Order of Precedence

This Handbook provides guidance for the rationale which underlays the requirements contained in NASA-STD-4009 but does not supersede nor waive established Agency requirements/guidance found in other documentation.

3. ACRONYMS AND DEFINITIONS

3.1 Acronyms and Abbreviations

API	application program interface
BIT	built-in test
BSP	board support package
C++	computer programming language
CORBA	Common Object Request Broker Architecture
COTS	commercial off the shelf
DLL	dynamic link library
DSP	digital signal processor
EDIF	electronic design interchange format
FPGA	field programmable gate array
GPM	general-purpose processing module
GPP	general purpose processor
GRC	Glenn Research Center
HAL	hardware abstraction layer
HDL	hardware description language
HID	hardware interface description
I/O	input/output
ID	identification, identifier
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
JTRS	Joint Tactical Radio System
N/A	not applicable
NASA	National Aeronautics and Space Administration
NPR	NASA Procedural Requirements
OE	operating environment
OMG	Object Management Group
OS	operating system
PIM	platform-independent model
POSIX	Portable Operating System Interface
PSE51	minimal real-time system profile 51, defined in IEEE Std 1003.13
PSM	platform-specific model
RF	radio frequency
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	real-time operating system
SCA	Software Communications Architecture
SCaN	Space Communications and Navigation
SDR	software-defined radio

SPM	signal-processing module
STD	standard
STRS	Space Telecommunications Radio System
SWaP	size, weight, and power
SWRADIO	software radio
UML	Unified Modeling Language
VHDL	VHSIC hardware description language
VHSIC	very high speed integrated circuits
W3C	World Wide Web Consortium (main international standards organization for the World Wide Web (abbreviated WWW or W3).
WF	Waveform
XML	Extensible Markup Language
XSD	XML 1.0 Schema Definition
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation

3.2 Definitions

Key terms and definitions are described in section 3.2 of NASA-STD-4009.

4. HIGH-LEVEL RATIONALE

The rationales for the STRS requirements in NASA-STD-4009 were derived from the Level 1 requirements in NASA/TM—2007-215042, STRS Architecture Goals/Objectives and Level 1 Requirements (summarized below), the restrictions of the space environment, and the use cases in NASA/TP—2008-214813, STRS Software Architecture Concepts and Analysis. The Object Management Group (OMG) Software Radio (SWRADIO) profile was considered in NASA/TM—2009-215478, Case Study: Using the OMG SWRADIO Profile and SDR Forum Input for NASA's Space Telecommunications Radio System, and the platform-independent model (PIM) was used as the starting point for the application software requirements.

STRS Goals and Objectives

4.1 Usable across most NASA mission types (scalability and flexibility).
4.2 Decrease development time and cost.
4.3 Increase reliability of software-defined radios (SDRs).
4.4 Accommodate advances in technology with minimal rework (extensibility).
4.5 Adaptable to evolving requirements (adaptability).
4.6 Enable over the air interoperability with existing assets (interoperability).
4.7 Leverage existing or developing standards, resources, and experience (state-of-the-art and state-of-practices).
4.8 Maintain vendor independence.
4.9 Enable waveform application portability.

STRS Level 1 Requirements

5.1 Layered Architecture.
5.2 Open Architecture.
5.3 Flexibility in Form Factor.
5.4 Remote Reconfiguration.
5.5 Remote Reprogrammability.
5.6 External Hardware Control.
5.7 Standard Spacecraft Interfaces.
5.8 Existing Waveform Support.
5.9 Multiple Waveform Support.
5.10 Simultaneous Operation of Multiple Waveforms.
5.11 Multi-Service Support.
5.12 Suitable for Any Radio Frequency Bands.
5.13 Multiple Frequency Bands
5.14 Multi-Channel Capability.
5.15 Commanded Built-In-Test and Status Reporting.
5.16 Operational Diagnostics.
5.17 Automated System Recovery/Initialization.
5.18 Navigation Support.
5.19 Network Support.
5.20 Security Compatibility.
5.21 Secure Transmission.
5.22 Processor Sharing.
5.23 Autonomous Link Optimization.

4.1 Operational Requirements

Many Level 1 requirements describe what the architecture has to allow in the way of operations. However, any mission needs to decide which of these are to be implemented in its specific radios to support the mission's needs. In NASA-STD-4009, requirements were written in a layered way so as to describe many architecture requirements in terms of applications, devices, services, other artifacts, and how they are used to perform the necessary functions.

The Level 1 requirements include responding to commands sent from an external source for remote reconfiguration, remote reprogrammability, processor sharing, and commanded Built-In-Test (BIT) and status reporting. These Level 1 requirements become part of the rationale for those requirements in NASA-STD-4009 pertaining to the startup and configuration of the radio as well as commanding the radio and obtaining information back about the configuration and command success.

4.2 Operating Environment (OE) Requirements

The STRS OE, as used in NASA-STD-4009, is the software environment in which the STRS applications are executed. Because an STRS radio is really a computer, it has an operating system (OS) usually created separately from the STRS infrastructure. Using an OS promotes the STRS goals and objectives for flexibility, decreasing development time and cost, and increasing

the reliability of SDRs, by leveraging existing standards, resources, and experience. A real-time operating system (RTOS) is likely to be used to meet timing deadlines and to support other operations, even though most of the real-time capability currently resides in field programmable gate arrays (FPGAs). Furthermore, an OS will need real-time capability if the general purpose processors (GPPs) are faster and assume more of the telecommunication functions or if the mission's telemetry requirements are stringent enough to warrant using an RTOS to be able to achieve processing constraints.

The Portable Operating System Interface (POSIX) is a standard that is used by many OSs. Therefore, POSIX was chosen to implement certain functions missing from the list of STRS Application-provided methods to minimize duplication of methods between STRS and POSIX.

4.3 Documentation Requirements

NASA-STD-4009 requires specific information to be delivered with an STRS-compliant platform and application. This information is requested to support the goals and objectives for extensibility, adaptability, portability, and vendor independence. The specific mission procuring the platform and/or application will require additional documentation to integrate and operate the platform and/or application with the rest of the system for the mission.

The hardware abstraction layer (HAL) documentation is required by STRS to allow, for example, the independence of the providers of the infrastructure and the FPGA. The basic goals and objectives include accommodating advances in technology with minimal rework (extensibility) and maintaining vendor independence. The STRS infrastructure has access to the HAL, and the HAL is specific to a platform. Since STRS is designed for the applications to be as portable as possible, STRS hides the HAL from the STRS application using a bridge pattern so that an STRS application can use a standardized application program interface (API) to interact with any specialized hardware.

4.4 Source Code Requirements

Because of size, weight, and power (SWaP) restrictions for space, many of the requirements in the Department of Defense's Software Communications Architecture (SCA) version 2.2 were eliminated for STRS. These include eliminating the requirement for Common Object Request Broker Architecture (CORBA) and the onboard parsing of the Extensible Markup Language (XML). Although object-oriented Unified Modeling Language (UML) diagrams were used for parts of the description to clarify relationships, they did not become part of the requirements. As described in the Case Study, STRS was able to use almost the same PIM as the OMG's SWRADIO to come up with a very different implementation.
Because the STRS architecture needed to support a C language interface to minimize SWaP, a pure object-oriented approach was unsatisfactory. To encapsulate functionality in a consistent manner, APIs were defined and the corresponding #include files were required to constrain the method signatures appropriately. A subset of the infrastructure APIs were required to call the appropriate application API. Because of the C language interface, the methods were differentiated so that the STRS infrastructure APIs had a different naming convention and two additional arguments:

a. The *fromWF* argument in many of the STRS infrastructure APIs is the handle identification (ID) used to indicate who the caller or source is.

b. The *toWF* argument in many of the STRS infrastructures APIs is the handle ID used to indicate who the responder or destination is. The STRS infrastructure uses the *toWF* to determine which application, device, file, or queue is to be used to perform any further processing. Then the infrastructure-provided method usually calls the corresponding Application-provided method.

A handle ID is used to control access to applications, devices, and so forth. The *fromWF* and *toWF* may be used by the infrastructure to validate whether the method is allowed to be called and to keep track of the history of the method call for error processing. They are used by the infrastructure when creating log messages to indicate the source of an error.

It's up to the infrastructure provider whether the handle ID is an index into a table, or an address of a structure, or a hash value used to look up the information that the infrastructure uses to access the application, device, file, or queue. The only restriction imposed by STRS is that a negative value indicates an error. It is the responsibility of the infrastructure to keep any additional information needed to intelligently populate the error logs. Each application is informed of its own handle ID and handle name using APP_Instance. The application uses its own handle ID as the *fromWF* argument to most STRS infrastructure-provided methods. There should be a handle ID for the infrastructure (or portions thereof) to indicate when the infrastructure is the source for error messages.

After considering the functionality required by the use cases and OMG SWRADIO profile, there needed to be APIs to standardize additional portions of the software. Therefore, APIs were created for devices, message queues, files, and timing. Some others were left to POSIX to implement.

4.5 Configuration File Requirements

Configuration files are required as a self-documenting way for the STRS infrastructure to start STRS applications into a known state to support the Level 1 requirements for remote partial configuration, reconfigurability, adaptability, automated system recovery and initialization, and multiple waveform support. The basic requirement derives from the need for each component to start in a known state. Requirement STRS-1 produces a known state for the platform. Using application configuration files produces a known state for the applications. The infrastructure configuration files are optional but are suggested to allow for evolution of the hardware and software in a standard way.

Using an application configuration file addresses the variability regarding what resources may be needed by the application, what variables need to be defined, and what state changes need to be made. There isn't necessarily a one-to-one correspondence between an application and a loadable file. A loadable file can contain parts of multiple applications; likewise, an application can span multiple loadable images. In a common case, a single application has both a GPP part and an FPGA part. In the case of an OS like Real-Time Executive for Multiprocessor Systems

(RTEMS), a free open-source RTOS designed for embedded systems, which does not support dynamic loading, there's no need to include separate information on the GPP part of an application, since the build process does not make a loadable image for the application alone, separate from the infrastructure.

The application configuration files are required to allow for the same application to be configured for different situations. For example, the environment may vary over the life of the radio and parameters may need to be adjusted accordingly. Another example might be allowing the original frequencies defined in a configuration file to be updated to avoid interference. It is less risky to send a short data file to the radio than to send a full software load. A current example for one of the SCaN Testbed SDRs is that the software and/or configurable hardware design has to remain stable at a certain point in the development process so that code review and test can be completed. However, there are parameters that are environmentally sensitive and can only be determined by testing under environmental conditions that simulate the conditions of space. These parameters are determined and entered into the application configuration file. A new application configuration file is then added to the SDR, but the software and configurable hardware design used for the code review and test remain unchanged.

When the waveform application is started, the state, resources, and the configurable parameters are specified in a configuration file so that any reinitialization is predictable. For example, if the radio needs to cycle power to correct some glitch, it should be able to do so and restart the application without intervention. STRS-101 requires the inclusion of "initial or default values for all distinct operationally configurable parameters." The "operationally configurable parameters" were those that could be configured using the STRS_Configure/APP_Configure commands instead of using subordinate parameters that don't have to be configured separately or using merely queryable parameters. For example, if a data item can be initialized in multiple ways, such as having parameters for both frequency and wavelength, only one would need to be configured. Also, a parameter could be queryable but not configurable (e.g., temperature, location, and power consumption).

The reasons for using XML for the predeployed application configuration files are as follows:

a. Using XML allows the data to have a standardized format that is easily created, read, and used by multiple entities.

b. XML is easy to learn, doesn't require much overhead, and is a W3C free and open standard.

c. Using XML saves time and money since tools already exist for displaying, editing, validating, parsing, and other functions.

d. Using XML allows the data to be self-documenting.

e. An exact format for the predeployed configuration files is not specified because the infrastructure for each vendor may need different information to start an application. The data in the application configuration files may not always be just name/value pairs.

f. Using XML allows for hierarchical as well as sequential data. Using hierarchical data allows for greater complexity. Using XML does not hinder the data from being used sequentially or in a specific order.

g. STRS-101 facilitated requiring specific information in the predeployed configuration file in XML that is easily transformed so that only the relevant subset appears in the deployed configuration file. The minimum data required are identification, any resources loaded, any parameters configured, and ending state information.

h. Using XML allows the STRS integrators to be able to verify that the data they enter meet some specified criteria so that the creation of the deployed configuration file will work properly.

i. The XML file is not intended for direct automatic processing. The items in STRS-101 are a somewhat minimal set of information that is useful for identifying and categorizing the application as well as for providing values for initialization. Since there is no content specified for the deployed configuration file, there can be no objection to categorizing the memory used and specifying the size in the XML file, and forcing the STRS integrator to do the computation to see if the application will fit. In the future, a particular platform could theoretically use the size information in the deployed format to help instantiate an application; however, that is not required by the architecture.

j. STRS began as an approach to adapt SCA for NASA space applications. A waveform application was implemented using SCA and from that, it was apparent that having CORBA and an XML parser in the radio added quite a lot to the complexity, size, and weight. STRS eliminated the need for CORBA, dynamic features, and an XML parser in the OE. So, the best resolution was having XML to start with and a processed file to deploy on the radio.

k. The SCA's properties files in XML could be used for STRS with minimal changes. An Extensible Stylesheet Language Transformation (XSLT) could be written to transform an SCA properties file into any deployed format.

l. Using XML allows validation of input values. Part of the compliance testing is to verify that the configuration file follows the format specified in the schema. Using an XML schema saves NASA from having to keep a different validation tool for each vendor.

m. Using XML allows data to be labeled with tags and attribute names so that the data are more easily validated and changed. The minimum was specified for the format of the application configuration files so that each vendor could use the format appropriate to that vendor's implementation. For example, here are some alternatives for a "wait" command in XML:

(1) <command>wait 100</command>
(2) <wait>100</wait>
(3) <wait delay="100"/>
(4) <wait delay="100" units="microseconds"/>
(5) <wait> <delay>100</delay> <units>microseconds</units> </wait>

Validation is difficult for alternative (1) using a schema. When "100" is separately identified, it is easier to validate. When "100" is labeled as a delay, it is easier for multiple entities to identify and modify. When the "100" is labeled as a delay in microseconds, it is even easier for multiple entities to identify and modify.

