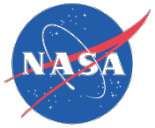
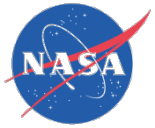


Space Telecommunications Radio System (STRS) Architecture

Tutorial Part 2 - Detailed

Glenn Research Center

February 2014



STRS Tutorials (4009)

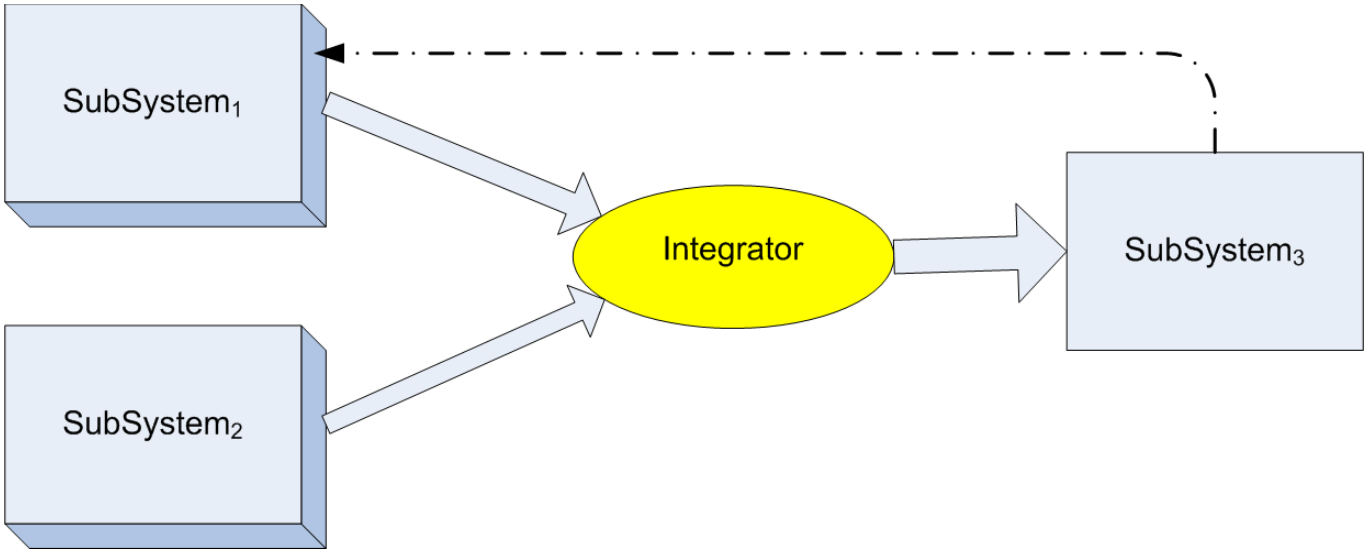
STRS Tutorial – Roles & Responsibilities

Roles Defined for STRS Requirements

- To abstract the responsible organizations.
- To promote vendor independence, scalability, flexibility, and extensibility, while specifying the smallest number of clearly defined roles possible.
- To obtain radios without restricting contracting or subcontracting for hardware and software.
- To allow clear responsibilities to be assigned by the project/mission.
- To allow separate entities to work together.
- To allow one entity to assume multiple roles.

STRS Roles: Providers & Integrators

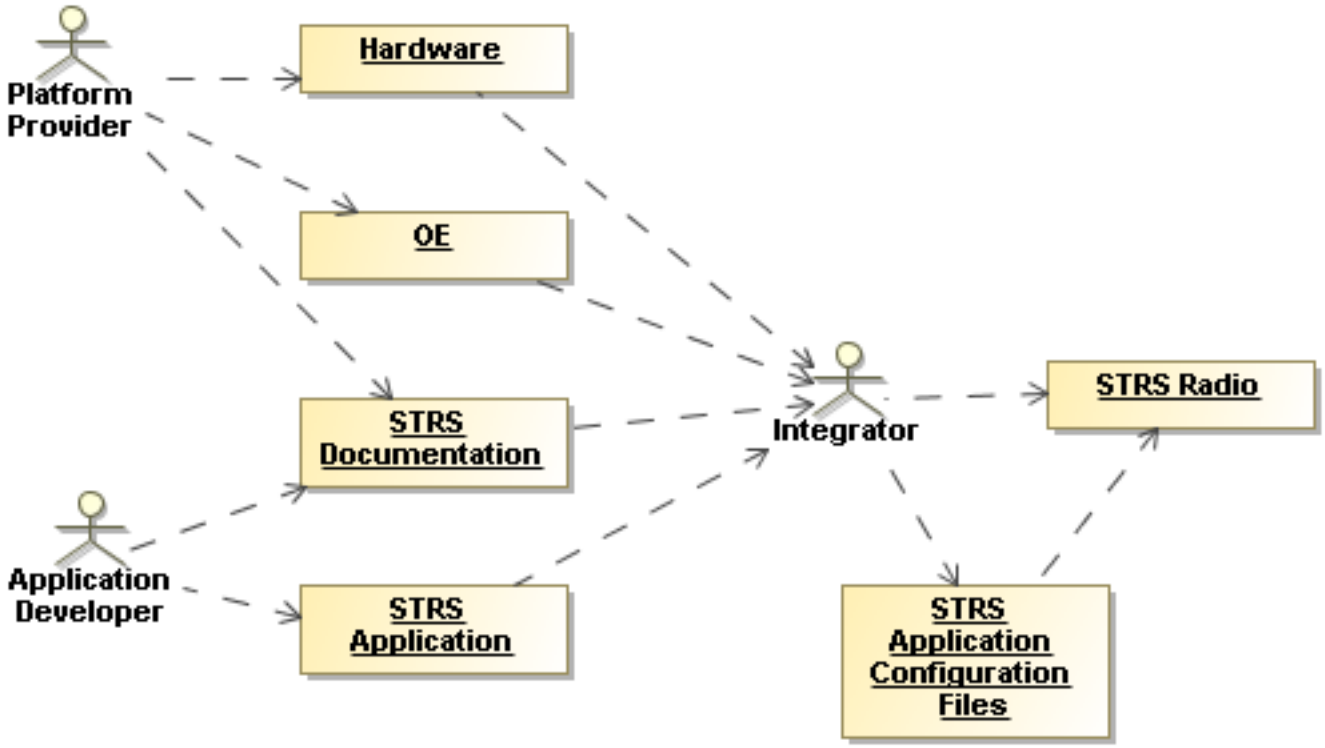
- Each subsystem provider acquires or develops the subsystem to be provided to the corresponding integrator.
- Each integrator combines the subsystems to create a new subsystem.
- The integrator may provide the new subsystem to another integrator.
- The roles and corresponding organizations were often expected to change at different stages of the radio's life cycle.