XML 1.0 is specified because the http://www.w3.org/ standards state that even though XML 1.1 is the current version, "You are encouraged to create or generate XML 1.0 documents if you do not need the new features in XML 1.1." Furthermore, when using XML version 1.1 in the commercial-off-the-shelf (COTS) product NASA used for testing, XMLSpy, an error was obtained.

To avoid the drawback of requiring an XML parser to be part of the radio, the STRS allows the configuration files to be preprocessed into a simpler form. Although XSLT is suggested in NASA-STD-4009 as a simple mechanism for this transformation, it is not required. Alternatively, the preprocessing could be handled by a program or script. Although a manual process using a text editor is not ruled out by requirement STRS-103, an automated process is preferred.

4.6 Roles and Responsibilities

For STRS, roles are specified as abstractions for the responsible organizations. The roles and corresponding organizations are expected to change at different stages of the radio's life cycle. For example, a developer or provider of some component may act as an STRS integrator for that component and other components at a subsequent stage of production. Then that STRS integrator may act as a provider for the next stage. NASA's goals are to promote vendor independence, scalability, flexibility, and extensibility, while specifying the smallest number of clearly defined roles possible.

The basic roles that NASA-STD-4009 defines are the STRS platform provider, who delivers a platform upon which STRS applications can be executed, an STRS application developer, who provides the desired functionality in the form of an STRS application, and an STRS integrator, who is responsible for integrating the parts to work together. The STRS platform provider could subcontract for hardware and software, but the responsibility for coordination, integration, and delivery of the infrastructure and related artifacts would reside in one STRS platform provider organization.

The STRS platform provider would usually act as STRS application developer and STRS integrator for at least a sample application. The roles and associated products are depicted in figure 1, Roles and Products.

The roles could have been broken down differently, allowing for various combinations of providers and integrators that could be very complex. Some suggested roles were as follows:

a. Application developer or provider.
b. Application integrator (OE + applications).
c. Configurable hardware design provider.

d. HAL/board support package (BSP)/drivers provider.
e. Hardware integrator.
f. Hardware parts supplier.
g. Infrastructure provider.
h. Kernel integrator (OS + POSIX + HAL).
i. OE integrator (OS + POSIX + HAL + infrastructure).
j. OE provider (OS + POSIX + HAL + infrastructure).
k. Operator.
l. OS provider.
m. Platform integrator.
n. Platform provider.
o. POSIX provider.
p. Radio integrator (OE + applications).
q. Radio operator (concerned with the mechanics of commanding the radio).
r. Spacecraft operator (concerned with the functionality of the radio in the larger. sense of including ground operations, experimenters; i.e., consumers of the data flowing through the radio).
s. System integrator (the entity that puts the radio into a system).

Some of these roles are duplicates. Some of these roles overlap. There were multiple interpretations for some of the roles, causing confusion. Therefore, the roles were simplified. Although a few operator roles were suggested, the operator roles have no STRS requirements, so these roles were not included. The operator roles are required for specific missions or projects rather than for the STRS architecture. There are no STRS requirements for specific external commands and how the commands and data get to the STRS radio. There are no STRS requirements for a specific process for an operator to turn the radio on and off, send configuration commands, consume and source data, deal with configuration management of software uploads, and other functions. There are no STRS requirements for the radio link parameters and for the actions of the experimenters and consumers of the data flowing through the radio. These types of requirements are mission-specific and have to be included in the requirements for the particular mission or project, which are in addition to NASA-STD-4009 requirements.

Document preparer roles and reviewer roles are not included, because STRS has no requirements concerning the process by which the documents are generated. There will be mission or project requirements for additional roles (including stakeholders) not mentioned here, for which STRS has no requirements.

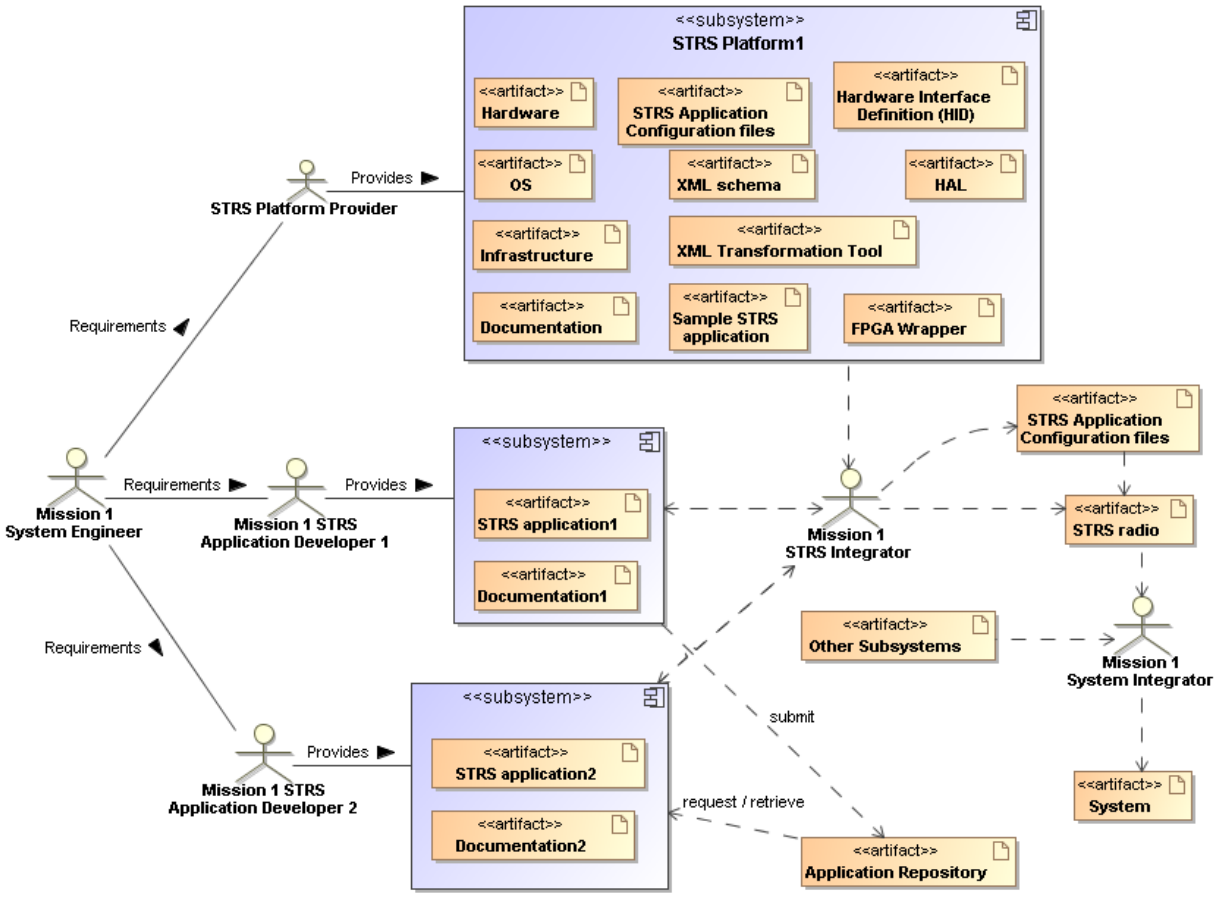

Figure 1—Roles and Products

5. HOW TO USE STRS APIs

This section contains recommendations and general information about how some operations would usually be performed using the STRS APIs.

5.1 How to Associate FPGA with an STRS Application

Since the FPGA(s) may be named differently on different platforms, the name of the FPGA used in an STRS application (GPP code) should be a configurable attribute, with its value set in the configuration files. Since the handle identifiers (IDs) are different on each platform, handle names should be used to obtain the handle ID with STRS_HandleRequest().

5.2 How to Load FPGA

An FPGA may be loaded directly by the infrastructure when it parses the configuration file, if supported, or may be loaded by an STRS_DeviceLoad call from the application GPP code. If the latter method is used, the name of the bitstream file should also be a configurable attribute set with the APP_Configure method. The STRS_HandleRequest method should be called to obtain the handle ID for the FPGA Device, and then the STRS_DeviceLoad method should be called for the FPGA to load the bitstream file. These STRS infrastructure calls may be performed in the APP_Configure directly or in the APP_Start method, as appropriate.

5.3 How to Set Attributes

An FPGA may be configured directly by the infrastructure when it parses the configuration file or by an STRS_Configure call to the STRS Device for the FPGA call from the application GPP code. If the latter method is used, the handle name of the FPGA Device should also be a configurable attribute set with the APP_Configure method. The STRS_HandleRequest method should be called to obtain the handle ID for the FPGA Device, and then the STRS_Configure method should be called for the FPGA Device to configure the FPGA. These STRS infrastructure calls may be performed in the APP_Configure directly or in the APP_Start method, as appropriate.

5.4 How to Get Attributes

Not all specialized hardware can be interrogated for its configuration; however, the infrastructure or the application may maintain any configuration data needed. The application has to implement APP_Query and, if appropriate, it should call the STRS_HandleRequest to obtain the handle ID for the FPGA followed by an STRS_Query call to the STRS FPGA Device to obtain the configuration data from the FPGA.

5.5 How to Push Packets

To push packets from an application, to a device, queue, file, or another application, STRS_Write is used. For generating packets in the same application used to send the packets, APP_Write may be used directly. If an application acts as a sink of packets pushed, it has to implement APP_Write, #include "STRS_Sink.h"; and if C++, the class has to implement STRS_Sink.

5.6 How to Pull Packets

To pull packets from a device, queue, file, or another application, STRS_Read is used. To pull packets from another module in the same application, APP_Read may be used directly. If an application acts as a source of packets pulled, it has to implement APP_Read, #include "STRS_Source.h", and, if C++, the class has to implement STRS_Source.

5.7 How to Process Errors

When a call to an STRS method is made, a variable of type STRS_Result is usually returned. To ensure consistent testing for errors, where an error is usually a negative value, STRS_IsOK tests that variable of type STRS_Result for errors, and returns a true or false boolean variable. The value returned from STRS_IsOK is true when there is no error and false when there is an error so that appropriate action may be taken.

When an error is detected in the operation of the application, STRS_Log should be invoked using an error queue handle ID (STRS_FATAL_QUEUE, STRS_ERROR_QUEUE, or STRS_WARNING_QUEUE), and a descriptive message. The error queue handle ID can be determined using STRS_GetErrorQueue with the error return value as an argument. The error queues are monitored and passed to the infrastructure for further action.

5.8 How to Make Multiple Instances of an Application

To create multiple instances of an application, be sure that the application is reentrant and that all pertinent data are configurable. Two configuration files have to be created, specifying a different handle name for each instance. When using C language applications, there may be method name duplication, which is discussed further in section 6.10, C Language Naming Duplication.

5.9 How to Map Memory Locations

Mapped memory locations should be defined in the configuration files related to the configurable attributes for the devices. The memory locations should not be hard-coded because hard-coding would limit the independence of the software and configurable hardware design. In the example shown for MEMORYMAP and MAPVALUE in NASA-STD-4009, Appendix A, there is a base name, associated location, and size. This is illustrated in figure 2, Memory Map. The individual item locations are specified relative to the base name in addressable storage units. The location may also have a bit offset and bit length. Then, when the configurable item is modified, the mapped location is used.

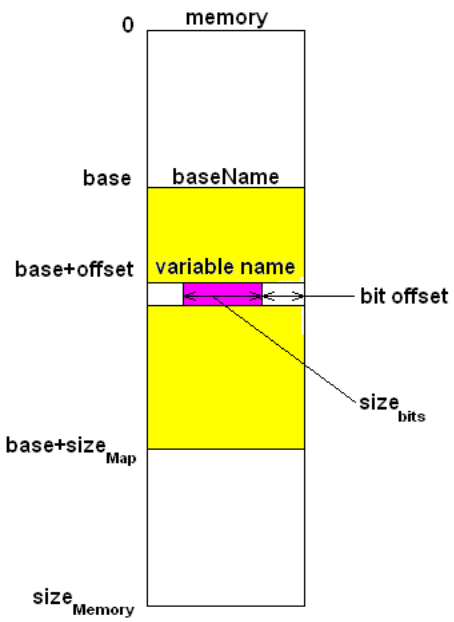

Figure 2—Memory Map

5.10 When to Use STRS_Log and STRS_Write

The two STRS infrastructure methods, STRS_Log and STRS_Write, have similar functionality except that STRS_Log adds a time stamp and possibly other identifying information. These methods should never be mixed for a given target. An STRS application developer should only write to the error queues using STRS_Log because the errors need to be identified further (STRS-54, STRS-55, STRS-56) and never with STRS_Write. Similarly, an STRS application developer should only write to the telemetry queues using STRS_Log because the telemetry data need to be identified further (STRS-57) and never using STRS_Write. Futhermore, the error queues are monitored for faults. An STRS application developer should only use STRS_Write to write buffered data to another application, service, device, file, or queue that does not require additional information added.

5.11 Difference Between Run Test and Ground Test

A run test is invoked using STRS_RunTest and implemented by APP_RunTest. A ground test is invoked using STRS_GroundTest and implemented by APP_GroundTest. A run test is invoked before or after deployment to determine whether the component is performing correctly. A ground test is generally invoked before deployment to perform unit testing and calibration. The ground tests help to automate and evaluate those tests. The term ground test was originally used to indicate testing for a satellite system, which is performed on the ground before launch.

Ground test may be invalid after deployment and indicates that such tests are normally completed before deployment and are not repeated thereafter. If allowed by the project and the ground tests will not be repeated after deployment, then the ground test code may be removed prior to deployment. The run tests and ground tests were separated because NASA generally requires significant testing prior to deployment; for example, vibration testing, environmental testing, radiation testing, etc.

5.12 When to use Start/Stop, Load/Unload, and Open/Close

The commands STRS_Start, STRS_DeviceLoad, STRS_DeviceOpen are specified in NASA-STD-4009 along with the reverse commands, STRS_Stop, STRS_DeviceUnload, and STRS_DeviceClose. The following describes the interaction of these commands under various common circumstances.