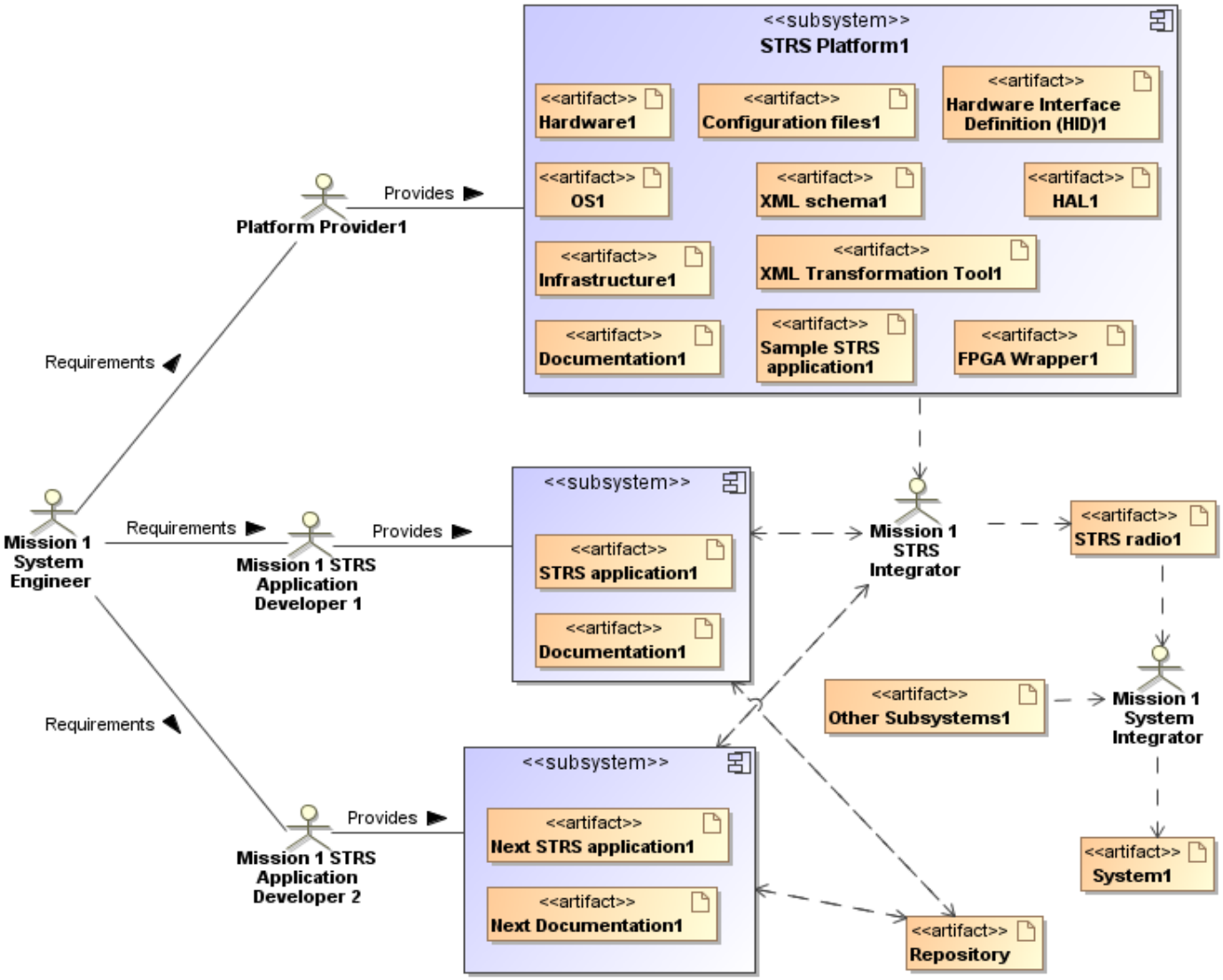
STRS Roles Defined

- **Platform provider** delivers a platform upon which STRS applications could be executed.
  - The 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 platform provider organization.
  - The platform provider would usually act as application developer and integrator for at least a sample application.
- **Application developer** provides the desired functionality in the form of an STRS application.
- **Integrator** gets the parts to work together.

STRS Roles Simplified



STRS Roles Illustrated

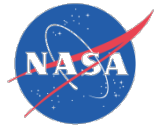


STRS Roles Not Defined

- There are roles missing for:
  - STRS Compliance Tester
  - STRS Repository Manager
  - STRS Configuration Manager

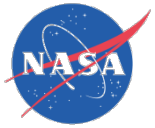
  because these are internal to NASA and not necessary to the creation of the STRS radio
- There are roles missing for:
  - Project Management
  - Change Control Board
  - Reviewers

  because these are required by the project and NPR 7150 but not required by STRS.



STRS Tutorial 2

Operation



Operation

Requirement	Description
STRS-1	An STRS platform shall have a known state after completion of the power-up process.
STRS-2	The STRS Operating Environment shall access each module's diagnostic information via the STRS APIs.
STRS-3	Self-diagnostic and fault-detection data shall be created for each module so that it is accessible to the STRS Operating Environment.
STRS-13	If the STRS application has a component resident outside the GPM (e.g., in configurable hardware design), then the component shall be controllable from the STRS Operating Environment.
STRS-94	An STRS platform shall accept, validate, and respond to external commands.
STRS-95	An STRS platform shall execute external application control commands using the standardized STRS APIs.
STRS-107	An STRS platform provider shall document the external commands describing their format, function, and any STRS methods invoked.
STRS-96	The STRS infrastructure shall use the STRS_Query method to service external system requests for information from an STRS application.

Operation

Rationale:

- Even if the method of commanding the radio is different for each radio, consistency in using the STRS architecture is necessary for an architecture to aid portability.

STRS Tutorial 3

Hardware & Hardware Abstraction Layer

Hardware & Hardware Abstraction Layer

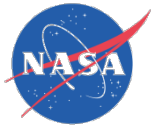
Requirement	Description
STRS-109	An STRS platform shall have a GPM that contains and executes the STRS OE and the control portions of the STRS applications and services software.

Hardware & Hardware Abstraction Layer

Requirement	Description
STRS-14	The STRS SPM developer shall provide a platform-specific wrapper for each user-programmable FPGA, which performs the following functions: 1) Provides an interface for command and data from the GPM to the waveform application 2) Provides the platform-specific pinout for the application developer. This may be a complete abstraction of the actual FPGA pinouts with only waveform application signal names provided.
STRS-11	The STRS infrastructure shall use the STRS Platform HAL APIs to communicate with application components on the platform specialized hardware via the physical interface defined by the platform developer.
STRS-92	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 other configuration data necessary to operate in the STRS platform environment.

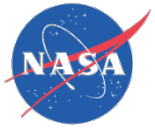
STRS Tutorial 4

Documentation & Repository



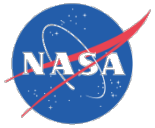
HID Documentation & Wrapper

Requirement	Document
STRS-4	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.
STRS-5	The STRS platform provider shall describe, in the HID document, the reconfigurability behavior and capability of each reconfigurable component.
STRS-6	The STRS platform provider shall describe, in the HID document, the behavior and performance of the RF modular component(s).
STRS-7	The STRS platform provider shall describe, in the HID document, the interfaces that are provided to and from each modular component of the radio platform.
STRS-8	The STRS platform provider shall describe, in the HID document, the control, telemetry, and data mechanisms of each modular component (i.e., how to program or control each modular component of the platform, and how to use or access each device or software component, noting any proprietary and nonstandard aspects).
STRS-9	The STRS platform provider shall describe, in the HID document, the behavior and performance of any power supply or power converter modular component(s).
STRS-108	The 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.