Initialize is used while the application is in the STRS_APP_STOPPED mode to set the application to a known initial condition. The application may be configured before and/or after initialize. Start is used to begin normal processing and change the state to STRS_APP_RUNNING. If any part of the application is in specialized hardware, that portion needs to be loaded before starting. To load an STRS Device, the Device has to be opened first. It is suggested that any part executing in specialized hardware not begin execution upon being loaded but rather during the start process. Similarly, it is suggested that stopping execution doesn't require any specialized hardware to be unloaded. Therefore, greater control is given to the application software for processing commands to start and stop waveform application operation in order to take advantage of windows of opportunity for execution as well as to promote consistency in control of the radio. Only certain allowed items may be configured after starting.

It is suggested that the STRS OE use configuration file(s) to start-up an application to a known initial state. STRS encourages that changeable data be specified in configuration files, rather than coding the data as constants within the application or device, so that greater portability and ease of modification is achieved. The STRS OE may process a configuration file to instantiate, open and load the device, initialize, configure, and start the application, or use any subset of these as determined by the project/mission and STRS platform provider.

As an example, the following use case is written for a waveform application using specialized hardware to send signals over the air to another radio assuming that the specialized hardware device has already been instantiated and initialized by the OE:

1. Radio receives a command that a new waveform application is needed. This may be multiple commands received or one command that invokes a series of operations. In either case, those operations follow.
2. OE checks for availability of the application and memory to instantiate it.
3. OE instantiates application (STRS_InstantiateApp).
4. OE initializes application (STRS_Initialize).
5. OE opens specialized hardware device (STRS_DeviceOpen).

6. OE loads specialized hardware device (STRS_DeviceLoad).
7. OE configures application (STRS_Configure).
8. OE starts application (STRS_Start).

The use case for the reverse process is as follows:

1. The radio receives a signal to stop and remove the application. This may be multiple commands received or one command that invokes a series of operations. In either case, those operations follow.
2. OE stops the application (STRS_Stop).
3. OE unloads the specialized hardware device (STRS_DeviceUnload).
4. OE closes the specialized hardware device (STRS_DeviceClose).
5. OE releases resources for application (STRS_ReleaseResources).

If there is no specialized hardware device, the steps pertaining to such a device may be eliminated. If the application merely performs calculation, start may mean perform the calculation and, before each calculation, the data is reconfigured. Alternately, start may mean ready to perform the calculation and start invokes a thread that loops waiting for new data so that each time new data is obtained, the computation is performed. Similarly, for a waveform application, start may mean to tell the specialized hardware device to begin processing signals or alternately, start may invoke a thread to perform the communication functions. A separate thread is used so that other commands may be processed independently.

6. QUESTIONS AND ANSWERS

The following questions have been asked by implementers of STRS-compliant platforms and applications and are included to provide additional insight for readers who might have similar questions.

6.1 Fault State and Use of the ERROR, WARNING, and FATAL Queues

NASA-STD-4009 does not specify how the fault state is set or detected. The fault state may be determined in a number of ways as specified by the mission or project. When STRS_Log sends a message to the error queue or fatal queue, it is assumed that there is an error or fatal error in that component and that the fault state is set accordingly. The fault state could also be set when a Health Manager or Watchdog Timer detects a problem (neither of which is required). The fault state could also be detected when the telemetry returns improper values. The fault state as shown in NASA-STD-4009 in the state diagram (figure 15, STRS Application State Diagram) implies that the radio detects and possibly recovers; however, it could be designed so that the fault state is kept by the flight computer or equivalent.

The use of the ERROR, WARNING, and FATAL queues are expected to be defined more closely by the mission or project. The three relevant types of queues are as follows:

a. The STRS_FATAL_QUEUE is the queue used when an STRS_FATAL error is encountered. STRS_FATAL_QUEUE denotes the queue for a nonrecoverable error in an attempt to capture information about the situation in a logging trail used to reconstruct the original cause of the error. Furthermore, sending a message to the STRS_FATAL_QUEUE is one way of initiating an orderly shutdown and reboot of the radio to a known state. The processing for a fatal error could imply turning off the heartbeat; that is, rebooting the radio and, if that doesn't work, reloading the software and/or configurable hardware design. It could imply running additional diagnostic tests. It is up to the mission to define whether there are alternative ways of rebooting under different circumstances, such as after trying three times. It may make a difference if the problem is overheating or if a bit has been changed in the radio so that it doesn't work properly.

b. STRS_ERROR_QUEUE denotes the queue for a recoverable error. The most likely reason is an invalid set of configuration parameters. The recovery would be to get a valid set of configuration parameters.

c. STRS_WARNING_QUEUE denotes the queue for a recoverable error that has little or no effect on the operation of the radio. The most likely reasons are trying to run a test in a state for which the test is not allowed, trying to configure or query a parameter when the value is not available in that state, or trying to run APP_Start when the application is already started.

6.2 Message Queues Need Clarification

The STRS doesn't require implementing messaging queues using the observer/publish-subscribe design pattern where the class inherits a notify method. The STRS is designed to work in C without inheritance, but the idea is that the publisher doesn't know the identity of the subscriber such that one or more applications or devices can funnel data to one or more different applications, devices, files, or queues. Figure 3, Sample Publisher-Subscriber Sequence Diagram, is just one possibility for a sequence diagram showing the creation and possible use of a messaging queue using one form of the publisher-subscriber paradigm:

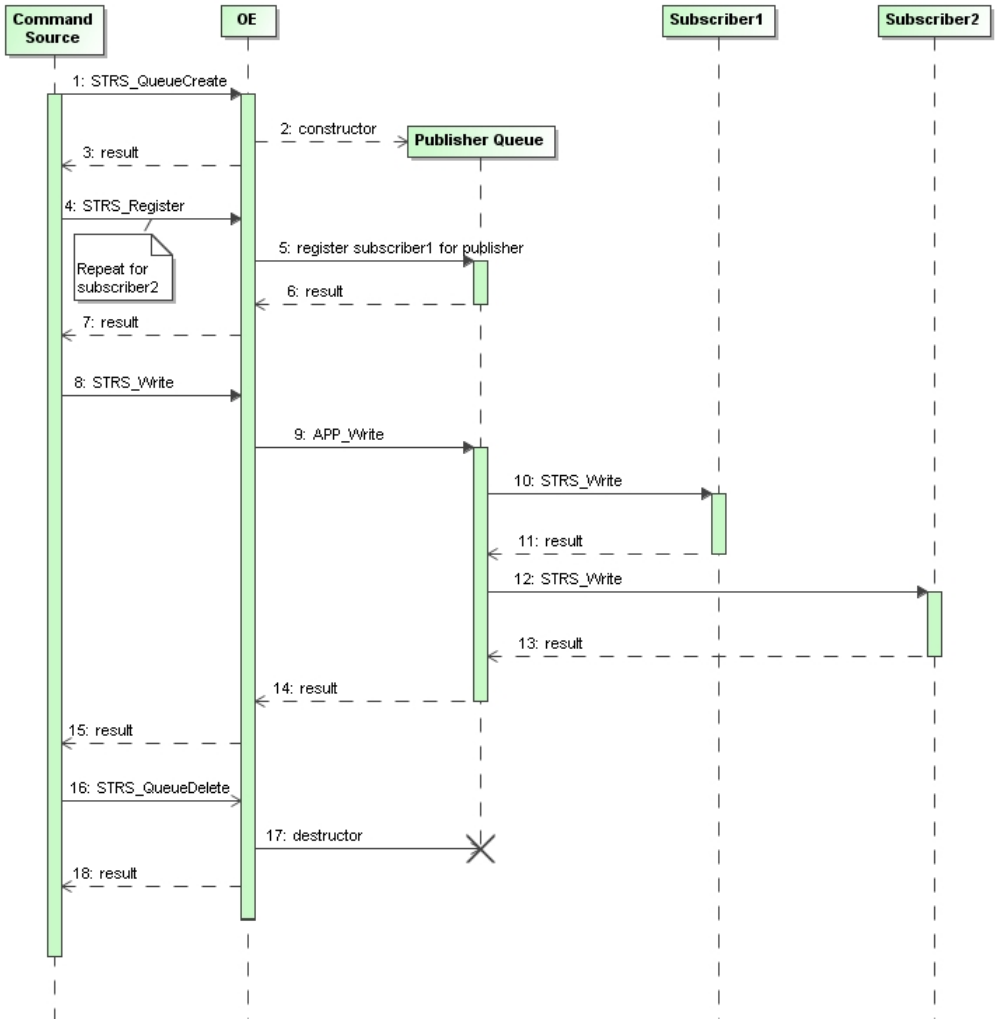

Figure 3—Sample Publisher-Subscriber Sequence Diagram

Detecting circularity and duplication is difficult with just the sequence diagram shown without adding additional methods. Circularity is where the message published eventually ends up back at the original publisher and is sent again in an infinite loop. Duplication is where the message published ends up at the same destination twice.

There is a problem of notification when it is a SIMPLE queue. STRS_Write will put the message on the queue, but when does STRS_Read obtain it from the queue and when does the message get deleted from the queue? Also, can the queue fill up so that further messages are rejected? The resolution to the message-queuing behavior is not included in the current version of NASA-STD-4009 but has to be covered by the specific design for the mission or project.

6.3 What is an STRS Device?

An STRS Device is shorthand for an entity that responds to both STRS application methods and STRS Device methods. For example, an FPGA may be implemented as an STRS Device such that it may be loaded, configured, started, stopped, unloaded, and so forth, using the corresponding STRS infrastructure Device Control API. There is no requirement that an STRS Device actually exists as separate software. It could be implemented as a subsystem within the infrastructure without defining a separate class. In NASA-STD-4009, section 7.3.6, STRS Infrastructure Device Control API, STRS Devices are discussed but are not required. There was only the suggestion that they be implemented as shown in figure 4, STRS Application/Device Structure, which is a copy of figure 14, STRS Application and Device Structure, of NASA-STD-4009.

The STRS infrastructure Device Control API is used instead of POSIX input/output (I/O) methods such as open, close, read, write, and so forth, to promote portability by standardizing the device interface for hardware devices where POSIX methods cannot be implemented. Currently, the POSIX methods can be used when all the device methods are in the same application. Another reason the Device Control API is required is to align with the Joint Tactical Radio System (JTRS)/SCA and the OMG/SWRADIO, which had devices of various types defined. The methods load and unload, shown in figure 4, were adapted from those standards. Because of the C language interface requirement, inheritance for various types of devices is not practical. Also, accessing the STRS Devices using a handle ID gave more flexibility in configuring a data source or sink. For example, an application might be used to transmit data over the air obtained from a data source that might be configured as an application, device, queue, or file. Similarly, an application might be used to receive data over the air and send it to a data sink that might be configured as an application, device, queue, or file. Thus, an STRS Device may be used either to distribute functionality over multiple waveform applications or to abstract hardware functionality, further giving greater flexibility. An STRS Device is similar to the bridge pattern used to separate an abstraction from its underlying implementation.

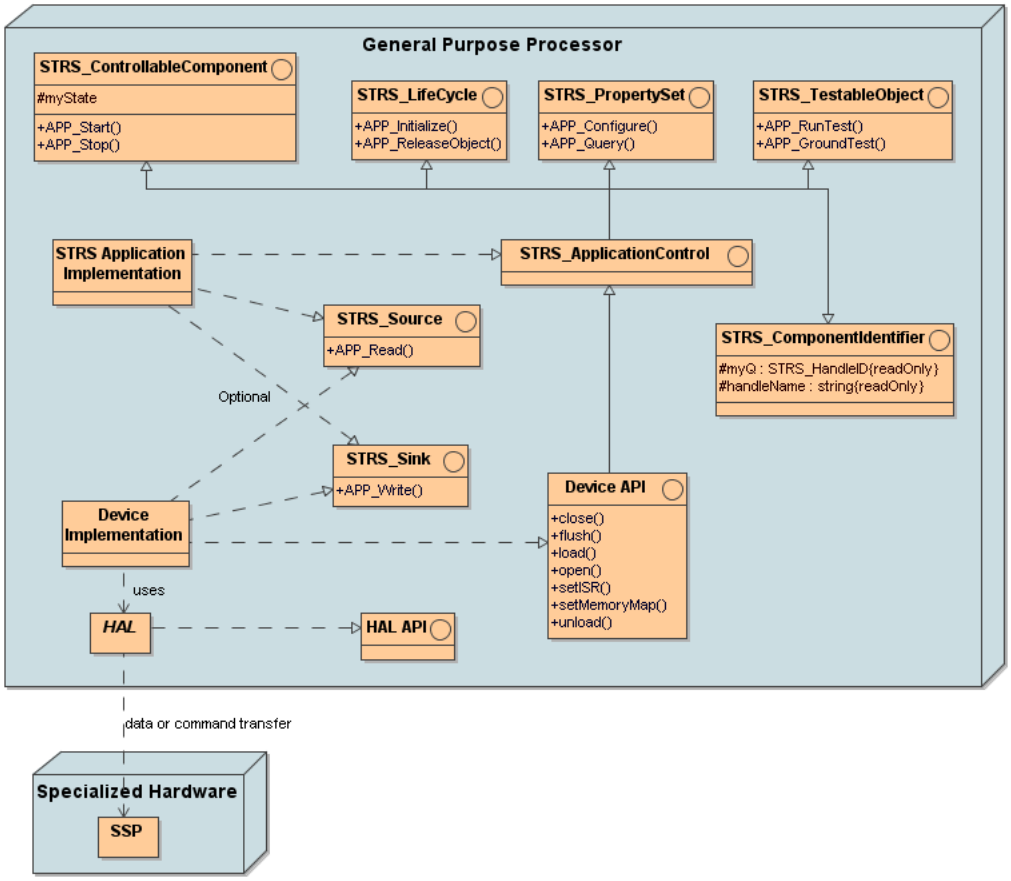

Figure 4—STRS Application/Device Structure

6.4 How to Configure and Control SDR Hardware?

The configuration and control of the SDR hardware depends on where the intelligence exists; that is, which software component knows how to configure and control SDR hardware? A combination of software components (waveform application, STRS infrastructure, STRS Device, and HAL) knows how to configure and control SDR hardware.

a. The application knows about data values, the STRS Device knows about mappings in the GPP, and the HAL knows about how to take values and transfer them to the hardware.

b. The STRS application should be the target component for the parameters it controls and could pass to an STRS Device those parameters that need to be passed to the HAL.

c. STRS Application.
(1) The STRS application would control what data are configured.

(2) The STRS application has limited intelligence on how and where to enable the application to be portable.

(3) The STRS application knows the handle ID of the STRS Device to use.

(4) For example, an STRS application processes FREQUENCY, converts from floating point to integer in a format that is recognized by the STRS Device, and calls the appropriate method to configure the STRS Device.

d. STRS Device.
(1) The STRS Device controls how data get to the FPGA or other hardware

(2) The STRS Device knows how to send data to the proper register in the FPGA using the HAL.

(3) The HAL may be external functions or inline functions that know the mappings from data addresses to registers in the FPGA.

e. STRS Infrastructure.
(1) The STRS infrastructure reads the configuration files or receives an external command and calls the STRS_Configure method for the appropriate target component.

(2) The STRS_Configure in the infrastructure calls the corresponding APP_Configure within the target component.

6.5 STRS Infrastructure Methods Do Not Belong to Any Class

The STRS infrastructure-provided methods beginning with “STRS_” do not belong to any class, since they have to be the same when called from C language implementations. If one is coding in C++, these methods should be defined using extern "C" {...}.

In NASA-STD-4009, the STRS infrastructure provides the STRS infrastructure-provided Application Control API that supports application operation using the STRS Application-provided Application Control API in section 7.3.1. The STRS Infrastructure-provided Application Control API methods (section 7.3.2) that begin with “STRS_” correspond to the STRS Application-provided Application Control API methods (section 7.3.1) that begin with “APP_” and are used to access those methods. The STRS infrastructure implements these methods for use by any STRS application or by any part of the infrastructure that is desired to be implemented in a portable way.

Since the C language is optional for STRS applications (see STRS-16, 18, 19), the STRS Application-provided application control methods beginning with “APP_” may belong to a class.

6.6 Explain Clocks and Timers

Clocks/timers are used for the following:

a. Coordinating external events.

b. Providing time for real-time functions in the GPP.
   (1) Some functions currently in FPGA will be transitioned from FPGA to GPP when the GPPs are fast enough and capable enough to handle the signal processing functionality.

NASA-STD-4009 is designed to allow a clock/timer to be an extension of an STRS Device so that the functionality can be embedded in specialized hardware, if necessary. Multiple timers are only defined when they are required by the mission. An offset is usually specified to ensure that the clock is monotonically increasing from a previous power reset or is synchronized with another clock/timer.

Normally each clock/timer has a base time, usually measured from when it is turned on. An offset may be used to keep the time monotonically increasing with each power cycle. An offset may also be used to coordinate with external events. The timing of external events, such as another satellite coming over the horizon or the availability of experimenters, may be used to power parts of the radio off and on so that the radio optimizes its power consumption and availability.

6.7 FPGA Partial Reconfiguration

Partial reconfiguration is the process of configuring selected areas of an FPGA after its initial configuration. Xilinx indicates that a bitstream file can contain all the configuration commands and data necessary for partial reconfiguration. Therefore, STRS_DeviceLoad will work, and no new methods need to be defined at this time.

6.8 Compliance Testing

STRS compliance of a vendor- or partner-provided SDR is assessed by source code inspection, document inspection, configuration file inspection, adding an application containing a reference to each STRS infrastructure method and testing that application. The name of that application is the STRS Command and Compliance, also known as WFCCN. WFCCN may be compiled with an STRS infrastructure to determine whether or not there are any missing constants, typedefs, or structs.

Since many of the STRS requirements are source code requirements, a standard test suite cannot test them fully. Since STRS is designed to allow multiple vendors to work together, certain source code artifacts have to be made available so that a subsequent STRS application developer or STRS integrator can use the methods, constants, typedefs, and structs required. The following is an example of a problem that WFCCN cannot be used to find: One vendor used noncompliant

method signatures with *int* instead of STRS_Buffer_Size, but on that platform, both integer items compiled as the same type. Using a type that happens to correspond to the vendor's implementation of an STRS type is not necessarily portable to the next platform.

The STRS compliance could be evaluated at and by each vendor or partner and the results shared and discussed in one or more workshops at various points in the project life cycle. This alternative is to be decided by the mission or project. A full release and delivery of all STRS OE source code is not required in order to perform STRS compliance testing. Each vendor or partner should inspect his or her own software and documents before delivery. However, NASA found noncompliances in the deliverables for each SCaN Testbed partner.

Once the STRS radio artifacts are tested for STRS compliance, any noncompliances will be reported to the supplier and the mission or project, along with any suggestions. It is the responsibility of the mission or project to decide whether to grant deviations and waivers for any noncompliances that are not resolved.

6.9 Configuration Files Examples

To help the reader and implementer of the STRS architecture understand the development and use of the configuration files described in NASA-STD-4009, Appendix A, Example Configuration Files, an example of a configuration file based on that format was developed. This example of a configuration file, for a sample application WF1, is shown in this Handbook in figure 5, Example of Predeployed Configuration File for NASA-STD-4009, Appendix A.

```
1  <?xml version="1.0" ?>
2  <?xml-stylesheet type="text/xsl" href="STRS.xsl"?>
3  <!-- ?xml version="1.0" encoding="UTF-8"? -->
4  <STRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="STRS.xsd">
5    <CONFIGURATION>
6      <!-- One waveform is defined. -->
7      <WAVEFORM>
8        <WFHANDLENAME>WF1a</WFHANDLENAME>
9        <WFNAME>WF1</WFNAME>
10       <WFACCESS>NONE</WFACCESS>
11       <WFSTATE>INSTANTIATED</WFSTATE>
12       <!-- The waveform is comprised of two files:
13     WF1.out for the GPP, and
14     WF1.bit for the FPGA. -->
15       <LOADFILE>
16         <LOADFILENAME>%STRS_BASE%\WF1\%STRS_TARGET%\%STRS_TARGET%\WF1.out</LOADFILENAME>
17         <LOADTARGET>SELF</LOADTARGET>
18         <LOADMEMORY>
19           <MEMORYSIZE><INTEGER>136546</INTEGER></MEMORYSIZE>
20           <MEMORYUNITS>BYTES</MEMORYUNITS>
21         </LOADMEMORY>
22         <LOADTHREADTYPE>POSIX</LOADTHREADTYPE>
23         <LOADTHREADTAG>W1</LOADTHREADTAG>
24         <LOADTHREADPRIORITY>50</LOADTHREADPRIORITY>
25       </LOADFILE>
26       <LOADFILE>
27         <LOADFILENAME>%STRS_BASE%\WF1\%STRS_TARGET%\WF1.bit</LOADFILENAME>
28         <LOADTARGET>FPGA</LOADTARGET>
29         <LOADMEMORY>
30           <MEMORYSIZE><INTEGER>2733252</INTEGER></MEMORYSIZE>
31           <MEMORYUNITS>BYTES</MEMORYUNITS>
32         </LOADMEMORY>
33       </LOADFILE>
34       <ATTRIBUTE>
35         <NAME>A</NAME>
36         <VALUE>5</VALUE>
37       </ATTRIBUTE>
38       <ATTRIBUTE>
39         <NAME>B</NAME>
40         <VALUE>27</VALUE>
41       </ATTRIBUTE>
42       <ATTRIBUTE>
43         <NAME>C</NAME>
44         <VALUE>Non-numeric</VALUE>
45       </ATTRIBUTE>
46     </WAVEFORM>
47     <INCLUDE>%STRS_BASE%\WF2\WF2_%STRS_TARGET%.cfg</INCLUDE>
48   </CONFIGURATION>
49 </STRS>
```

Figure 5—Example of Predeployed Configuration File for NASA-STD-4009, Appendix A

Here is the explanation, line by line:

(1) XML declaration.
(2) Extensible Stylesheet Language (XSL) file declaration.
(3) Comment.
(4) Open tag STRS and corresponding XML schema declaration.
(5) Open tag CONFIGURATION.
(6) Comment.
(7) Open tag WAVEFORM.

(8) Tag WFHANDLENAME containing WF1a as the handle name.
(9) Tag WFNAME containing WF1 as the class name.
(10) Tag WFACCESS with WF1 having no READ/WRITE access; that is, neither APP_Read nor APP_Write are implemented.
(11) Tag WFSTATE with final state for WF1a as STRS_APP_INSTANTIATED.
(12) Comment.
(13) Comment.
(14) Comment.
(15) Open tag LOADFILE.
(16) Tag LOADFILENAME containing the path to load WF1.out.
(17) Tag LOADTARGET containing SELF to indicate that it is loaded on the current GPP.
(18) Open tag LOADMEMORY.
(19) Tag MEMORYSIZE indicating that the size is 134K bytes.
(20) Tag MEMORYUNITS indicating that the size is measured in bytes.
(21) Close tag LOADMEMORY.
(22) Tag LOADTHREADTYPE to indicate that POSIX threads are used.
(23) Tag LOADTHREADTAG containing W1 as the name of the thread.
(24) Tag LOADTHREADPRIORITY containing 50 as the priority of the thread.
(25) Close tag LOADFILE.
(26) Open tag LOADFILE.
(27) Tag LOADFILENAME containing the path to WF1.bit.
(28) Tag LOADTARGET containing FPGA to indicate that it is loaded on the FPGA.
(29) Open tag LOADMEMORY.
(30) Tag MEMORYSIZE indicating that the size is 2670K bytes.
(31) Tag MEMORYUNITS indicating that the size is measured in bytes.
(32) Close tag LOADMEMORY.
(33) Close tag LOADFILE.
(34) Open tag ATTRIBUTE.
(35) Tag NAME containing A.
(36) Tag VALUE containing 5 as the value for A.
(37) Close tag ATTRIBUTE.
(38) Open tag ATTRIBUTE.
(39) Tag NAME with B.
(40) Tag VALUE containing 27 as the value for B.
(41) Close tag ATTRIBUTE.
(42) Open tag ATTRIBUTE.
(43) Tag NAME with C.
(44) Tag VALUE containing “Non-numeric” as the value for C.
(45) Close tag ATTRIBUTE.
(46) Close tag WAVEFORM.
(47) Tag INCLUDE containing the path to additional items to process.
(48) Close tag CONFIGURATION.
(49) Close tag STRS.

The example in NASA-STD-4009, Appendix A, splits the formatting into three parts to group the information logically. The following explanations further clarify the necessity and intent of Appendix A.

a. There is no necessity or requirement for splitting the platform configuration files into hardware and software parts as shown in NASA-STD-4009, Appendix A.1 and A.2. Splitting up the description this way was just a logical way to organize the description.

b. In NASA-STD-4009, section 9.3, Application Configuration Files, says "…the format of the application configuration file should be a subset of the format of the platform configuration file." The purpose is to allow some applications and services to be instantiated at boot-up or restart. The application configuration file format as a subset of the platform configuration file format is shown in Appendix A.3 of NASA-STD-4009. However, the schema would normally have the platform configuration information pruned to create an independent application configuration file.

6.10 C Language Naming Duplication

There will most likely be more than one application in an STRS radio. In the C language, there is no namespace support as in C++ or other object-oriented (OO) languages that scope the member functions to the class. Thus, there will be multiple implementations of the same STRS Application-provided API method names, starting with "APP_", one in each implementation of an application.

One technique to allow multiple "instances" of C language applications could use APP_Instance to return a pointer to a table of pointers to the methods. Then, the OE could use these method locations to call the methods. This technique and variations are described in NASA/TM—2011-216948, Symbol Tables and Branch Tables: Linking Applications Together. The techniques specify the creation of a branch table or indirect address table for each application. To suppress the common method names, compile and link each application separately. When the table is registered with the OE, the OE could use the table to call the appropriate method.

One technique to allow multiple "instances" of C language applications with the same method names depends on loading new applications one at a time, sequentially, and capturing the new method locations instead of the old at the appropriate point in the process. The method locations are saved in a structure associated with the STRS application and the appropriate method is called as needed. The flow chart shown in figure 6, Obtain Array of Pointers to Methods, gives some highlights. Note that, besides the method illustrated in figure 6**,** there are other ways of creating an array of pointers to the C language methods. This array of pointers may be used to invoke those methods later.

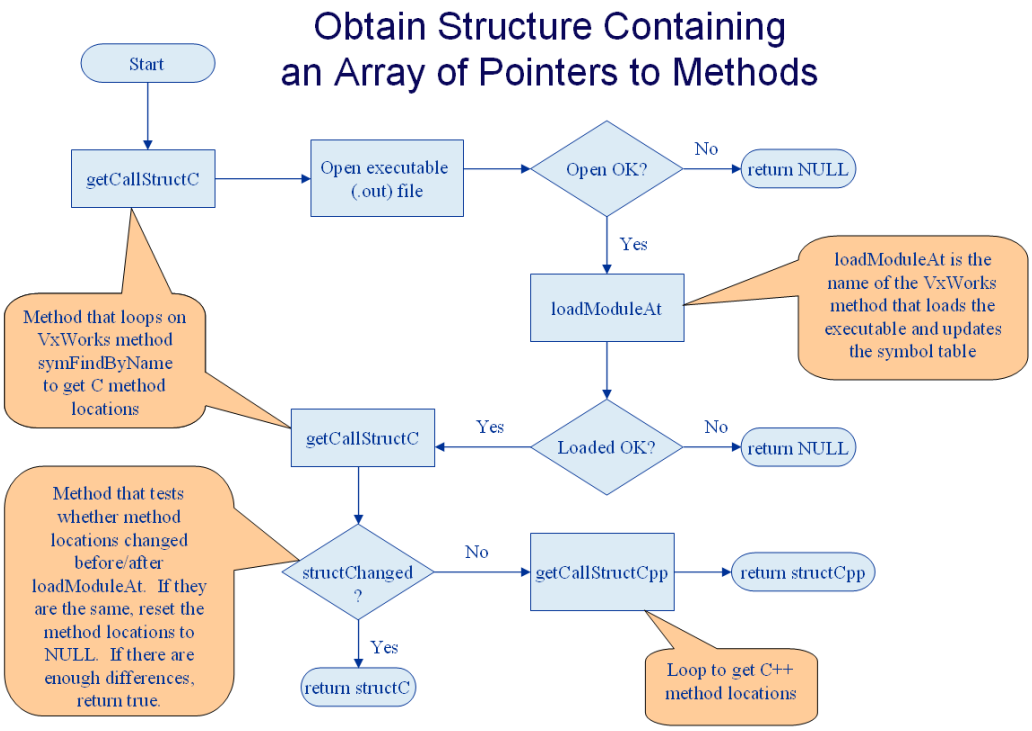

Figure 6—Obtain Array of Pointers to Methods

The technique demonstrated in figure 6 might not be possible on a platform that needs everything to be compiled and linked together ahead of time. Another technique would be to use message queuing to communicate between independent applications, but this technique might be awkward to use in practice.

6.11 Sequence Diagrams Depicting STRS API Calls

The following sequence diagrams depict the relationship between the STRS infrastructure-provided Application Control API beginning with "STRS_" and the corresponding STRS Application-provided Application Control API beginning with "APP_." The methods described in NASA-STD-4009 for figure 15, STRS Application State Diagram, are those that cause a change in state. In this Handbook, the methods depicted in figure 7**,** Simplified Sequence Diagram for STRS_InstantiateApp, and figure 8, Simplified Sequence Diagram for STRS_AbortApp, and figure 9, Simplified Sequence Diagram for STRS_Configure, contain both those that cause a change in state as well as those that do not. In this Handbook, since an STRS Device inherits all the methods from an STRS application, as shown in figure 4**,** the methods in figures 7, 8, and 9 for STRS applications could apply to STRS Devices as well. In figures 7, 8, and 9, "Command Source" is used for the object, internal to the radio, either an STRS application or part of the OE, which calls the STRS infrastructure methods.

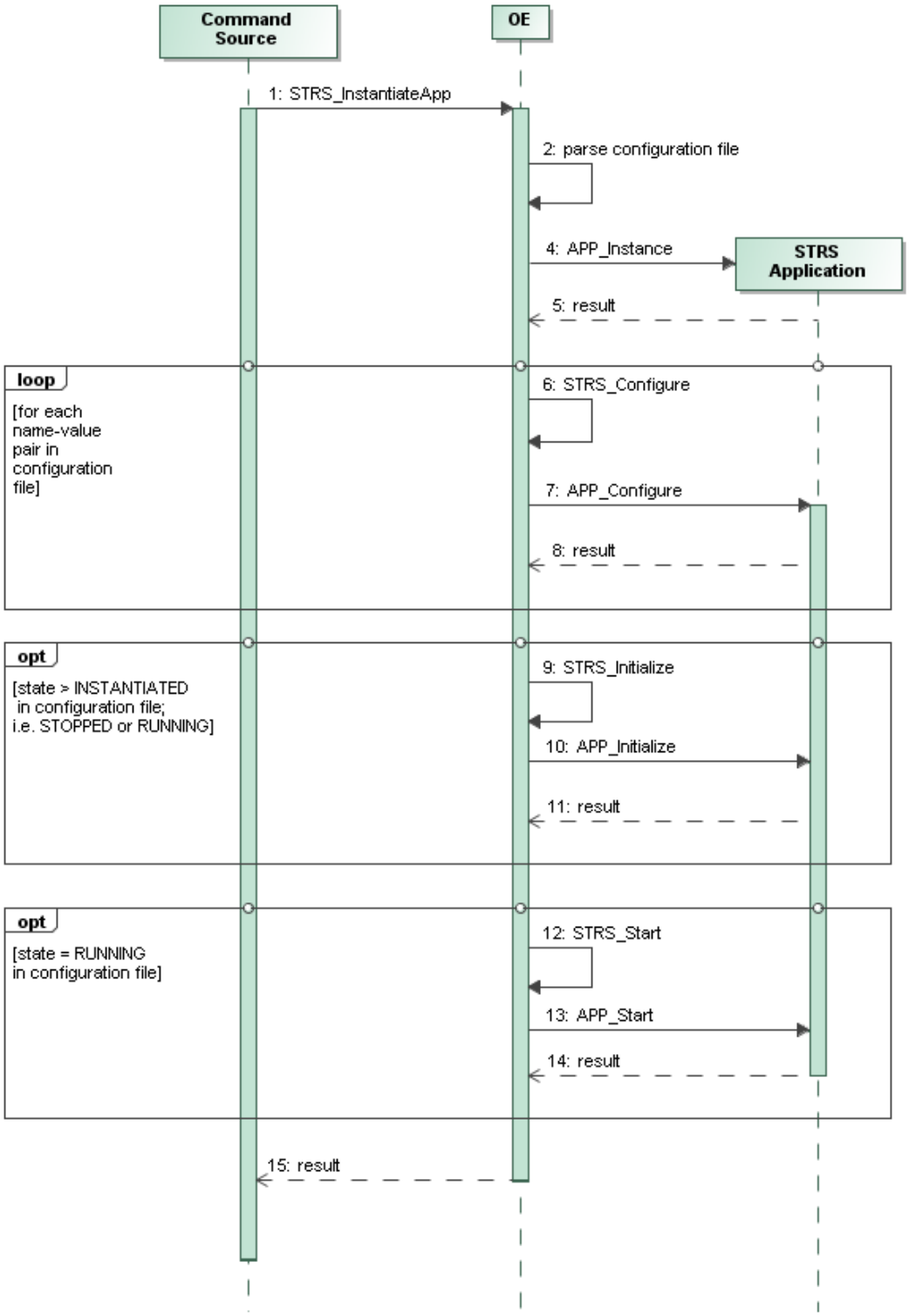

Figure 7—Simplified Sequence Diagram for STRS_InstantiateApp

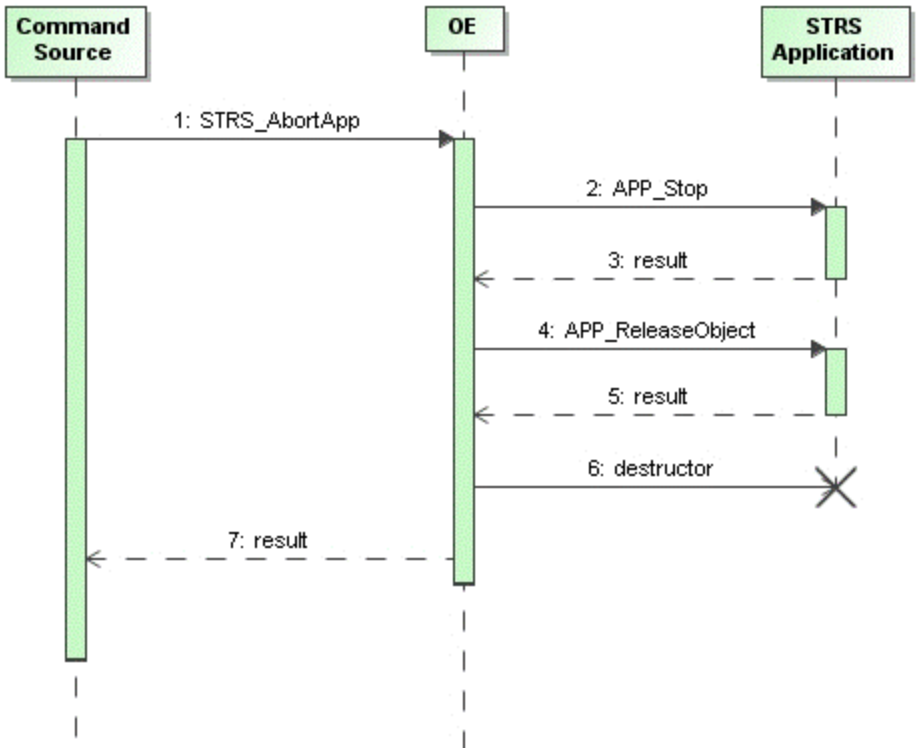

Figure 8—Simplified Sequence Diagram for STRS_AbortApp

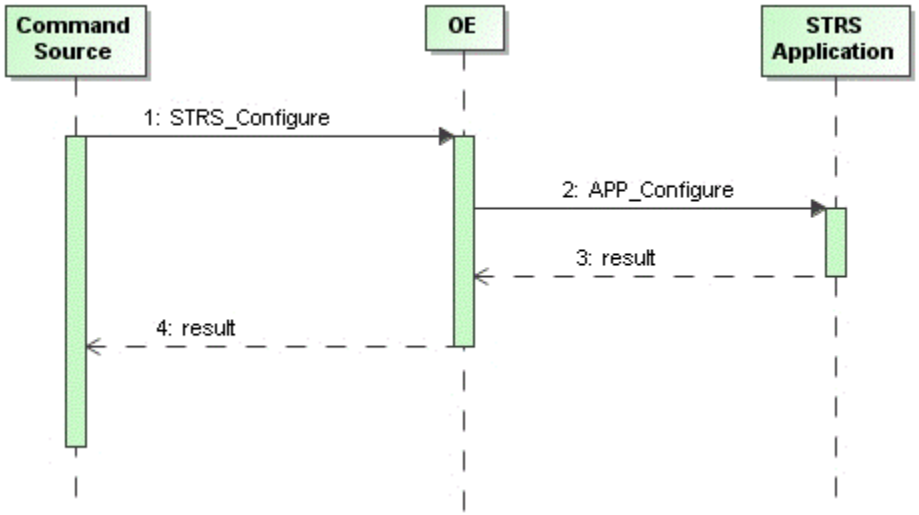

Figure 9—Simplified Sequence Diagram for STRS_Configure

A sequence diagram for each row in table 1, Substitutions for Figure 9, can be made from the diagram in figure 9, by substituting the "COMMAND SOURCE TO OE" method in place of STRS_Configure and the corresponding "OE TO STRS APPLICATION" method in place of APP_Configure:

Table 1—Substitutions for Figure 9

COMMAND SOURCE TO OE	OE TO STRS APPLICATION
STRS_GroundTest	APP_GroundTest
STRS_Initialize	APP _Initialize
STRS_Query	APP _Query
STRS_Read	APP _Read
STRS_ReleaseObject	APP _ReleaseObject
STRS_RunTest	APP _RunTest
STRS_Start	APP _Start
STRS_Stop	APP _Stop
STRS_Write	APP _Write

6.12 Why are APP_Instance and APP_Initialize Separate?

The APP_Instance and APP_Initialize methods are often used together successively but should not be combined because they have different functionality. The separation of APP_Instance and APP_Initialize supports encapsulation. It allows configuration to occur before APP_Initialize. In figure 7, STRS_InstantiateApp calls APP_Instance and then it may call APP_Configure and APP_Initialize, as specified by the configuration file. STRS_InitializeApp may do everything in one call or additional calls may be needed thereby giving the greatest flexibility. Also note that APP_Instance is a convenience function containing a constructor and saving any application identifying information.

6.13 Why Start with SCA?

The Wireless Innovation Forum (formerly SDR Forum) and SWRADIO by the OMG put so much effort into SCA, that it was decided to investigate these architectures. The result of that investigation was that the CORBA requirements and XML parser requirements took up a lot of memory and machine cycles but were not really necessary for NASA. One study showed that eliminating CORBA and an XML parser reduced the memory footprint by an order of magnitude. So, to save on SWaP for NASA's space platforms, it was decided to create a similar STRS architecture without those disadvantages. After looking at use cases for NASA radios, very similar functionality to the SCA and SWRADIO was decided to be necessary. Similar method names to the application method names in SCA and SWRADIO were chosen for STRS. The reasoning was that it would be easy to take advantage of the many man-years of effort that had gone into defining those architectures, the Wireless Innovation Forum could comment on the STRS architecture due to the similarities, and that SDR design tools might be easier to use/generate for STRS applications.

A key recommendation from the Forum's space working group was to align with the OMG SWRADIO specification where possible. To that end, mappings from the OMG SWRADIO PIM to the STRS platform-specific model (PSM) were discussed. There were only minor differences in the Space PIM that could map into STRS from OMG's SWRADIO PIM that mapped into SCA. A quote from the Forum's study:

> *The SDR Forum recommended that the STRS align with the SDR Forum, the OMG, and the IEEE SCC41 for purposes of distributing the burden and cost of non-recurring engineering (NRE) across NASA and all consortia members contributing to the STRS, and to further broaden and enhance the quality of the implementation and deployment of STRS-based standards.*

NASA's configuration files could be much simpler, because NASA's radios were less distributed with no dynamic aspects needed to be specified in the configuration files. Furthermore, it was decided that by preprocessing any XML configuration files, a much simpler parser could be used on much simpler data.

6.14 Security for STRS

Security aspects need to be considered for any STRS radio. There are currently no STRS requirements for security, and it is assumed to be up to the project/mission to define any security requirements. It was determined that NASA radios typically do not require DO-178B, Software Considerations in Airborne Systems and Equipment Certification; and red/black separation. Security is needed to:

a. Verify/validate external commands such that:
   (1) They come from the appropriate source (e.g., using data in a CCSDS wrapper).

   (2) They have not been compromised (e.g., using encryption, signing, parity bit, checksum, cyclic redundancy check).

   (3) They are in the appropriate format for commands.

   This is usually defined by the command and control for the mission and not by STRS. The security functions should be encapsulated, separate from the external command and control interpreter, so that the functionality may be changed if necessary, without affecting the STRS application implementation. This functionality may be invoked by the STRS OE implementation or as a service for over-the-air command and control.

b. Verify/validate internal commands; i.e., don't allow the radio to try to do anything risky or make itself inoperable.
   (1) Don't allow radio to call methods for which the handle ID is inappropriate.

   (2) Restricting the radio as to what it can do is left as a possible project/mission requirement. For example, one might restrict one waveform from aborting any other waveform. In this case, it is suggested that a table be configured containing allowed

or disallowed commands and the associated source(s) handle names and target(s) handle names for which the command applies such that the table could be used to validate a command.

(3) Specifying a key to allow the radio to override the restrictions of item "b(2)" is also left as a possible project/mission requirement. In which case, security keys and an authentication method is required.

Security requirements are defined by the project/mission and not by STRS. The security functions of item "b(2)" and item "b(3)" should be encapsulated so that the functionality may be changed if necessary, without affecting the STRS application implementation. This functionality may then be invoked by the STRS infrastructure implementation.

6.15 What is Configurable Hardware Design?

The term "configurable hardware design" is used throughout the STRS documentation to signify the items required to capture the digital logic of the hardware that can be configured remotely, such as an FPGA. Configurable hardware design includes the items created to document the design of the hardware including the source code (e.g., very high speed integrated circuits (VHSIC) hardware description language (VHDL), Verilog) and the loadable files (e.g., FPGA image).

The term "configurable hardware design" replaces the commonly used term "firmware" in earlier versions of STRS documentation. Many definitions for firmware, including the latest IEEE definition, which was written in 1990, state that firmware resides in read-only memory and cannot be modified. The changing definition of "firmware" would likely lead to confusion, and a new term was selected for the STRS documentation. Additional terms such as "complex electronics," "configurable logic device," "programmable logic device" and "software" were considered, but each term was rejected due to potential confusion or implied limitations if the term was used.

The SDR community is unique in that it uses GPPs and configurable hardware design in a single application. The term "software" in some contexts of the STRS (and other SDR-related) documentation may include configurable hardware design. For example, whenever the term software defined radio is used, both GPP and configurable hardware design are included. The STRS architecture does not dictate processes or organizational structure for use in developing the software or configurable hardware design. The project developing the SDR or application has to dictate the required process.

7. STRS REQUIREMENTS, RATIONALE, AND VERIFICATION METHOD

The following sections address each requirement in turn, displaying the rationale, the related higher-level requirements, verification method, and other pertinent information, which augment the general rationale given earlier in this document.

In each section, the title line contains the requirement number and the title of the requirement. That is followed by the text of the requirement. The *rationale* describes why the requirement is needed. The *category* contains one or more of the summary capabilities from NASA/TM—2007-215042. The categories are chosen from the following list: adaptability, availability, extensibility, flexibility, interoperability, portability, scalability, reliability, and reconfigurability. The *traced-from* specifies the section numbers in NASA/TM—2007-215042 that apply to this requirement. The *use case* specifies the names of the use case sections in NASA/TP—2008-214813, STRS Software Architecture Concepts and Analysis, that apply to this requirement. The *related to* specifies the part of the STRS radio that has to satisfy the requirement where *platform* indicates that the hardware and related documentation are tested, *OE* indicates that the OS and infrastructure and related documentation are tested, and *application* indicates that the application and related documentation are tested. The *notes* contains additional explanations for the requirement.

Verification methods are used to show that the requirement has been met. The verification method is chosen from the following list: Analysis, inspection, observation, similarity, or test. Tests are not used because tests are expected to be mission requirements rather than STRS requirements.

a. Analysis is the process of utilizing analytical techniques to verify that requirements have been satisfied. This method may be used when qualification by test is not possible, when a test would introduce significant risk into the software, or when analysis is an appropriate, cost-effective qualification method.

b. Inspection is a qualification method consisting of investigation without the use of special tests. Inspection is usually a visual examination, but it may be computer-aided. Using a script or WFCCN refers to the STRS compliance tools as described in the NASA/TM—2011-217266**,** STRS Compliance Testing document. A compliance certification testing facility is available at Glenn Research Center (GRC) to perform compliance testing and will test all STRS applications submitted to the STRS application repository. The users may use their own tools as an independent check of an OE or of an application prior to submitting the application to the STRS application repository.

(1) Using a script or WFCCN is a type of inspection that is computer-aided.
(2) Using a compliance tool implies a script or WFCCN.
(3) Using a program, such as XMLSpy, validates the XML schema and the predeployed configuration file against its schema.

c. Observation is a method of qualification that is limited to readily observable functional operation to determine compliance with requirements. This method of qualification does not require the use of special equipment or sophisticated instrumentation.

d. Similarity is the process of using analysis and/or “delta testing” to prove the design adequacy of an item by reference to the prior qualification of an identifiable item that has been qualified for a similar application.

e. Test is a qualification method that employs technical means including, but not limited to, the evaluation of functional characteristics by the use of special equipment or instrumentation, simulation techniques, and the application of established principles and procedures to determine compliance with requirements. The analysis of data derived from a test is an integral part of the method.

7.1 STRS-1 Power Up

Requirement	An STRS platform shall have a known state after completion of the power-up process.
Rationale	To increase the reliability of the STRS platform after reboot or power cycle, the radio has to be able to return to full operation autonomously without the need for external equipment or procedures.
Category	Availability, Reliability
Traced-from	4.3, 5.17
Use Case	Power On
Related to	OE
Notes	A known state is one that is predictable from documentation or from configuration file(s) or some combination thereof.
Verification Method	Observation of radio operation.

7.2 STRS-2 Provide Platform Diagnostics

Requirement	The STRS OE shall access each module's diagnostic information via the STRS APIs.
Rationale	To increase the reliability and availability of the STRS platform, there has to be a means of providing data to identify configuration information as well as status and fault identification. Data for both BITs and recognition of operational degradation and malfunction has to be available.
Category	Reliability, Availability
Traced-from	4.3, 5.15, 5.16
Use Case	Fault Management, Built-In Test
Related to	OE
Notes	None
Verification Method	Observation of radio operation.

7.3 STRS-3 Use Platform Diagnostics

Requirement	Self-diagnostic and fault-detection data shall be created for each module so that it is accessible to the STRS OE.
Rationale	To increase reliability and availability of the STRS platform, there has to be a means of providing data to identify configuration information as well as status and fault identification. Data for both Built-In-Tests and recognition of operational degradation and malfunction have to be available.
Category	Reliability, Availability
Traced-from	5.15, 5.16
Use Case	Fault Management, Built-In Test
Related to	OE
Notes	None
Verification Method	Observation of radio operation.

7.4 STRS-4 Document Resources

Requirement	The STRS platform provider shall describe, in the HID document, the behavior and capability of each major functional device or resource available for use by waveforms, services, or other applications (e.g., FPGA, GPP, DSP, or memory), noting any operational limitations.
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Adaptability
Traced-from	4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.5 STRS-5 Document Capability

Requirement	The STRS platform provider shall describe, in the HID document, the reconfigurability behavior and capability of each reconfigurable component.
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Adaptability
Traced-from	4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.6 STRS-6 Document Radio Frequency (RF) Behavior

Requirement	The STRS platform provider shall describe, in the HID document, the behavior and performance of the RF modular component(s).
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Interoperability, Adaptability
Traced-from	4.6, 4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.7 STRS-7 Document Module Interfaces

Requirement	The STRS platform provider shall describe, in the HID document, the interfaces that are provided to and from each modular component of the radio platform.
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Interoperability, Adaptability
Traced-from	4.2, 4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.8 STRS-8 Document Module Control

Requirement	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).
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Interoperability, Adaptability
Traced-from	4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.9 STRS-9 Document Power

Requirement	The STRS platform provider shall describe, in the HID document, the behavior and performance of any power supply or power converter modular component(s).
Rationale	Waveform developers need to know the features and limitations of the platform for their applications. Once the radio has been procured, NASA has the knowledge to procure or produce new or additional modules using HID information. Also, future module replacement or additions will be possible without designing a new platform.
Category	Reliability, Adaptability
Traced-from	4.7, 4.8, 5.2, 5.3
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of HID document.

7.10 STRS-10 STRS Application Uses OE

Requirement	An STRS application shall use the infrastructure STRS API and POSIX API for access to platform resources.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. Where possible, use currently available standards. Thus POSIX subsets were chosen to implement certain OS services missing from the list of STRS Application-provided methods The POSIX subsets are widely available, implemented by multiple OSs, and scalable. Layering of the architecture separates and encapsulates functionality so that the parts are less influenced by changes to the other. This separation of functionality promotes portability.
Category	Portability
Traced-from	4.2, 4.7, 5.1
Use Case	None
Related to	Application
Notes	Notes
Verification Method	Inspection using compliance tool.

7.11 STRS-11 OE Uses HAL

Requirement	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.
Rationale	The HAL API is to be published so that specialized hardware made by one company may be integrated with the STRS infrastructure made by a different company. The HAL API documentation is to include a description of each method or function used, including its calling sequence, return values, an explanation of its functionality, any preconditions before using the method or function, and the status after using the method or function. The HAL API documentation is to also contain information about the underlying hardware such as address and data interfaces, interrupt input and output, power connections, and other control and data lines necessary to operate in the STRS platform environment.
Category	Adaptability, Extensibility
Traced-from	4.4, 4.5, 5.1, 5.2
Use Case	None
Related to	OE
Notes	See also STRS-92.
Verification Method	Inspection of HAL document.

7.12 STRS-12 STRS Application Repository

Requirement — The following application development artifacts shall be submitted to the NASA STRS application repository.

(1) High-level system or component software model.
(2) Documentation of application configurable hardware design external interfaces (e.g., signal names, descriptions, polarity, format, data type, and timing constraints).
(3) Documentation of STRS application behavior.
(4) Application function sources (e.g., C, C++, header files, VHSIC VHDL, and Verilog).
(5) Application libraries, if applicable (e.g., electronic design interchange format (EDIF) and Dynamic Link Library (DLL)).
(6) Documentation of application development environment and tool suite.
   A. Include application name, purpose, developer, version, and configuration specifics.
   B. Include the hardware on which the application is executed, its OS, OS developer, OS version, and OS configuration specifics.
   C. Include the infrastructure description, developer, version, and unique implementation items used for application development.
(7) Test plans, procedures and results documentation.
(8) Identification of software development standards used
(9) Version of NASA-STD-4009.
(10) Information, along with supporting documentation, required to make the appropriate decisions regarding ownership, distribution rights, and release (technology transfer) of the application and associated artifacts.
(11) Version Description Document or equivalent with version numbers defined down to the lowest level components.
(12) Documentation of the platform component hardware used by the application, its function and the interconnections. If the component executes an operating system, document the OS, OS developer, OS version, and OS configuration.

Category — Portability

Traced-from — 4.2, 4.9, 5.2

Use Case — None

Related to — Application

Notes	See also STRS-92.
Verification Method	Inspection of deliverable items and documentation.

7.13 STRS-13 OE Controls Signal-Processing Module (SPM)

Requirement	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.
Rationale	The layering of the architecture introduces the need for the GPP to be able to control, configure, and monitor many aspects of the SPM. For portability, waveform applications use STRS APIs, which access the HAL or POSIX API within the STRS OE as needed.
Category	Portability, Reconfigurability, Adaptability
Traced-from	4.5, 4.9, 5.1, 5.4, 5.22
Use Case	None
Related to	Application
Notes	None
Verification Method	Observation of operation of radio.

7.14 STRS-14 Provide Platform-Specific Wrapper

Requirement	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.
Rationale	To aid in the portability of waveform applications within an FPGA, a platform-specific wrapper provides an additional layer separating the interface between the GPP and SPM/FPGA from the signal processing functionality within the FPGA.
Category	Portability, Extensibility
Traced-from	4.2, 4.4, 4.9, 5.1, 5.8
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of document and code.

7.15 STRS-15 Document Platform-Specific Wrapper

Requirement	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.
Rationale	When functions, interfaces, components, and/or design rules are defined and published, the architecture is open. Open architecture facilitates interoperability among commercial and government developers and minimizes the operational impact of upgrading hardware and software components.
Category	Portability, Adaptability
Traced-from	4.2, 4.4, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection of document.

7.16 STRS-16 Use C/C++ WF Interface

Requirement	The STRS Application-provided Application Control API shall be implemented using ISO/IEC C or C++.
Rationale	Because portability is a basic goal but middleware is not required, a totally language-independent solution was not available. The lowest common denominator turns out to be a C or C++ language interface. Using a standard ISO/IEC 9899 C or ISO/IEC 14882 C++ aids portability. The year is not included in the requirement, so that obsolete compilers are not mandated.
Category	Portability, Scalability
Traced-from	4.1, 4.2, 4.7, 4.9, 5.1
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection of code.

7.17 STRS-17 OE Uses STRS Application Control API

Requirement	The STRS infrastructure shall use the STRS Application-provided Application Control API to control STRS applications.
Rationale	Layering of the architecture separates and encapsulates functionality so that the parts are less influenced by changes to the other. This separation of functionality promotes portability.
Category	Portability
Traced-from	4.1, 4.2, 4.7, 4.9, 5.1
Use Case	None
Related to	OE
Notes	The STRS Application-provided Application Control API refers to the API defined in STRS-29 through STRS-39 and the corresponding tables 5 through 15. The method names in the STRS Application-provided Application Control API begin with “APP_”.
Verification Method	Inspection using OE script and WFCCN.

7.18 STRS-18 Use C/C++ Compile-Time

Requirement	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.
Rationale	Because portability is a basic goal but middleware is not required, a totally language-independent solution was not available. The lowest common denominator turns out to be a C or C++, or both, language interface. Using a standard ISO/IEC 9899 C or ISO/IEC 14882 C++ aids portability. The year is not included in the requirement, so that obsolete compilers are not mandated.
Category	Portability
Traced-from	4.1, 4.2, 4.7, 4.9, 5.1
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection

7.19 STRS-19 Use C/C++ Run-Time

Requirement	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.
Rationale	Because portability is a basic goal but middleware is not required, a totally language-independent solution was not available. The lowest common denominator turns out to be a C or C++, or both, language interface. Using a standard ISO/IEC 9899 C or ISO/IEC 14882 C++ aids portability. The year is not included in the requirement, so that obsolete compilers are not mandated.
Category	Portability
Traced-from	4.1, 4.2, 4.7, 4.9, 5.1
Use Case	None
Related to	OE
Notes	None
Verification Method	Use WFCCN.

7.20 STRS-20 Include STRS_ApplicationControl.h

Requirement	Each STRS application shall contain: *#include "STRS_ApplicationControl.h"*.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection using compliance tool.

7.21 STRS-21 Provide STRS_ApplicationControl.h

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection

7.22 STRS-22 STRS_ApplicationControl Base Class

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection

7.23 STRS-23 Include STRS_Sink.h

Requirement	If the STRS application provides the APP_Write method, the STRS application shall contain: *#include "STRS_Sink.h"*.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection using compliance tool.

7.24 STRS-24 Provide STRS_Sink.h

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection

7.25 STRS-25 STRS_Sink Base Class

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection

7.26 STRS-26 Include STRS_Source.h

Requirement	If the STRS application provides the APP_Read method, the STRS application shall contain: *#include "STRS_Source.h"*.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection using compliance tool.

7.27 STRS-27 Provide STRS_Source.h

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection

7.28 STRS-28 STRS_Source Base Class

Requirement	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.
Rationale	For portability, standard names are defined for various constants, data types, and method prototypes in the API.
Category	Portability
Traced-from	4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection

7.29 STRS-29 APP_Configure

Requirement	Each STRS application shall contain a callable APP_Configure method as described in table 5, APP_Configure().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_Configure was patterned after the configure method in the PropertySet interface in JTRS/SCA and OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Set Waveform Parameter
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.30 STRS-30 APP_GroundTest

Requirement	Each STRS application shall contain a callable APP_GroundTest method as described in table 6, APP_GroundTest().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_GroundTest was patterned after the runTest method in the TestableObject interface in JTRS/SCA and OMG/SWRADIO. It performs system and unit testing usually done before deployment.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3, 5.15
Use Case	Built-In Test
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.31 STRS-31 APP_Initialize

Requirement	Each STRS application shall contain a callable APP_Initialize method as described in table 7, APP_Initialize().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_Initialize was patterned after the initialize method in the LifeCycle interface in JTRS/SCA and OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Waveform Instantiation
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.32 STRS-32 APP_Instance

Requirement	Each STRS application shall contain a callable APP_Instance method as described in table 8, APP_Instance().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Waveform Instantiation
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool

7.33 STRS-33 APP_Query

Requirement	Each STRS application shall contain a callable APP_Query method as described in table 9, APP_Query().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_Query was patterned after the query method in the PropertySet interface in JTRS/SCA and OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Get Waveform Parameter
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.34 STRS-34 APP_Read

Requirement	If the STRS application provides data to the infrastructure, then the STRS application shall contain a callable APP_Read method as described in table 10, APP_Read().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Transmit a Packet
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.35 STRS-35 APP_ReleaseObject

Requirement	Each STRS application shall contain a callable APP_ReleaseObject method as described in table 11, APP_ReleaseObject().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_ReleaseObject was patterned after the releaseObject method in the LifeCycle interface in JTRS/SCA and OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Waveform Deallocation
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.36 STRS-36 APP_RunTest

Requirement	Each STRS application shall contain a callable APP_RunTest method as described in table 12, APP_RunTest().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_RunTest was patterned after the runTest method in the TestableObject interface in JTRS/SCA and OMG/SWRADIO. It performs system and unit testing usually done after deployment.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3, 5.15
Use Case	Built-In Test
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.37 STRS-37 APP_Start

Requirement	Each STRS application shall contain a callable APP_Start method as described in table 13, APP_Start().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_Start was patterned after the start method in the Resource interface in JTRS/SCA and ControllableComponent interface in OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Waveform Start
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.38 STRS-38 APP_Stop

Requirement	Each STRS application shall contain a callable APP_Stop method as described in table 14, APP_Stop().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. In addition, APP_Stop was patterned after the stop method in the Resource interface in JTRS/SCA and ControllableComponent interface in OMG/SWRADIO.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Waveform Stop
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.39 STRS-39 APP_Write

Requirement	If the STRS application receives data from the infrastructure, then the STRS application shall contain a callable APP_Write method as described in table 15, APP_Write().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.3
Use Case	Receive a Packet
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using compliance tool.

7.40 STRS-40 STRS_Configure

Requirement	The STRS infrastructure shall contain a callable STRS_Configure method as described in table 16, STRS_Configure().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Set Waveform Parameter
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.41 STRS-41 STRS_GroundTest

Requirement	The STRS infrastructure shall contain a callable STRS_GroundTest method as described in table 17, STRS_GroundTest().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately. It performs system and unit testing usually done before deployment.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.15
Use Case	Built-In Test
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.42 STRS-42 STRS_Initialize

Requirement	The STRS infrastructure shall contain a callable STRS_Initialize method as described in table 18, STRS_Initialize().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Instantiation
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.43 STRS-43 STRS_Query

Requirement	The STRS infrastructure shall contain a callable STRS_Query method as described in table 19, STRS_Query().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Get Waveform Parameter
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.44 STRS-44 STRS_ReleaseObject

Requirement	The STRS infrastructure shall contain a callable STRS_ReleaseObject method as described in table 20, STRS_ReleaseObject().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Get Waveform Parameter
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.45 STRS-45 STRS_RunTest

Requirement	The STRS infrastructure shall contain a callable STRS_RunTest method as described in table 21, STRS_RunTest().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately. It performs system and unit testing usually done after deployment.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2, 5.15
Use Case	Built-In Test
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.46 STRS-46 STRS_Start

Requirement	The STRS infrastructure shall contain a callable STRS_Start method as described in table 22, STRS_Start().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Start
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.47 STRS-47 STRS_Stop

Requirement	The STRS infrastructure shall contain a callable STRS_Stop method as described in table 23, STRS_Stop().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms. The signature of the infrastructure method is different from the signature of the corresponding application method because there has to be a C language interface to the infrastructure method and it has to contain additional information that allows the infrastructure to determine whether the target component is C or C++ and call the corresponding application method appropriately.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Start
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.48 STRS-48 STRS_AbortApp

Requirement	The STRS infrastructure shall contain a callable STRS_AbortApp method as described in table 24, STRS_AbortApp().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to implement and use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Abort
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.49 STRS-49 STRS_GetErrorQueue

Requirement	The STRS infrastructure shall contain a callable STRS_GetErrorQueue method as described in table 25, STRS_GetErrorQueue().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.50 STRS-50 STRS_HandleRequest

Requirement	The STRS infrastructure shall contain a callable STRS_HandleRequest method as described in table 26, STRS_HandleRequest().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.51 STRS-51 STRS_InstantiateApp

Requirement	The STRS infrastructure shall contain a callable STRS_InstantiateApp method as described in table 27, STRS_InstantiateApp().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Instantiation
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.52 STRS-52 STRS_IsOK

Requirement	The STRS infrastructure shall contain a callable STRS_IsOK method as described in table 28, STRS_IsOK().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.53 STRS-53 STRS_Log

Requirement	The STRS infrastructure shall contain a callable STRS_Log method as described in table 29, STRS_Log().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Fault Management
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.54 STRS-54 STRS_Log Error

Requirement	When an STRS application has a nonfatal error, the STRS application shall use the callable STRS_Log method as described in table 29, STRS_Log(), with a target handle ID of constant STRS_ERROR_QUEUE.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Fault Management
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.55 STRS-55 STRS_Log Fatal

Requirement	When an STRS application has a fatal error, the STRS application shall use the callable STRS_Log method as described in table 29, STRS_Log(), with a target handle ID of constant STRS_FATAL_QUEUE.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Fault Management
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.56 STRS-56 STRS_Log Warning

Requirement	When an STRS application has a warning condition, the STRS application shall use the callable STRS_Log method as described in table 29, STRS_Log(), with a target handle ID of constant STRS_WARNING_QUEUE.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Fault Management
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.57 STRS-57 STRS_Log Telemetry

Requirement	When an STRS application needs to send telemetry, the STRS application shall use the callable STRS_Log method as described in table 29, STRS_Log(), with a target handle ID of constant STRS_TELEMETRY_QUEUE.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.58 STRS-58 STRS_Write

Requirement	The STRS infrastructure shall contain a callable STRS_Write method as described in table 30, STRS_Write().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Receive a Packet
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.59 STRS-59 STRS_Read

Requirement	The STRS infrastructure shall contain a callable STRS_Read method as described in table 31, STRS_Read().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability, Extensibility
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Transmit a Packet
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.60 STRS-60 Device Control

Requirement	The STRS applications shall use the methods in the STRS infrastructure Device Control API, STRS infrastructure-provided Application Control API, Infrastructure Data Source API (if appropriate), and Infrastructure Data Sink API (if appropriate) to control the STRS Devices.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	The STRS infrastructure Device Control API refers to the API defined in STRS-61 through STRS-69 and corresponding tables 32 through 40. The STRS infrastructure-provided Application Control API refers to the API defined in STRS-40 through STRS-47 and corresponding tables 16 through 23. The STRS infrastructure Data Source API refers to the API defined in STRS-59 and corresponding table 31. The STRS infrastructure Data Sink API refers to the API defined in STRS-58 and corresponding table 30. The method names in the STRS infrastructure-provided APIs begin with “STRS_”.
Verification Method	Inspection

7.61 STRS-61 STRS_DeviceClose

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceClose method as described in table 32, STRS_DeviceClose().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.62 STRS-62 STRS_DeviceFlush

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceFlush method as described in table 33, STRS_DeviceFlush().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.63 STRS-63 STRS_DeviceLoad

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceLoad method as described in table 34, STRS_DeviceLoad().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.64 STRS-64 STRS_DeviceOpen

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceOpen method as described in table 35, STRS_DeviceOpen().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.65 STRS-65 STRS_DeviceReset

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceReset method as described in table 36, STRS_DeviceReset().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.66 STRS-66 STRS_DeviceStart

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceStart method as described in table 37, STRS_DeviceStart().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.67 STRS-67 STRS_DeviceStop

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceStop method as described in table 38, STRS_DeviceStop().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Stop
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.68 STRS-68 STRS_DeviceUnload

Requirement	The STRS infrastructure shall contain a callable STRS_DeviceUnload method as described in table 39, STRS_DeviceUnload().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have t use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Deallocation
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.69 STRS-69 STRS_SetISR

Requirement	The STRS infrastructure shall contain a callable STRS_SetISR method as described in table 40, STRS_SetISR().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.70 STRS-70 STRS_FileClose

Requirement	The STRS infrastructure shall contain a callable STRS_FileClose method as described in table 41, STRS_FileClose().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.71 STRS-71 STRS_FileGetFreeSpace

Requirement	The STRS infrastructure shall contain a callable STRS_FileGetFreeSpace method as described in table 42, STRS_FileGetFreeSpace().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.72 STRS-72 STRS_FileGetSize

Requirement	The STRS infrastructure shall contain a callable STRS_FileGetSize method as described in table 43, STRS_FileGetSize().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.73 STRS-73 STRS_FileGetStreamPointer

Requirement	The STRS infrastructure shall contain a callable STRS_FileGetStreamPointer method as described in table 44, STRS_FileGetStreamPointer().
Rationale	STRS-73 solves the potential problem of I/O methods missing from NASA-STD-4009. Since not all SDRs will have a file system, this method should be used sparingly with comments describing its purpose.
Category	Extensibility
Traced-from	4.4, 4.5
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.74 STRS-74 STRS_FileOpen

Requirement	The STRS infrastructure shall contain a callable STRS_FileOpen method as described in table 45, STRS_FileOpen().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.75 STRS-75 STRS_FileRemove

Requirement	The STRS infrastructure shall contain a callable STRS_FileRemove method as described in table 46, STRS_FileRemove().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	Waveform Remove
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.76 STRS-76 STRS_FileRename

Requirement	The STRS infrastructure shall contain a callable STRS_FileRename method as described in table 47, STRS_FileRename().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.77 STRS-77 Use Messaging API

Requirement	The STRS applications shall use the STRS Infrastructure Messaging, STRS Infrastructure Data Source, and STRS Infrastructure Data Sink methods to establish queues to send messages between components.
Rationale	In an SDR executing multiple threads or processes, messages have to be processed using a queuing method so that they don't interfere with each other. One example might be a receive application queuing its data for a subsequent transmit application. Another example might be the queuing of error messages from STRS_Log.
Category	Portability, Adaptability
Traced-from	4.5, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	The STRS Infrastructure Messaging methods refer to the API defined in STRS-78 through STRS-81 and corresponding tables 48 through 51.
Verification Method	Inspection

7.78 STRS-78 STRS_QueueCreate

Requirement	The STRS infrastructure shall contain a callable STRS_QueueCreate method as described in table 48, STRS_QueueCreate().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.79 STRS-79 STRS_QueueDelete

Requirement	The STRS infrastructure shall contain a callable STRS_QueueDelete method as described in table 49, STRS_QueueDelete().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.80 STRS-80 STRS_Register

Requirement	The STRS infrastructure shall contain a callable STRS_Register method as described in table 50, STRS_Register().
Rationale	The publish-subscribe design pattern provided a way for the publisher of a message to send the message to all subscribers without knowing the details. For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.81 STRS-81 STRS_Unregister

Requirement	The STRS infrastructure shall contain a callable STRS_Unregister method as described in table 51, STRS_Unregister().
Rationale	The publish-subscribe design pattern provides a way for the publisher of a message to send the message to all subscribers without knowing the details. For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.82 STRS-82 Use Time Control API

Requirement	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.
Rationale	For portability of waveform applications, a standard API for using timers in the GPP was necessary. The timers are expected to be used for relatively low accuracy timing such as time stamps, timed events, and time constraints. As the speed of new GPPs increases over time, the timers are expected to be used for signal processing.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	The STRS Infrastructure Time Control methods refer to the API defined in STRS-83 through STRS-88 and corresponding tables 52 through 57 in NASA-STD-4009.
Verification Method	Inspection

7.83 STRS-83 STRS_GetNanoseconds

Requirement	The STRS infrastructure shall contain a callable STRS_GetNanoseconds method as described in table 52, STRS_GetNanoseconds().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.84 STRS-84 STRS_GetSeconds

Requirement	The STRS infrastructure shall contain a callable STRS_GetSeconds method as described in table 53, STRS_GetSeconds().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.85 STRS-85 STRS_GetTime

Requirement	The STRS infrastructure shall contain a callable STRS_GetTime method as described in table 54, STRS_GetTime().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.86 STRS-86 STRS_GetTimeWarp

Requirement	The STRS infrastructure shall contain a callable STRS_GetTimeWarp method as described in table 55, STRS_GetTimeWarp().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.87 STRS-87 STRS_SetTime

Requirement	The STRS infrastructure shall contain a callable STRS_SetTime method as described in table 56, STRS_SetTime().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.88 STRS-88 STRS_Synch

Requirement	The STRS infrastructure shall contain a callable STRS_Synch method as described in table 57, STRS_Synch().
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. In particular, waveform applications and services have to use standard interfaces across all platforms.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Use WFCCN.

7.89 STRS-89 Provide STRS.h

Requirement	The STRS platform provider shall provide an STRS.h file containing the STRS predefined data shown in table 58, STRS Predefined Data.
Rationale	For portability, standard names are defined for various constants and data types.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection using OE script and WFCCN.

7.90 STRS-90 Provide POSIX

Requirement	The STRS OE shall provide the interfaces described in POSIX IEEE Standard 1003.13-2003 profile PSE51.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. Where possible, use currently available standards. Thus POSIX subsets were chosen to implement certain OS services missing from the list of STRS Application-provided methods. The POSIX subsets are widely available, are implemented by multiple OSs, and are scalable. Layering of the architecture separates and encapsulates functionality so that the parts are less influenced by changes to the other. This separation of functionality promotes portability.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection

7.91 STRS-91 Use POSIX

Requirement	STRS applications shall use POSIX methods except for the unsafe functions listed in table 59, Replacements for Unsafe Functions.
Rationale	For an open architecture to support portability, the architecture has to be standardized across platforms and implementations. Where possible, use currently available standards. Thus POSIX subsets were chosen to implement certain OS services missing from the list of STRS Application-provided methods. The POSIX subsets are widely available, are implemented by multiple OSs, and are scalable. Layering of the architecture separates and encapsulates functionality so that the parts are less influenced by changes to the other. This separation of functionality promotes portability.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.7, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	Table 2, STRS Architecture Standard, Table 59, Replacements for Unsafe Functions, is a copy of table 59 of NASA-STD-4009 cited in the requirement.
Verification Method	Inspection using compliance tool.

Table 2—STRS Architecture Standard, Table 59, Replacements for Unsafe Functions

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

7.92 STRS-92 Document HAL

Requirement	The STRS platform provider shall provide 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 interrupt input and output, memory mapping, and the configuration data necessary to operate in the STRS platform environment.
Rationale	The HAL API is to be published so that specialized hardware made by one company may be integrated with the STRS infrastructure made by a different company. The HAL API documentation is to include a description of each method or function used, including its calling sequence, return values, an explanation of its functionality, any preconditions before using the method/function, and the status after using the method or function. The HAL API documentation is to also contain information about the underlying hardware, such as address and data interfaces, interrupt input and output, power connections, plus other control and data lines necessary to operate in the STRS platform environment.
Category	Reconfigurability, Adaptability, Extensibility
Traced-from	4.4, 4.5, 4.7, 4.8, 5.1, 5.2
Use Case	None
Related to	Platform
Notes	See STRS-11.
Verification Method	Inspection of HAL document.

7.93 STRS-93 OE Uses HAL (Deleted)

Requirement	This requirement was deleted because it was the same as STRS-11.
Rationale	STRS-11 stated the same thing, with only a few words different, so STRS-93 is redundant. STRS-93 previously stated: The STRS infrastructure shall use the HAL APIs to communicate with the specialized hardware via the physical interface defined by the STRS platform provider.
Category	Adaptability, Extensibility
Traced-from	4.4, 4.5, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Not applicable (N/A)

7.94 STRS-94 External Commands

Requirement	An STRS platform shall accept, validate, and respond to external commands.
Rationale	To adapt to changing circumstances, an STRS radio has to accept external commands from a ground station, another satellite, or another system on the same satellite. The external commands have to be validated as required by the mission. There has to be a way to determine whether or not the command worked and, for some commands, the resulting values.
Category	Adaptability
Traced-from	4.5, 5.4, 5.15
Use Case	Waveform Upload, STRS OE Upload, Waveform Instantiation, Waveform Start, Processor Resource Sharing with Flight Computer, Set Waveform Parameter, Get Waveform Parameter, Transmit a Packet, Receive a Packet, Waveform Stop, Waveform Deallocation, Waveform Abort, Waveform Remove, Built-In-Test
Related to	OE
Notes	None
Verification Method	Observation of radio operation.

7.95 STRS-95 Use STRS APIs

Requirement	An STRS platform shall execute external application control commands using the standardized STRS APIs.
Rationale	To promote portability and adaptability, the use of the standard STRS APIs is required. One waveform should be able to control another waveform or device in a portable manner.
Category	Portability, Adaptability
Traced-from	4.1, 4.2, 4.3, 4.5, 4.9, 5.1, 5.2, 5.4, 5.6, 5.7
Use Case	Waveform Upload, STRS OE Upload, Waveform Instantiation, Waveform Start, Processor Resource Sharing with Flight Computer, Set Waveform Parameter, Get Waveform Parameter, Transmit a Packet, Receive a Packet, Waveform Stop, Waveform Deallocation, Waveform Abort, Waveform Remove, Built-In Test
Related to	OE
Notes	None
Verification Method	Use WFCCN.

7.96 STRS-96 Use STRS_Query

Requirement	The STRS infrastructure shall use the STRS_Query method to service external system requests for information from an STRS application.
Rationale	The only way to request information from an application is by means of data values returned when an application method is invoked. The STRS_Query/APP_Query methods are designed for this purpose. Although STRS_RunTest/APP_RunTest could be used to request data values, they are designed for testing.
Category	Adaptability, Extensibility
Traced-from	4.4, 4.5, 5.1, 5.2
Use Case	Get Waveform Parameter
Related to	OE
Notes	None
Verification Method	Inspection

7.97 STRS-97 Use STRS_Log (Deleted)

Requirement	This requirement was deleted because it was the same as STRS-57.
Rationale	STRS-97 previously stated: An STRS application shall use the STRS_Log and STRS_Write methods to send STRS telemetry set information to the external system. STRS-97 was less clear than STRS-57 because STRS-97 involved the external interface, not just the application to infrastructure call. Also, in some implementations, the application telemetry may not be sent directly to the external interface but may be sent to a file from which the telemetry may be downloaded as necessary.
Category	Portability
Traced-from	4.9, 5.1, 5.15, 5.16, 5.23
Use Case	None
Related to	Application
Notes	Replaced by STRS-57 after simplifying “STRS_Log and STRS_Write”.
Verification Method	N/A.

7.98 STRS-98 Document Platform for XML

Requirement	The STRS platform provider shall document the necessary platform information (including a sample file) to develop a predeployed application configuration file in XML 1.0.
Rationale	When functions, interfaces, components, and/or design rules are defined and published, the architecture is open. Open architectures facilitate interoperability among commercial and government developers and minimize the operational impact of upgrading hardware and software components. Leveraging the existing XML standard may reduce NASA's costs and risks by increasing reliability.
Category	Reconfigurability, Adaptability, Extensibility
Traced-from	4.2, 4.4, 4.5, 4.7, 5.2, 5.4
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection of document and sample file.

7.99 STRS-99 Document WF for XML

Requirement	The STRS application developer shall document the necessary application information to develop a predeployed application configuration file in XML 1.0.
Rationale	When functions, interfaces, components, and/or design rules are defined and published, the architecture is open. Open architectures facilitate interoperability among commercial and government developers and minimize the operational impact of upgrading hardware and software components. Leveraging the existing XML standard may reduce NASA's costs and risks by increasing reliability.
Category	Reconfigurability, Adaptability, Extensibility
Traced-from	4.2, 4.4, 4.5, 4.7, 5.2, 5.4
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection of delivered documentation.

7.100 STRS-100 Provide XML File

Requirement	The STRS integrator shall provide a predeployed application configuration file in XML 1.0.
Rationale	A waveform (STRS application) configuration file contains specific information that (1) allows STRS to instantiate the application; (2) provides default configuration values; and (3) provides connection references to ports and services needed by the application. The format of the configuration files has to be defined in XML using an XML schema. The XML should be preprocessed to optimize space in the STRS radio memory while keeping the equivalent content. Examples include platform configuration files, STRS infrastructure configuration files as a XML schema, and waveform configuration files that contain specific information that allows STRS to instantiate the application, provide default configuration values, and provide connection references to ports and services needed by the application. Leveraging the existing XML standard may reduce NASA's costs and risks by increasing reliability.
Category	Reconfigurability
Traced-from	4.7, 5.3, 5.4
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection using XMLSpy.

7.101 STRS-101 XML Content

Requirement	The predeployed STRS application configuration file shall identify the following application attributes and default values: (1) Identification. A. Unique STRS handle name for the application. B. Class name (if applicable). (2) State after processing the configuration file. (3) Any resources to be loaded separately. A. Filename of loadable image. B. Target on which to put loadable image file. C. Target memory in bytes, number of gates, or logic elements. (4) Initial or default values for all distinct operationally configurable parameters.
Rationale	A waveform (STRS application) configuration file contains specific information that (1) allows STRS to instantiate the application; (2) provides default configuration values; and (3) provides connection references to ports and services needed by the application. The format of the configuration files has to be defined in XML using an XML schema. The XML should be preprocessed to optimize space in the STRS radio memory while keeping the equivalent content. Examples include platform configuration files, STRS infrastructure configuration files as a XML schema, and waveform configuration files that contain specific information that allows STRS to instantiate the application, provide default configuration values, and provide connection references to ports and services needed by the application.
Category	Reconfigurability
Traced-from	5.4
Use Case	None
Related to	Application
Notes	None
Verification Method	Inspection of delivered files and documentation.

7.102 STRS-102 Provide XML Schema

Requirement	The STRS platform provider shall provide an XML 1.0 schema definition (XSD) file to validate the format and data for predeployed STRS application configuration files, including the order of the tags, the number of occurrences of each tag, and the values or attributes.
Rationale	A waveform (STRS application) configuration file contains specific information that (1) allows STRS to instantiate the application; (2) provides default configuration values; and (3) provides connection references to ports and services needed by the application. The format of the configuration files has to be defined in XML using an XML schema. Since the term XML schema was variously interpreted to mean either a description or a file, the requirement was clarified to specify that an XML schema definition (XSD) file is required. The XML should be preprocessed to optimize space in the STRS radio memory while keeping the equivalent content. Examples include platform configuration files, STRS infrastructure configuration files as a XML schema, and waveform configuration files that contain specific information that allows STRS to instantiate the application, provide default configuration values, and provide connection references to ports and services needed by the application. Leveraging the existing XML standard may reduce NASA's costs and risks by increasing reliability.
Category	Reliability
Traced-from	4.3, 4.7, 5.3, 5.4
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection using XMLSpy.

7.103 STRS-103 Provide XML Transformation Tool

Requirement	The STRS platform provider shall document the transformation (if any) from a predeployed application configuration file in XML into a deployed application configuration file and provide the tools to perform such transformation.
Rationale	A waveform (STRS application) configuration file contains specific information that (1) allows STRS to instantiate the application; (2) provides default configuration values; and (3) provides connection references to ports and services needed by the application. The format of the configuration files has to be defined in XML using an XML schema. The XML should be preprocessed to optimize space in the STRS radio memory while keeping the equivalent content. Examples include platform configuration files, STRS infrastructure configuration files as an XML schema, and waveform configuration files that contain specific information that allows STRS to instantiate the application, provide default configuration values, and provide connection references to ports and services needed by the application.
Category	Reconfigurability, Adaptability, Extensibility
Traced-from	4.4, 4.5, 5.2, 5.3, 5.4
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection of document and tools.

7.104 STRS-104 Provide XML Transformed

Requirement	The STRS integrator shall provide a deployed STRS application configuration file for the STRS infrastructure to place the STRS application in the specified state.
Rationale	A waveform (STRS application) configuration file contains specific information that (1) allows STRS to instantiate the application; (2) provides default configuration values; and (3) provides connection references to ports and services needed by the application. The format of the configuration files has to be defined in XML using an XML schema. The XML should be preprocessed to optimize space in the STRS radio memory while keeping the equivalent content. Examples include platform configuration files, STRS infrastructure configuration files as an XML schema, and waveform configuration files that contain specific information that allows STRS to instantiate the application, provide default configuration values, and provide connection references to ports and services needed by the application.
Category	Reconfigurability
Traced-from	5.4
Use Case	None
Related to	OE
Notes	None
Verification Method	Inspection of delivered files and documentation.

7.105 STRS-105 OE Provides API in C

Requirement	The STRS infrastructure APIs shall have an ISO/IEC C language compatible interface.
Rationale	Because portability is a basic goal but middleware is not required, a totally language-independent solution was not available. The lowest common denominator turns out to be a C language interface. Using a standard ISO/IEC 9899 C or ISO/IEC 14882 C++ aids portability. The year is not included in the requirement, so that obsolete compilers are not mandated.
Category	Portability
Traced-from	4.1, 4.9, 5.1, 5.2
Use Case	None
Related to	OE
Notes	None
Verification Method	Use WFCCN.

7.106 STRS-106 Use STRS.h

Requirement	An STRS application shall use the appropriate constant, typedef, or struct defined in table 58, STRS Predefined Data when the data are used to interact with the STRS APIs.
Rationale	For portability, standard names are defined for various constants and data types.
Category	Portability
Traced-from	4.1, 4.2, 4.3, 4.4, 4.5, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Application
Notes	The table in the requirement is in NASA-STD-4009.
Verification Method	Inspection

7.107 STRS-107 Document External Commands

Requirement	An STRS platform provider shall document the external commands describing their format, function, and any STRS methods invoked.
Rationale	To adapt to changing circumstances, an STRS radio has to accept external commands from a ground station, another satellite, or another system on the same satellite. The external commands have to be validated as required by the mission. There has to be a way to determine whether the command worked and, for some commands, the resulting values. To promote portability and adaptability, the use of the standard STRS APIs is required.
Category	Adaptability
Traced-from	4.5, 5.1, 5.4, 5.5, 5.6, 5.7, 5.15
Use Case	Waveform Upload, STRS OE Upload, Waveform Instantiation, Waveform Start, Processor Resource Sharing with Flight Computer, Set Waveform Parameter, Get Waveform Parameter, Transmit a Packet, Receive a Packet, Waveform Stop, Waveform Deallocation, Waveform Abort, Waveform Remove, Built-In-Test
Related to	OE
Notes	None
Verification Method	Inspection of documentation.

7.108 STRS-108 Document Thermal and Power Limits

Requirement	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.
Rationale	The power consumption and resulting heat generation of a reprogrammable FPGA will vary according to the amount of logic used 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.
Category	Reliability, Adaptability
Traced-from	3.4, 3.5, 4.3, 4.7, 5.9, 5.10, 5.15, 5.16
Use Case	None
Related to	Platform
Notes	See also STRS-9
Verification Method	Inspection of HID document.

7.109 STRS-109 Provide General-Purpose Processing Module

Requirement	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.
Rationale	The GPM contains and executes the STRS OE, including POSIX, STRS interface code, and configuration file parsing, to support the corresponding requirements. A layered hardware architecture augments the layered software architecture by providing the ability to change portions without affecting other portions to support extensibility, adaptability, and portability.
Category	Portability, Adaptability
Traced-from	4.3, 4.4, 4.5, 4.6, 4.8, 4.9, 5.1, 5.2
Use Case	None
Related to	Platform
Notes	None
Verification Method	Inspection