HID Documentation & Wrapper

Requirement	Document
STRS-15	The STRS SPM developer shall provide documentation on the configurable hardware design interfaces of the platform-specific wrapper for each user-programmable FPGA, which describes the following: (1) Signal names and descriptions. (2) Signal polarity, format, and data type. (3) Signal direction. (4) Signal-timing constraints. (5) Clock generation and synchronization methods. (6) Signal-registering methods. (7) Identification of development tool set used. (8) Any included noninterface functionality.



Documentation & Repository

Requirement	Document
STRS-12	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.

Documentation & Repository

Rationale:

- Hardware behavior, capabilities, limitations, and identification need to be captured so that the software developers can use the hardware to advantage.
- Hardware and software needs to be inventoried to know when updates are needed or made.
- Software artifacts need to be captured and inventoried so that the platform and/or software can be reused and/or updated.
- Licensing and intellectual property issues need to be documented to avoid legal disputes.

STRS Tutorial 5

STRS C/C++ Header Files & Predefined Data

STRS C/C++ Header Files

Requirement	Description
STRS-16	The STRS *Application-provided Application Control API* shall be implemented using ISO/IEC C or C++.
STRS-17	The STRS infrastructure shall use the *STRS Application-provided Application Control API* to control STRS applications.
STRS-105	The STRS infrastructure APIs shall have an ISO/IEC C language compatible interface.
STRS-18	The STRS Operating Environment shall support ISO/IEC C or C++, or both, language interfaces for the *STRS Application-provided Application Control API* at compile-time.
STRS-19	The STRS Operating Environment shall support ISO/IEC C or C++, or both, language interfaces for the *STRS Application-provided Application Control API* at run-time.

STRS C/C++ Header Files

Requirement	Description
STRS-20	Each STRS application shall contain: *#include "STRS_ApplicationControl.h"*
STRS-21	The STRS platform provider shall provide an “STRS_ApplicationControl.h” that contains the method prototypes for each STRS application and, for C++, the class definition for the base class STRS_ApplicationControl.
STRS-22	If the *STRS Application-provided Application Control API* is implemented in C++, the STRS application class shall be derived from the *STRS_ApplicationControl* base class.
STRS-23	If the STRS application provides the *APP_Write* method, the STRS application shall contain: *#include "STRS_Sink.h"*
STRS-24	The STRS platform developer shall provide an “STRS_Sink.h” that contains the method prototypes for APP_Write and, for C++, the class definition for the base class STRS_Sink.
STRS-25	If the *STRS Application-provided Application Control API* is implemented in C++ AND the STRS application provides the *APP_Write* method, the STRS application class shall be derived from the *STRS_Sink* base class.
STRS-26	If the STRS application provides the *APP_Read* method, the STRS application shall contain: *#include "STRS_Source.h"*
STRS-27	The STRS platform developer shall provide an “STRS_Source.h” that contains the method prototypes for APP_Read and, for C++, the class definition for the base class STRS_Source.
STRS-28	If the *STRS Application-provided Application Control API* is implemented in C++ AND the STRS application provides the *APP_Read* method, the STRS application class shall be derived from the *STRS_Source* base class.

STRS C/C++ Header Files

Rationale:

- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.

STRS C/C++ Header Files

Examples:

Minimally, a C++ header file for the application or other component should contain a class definition of the form:

class MyWaveform : public STRS_ApplicationControl {…};

A sink is used for a push model of passing data by writing data to the component; i.e., implementing APP_Write. Then, a header file should contain a class definition of the form:

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

A source is used for a pull model of passing data by reading data from the component; i.e. implementing APP_Read. For example, a header file should contain a class definition of the form:

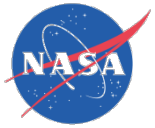
```
class MyWaveform :      public STRS_ApplicationControl,
                        public STRS_Source
{…};
```

If both *APP_Read* and *APP_Write* are provided in the same waveform, the C++ class will be derived from all three base classes named in requirements (STRS-22, STRS-25, and STRS-28). For example, a header file should contain a class definition of the form:

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

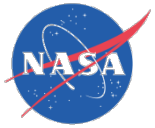
STRS C/C++ Predefined Data

Requirement	Description
STRS-89	The STRS platform provider shall provide an STRS.h file containing the STRS predefined data shown in table 58, STRS Predefined Data.
STRS-106	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.



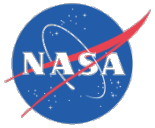
STRS C/C++ Predefined Data

Typedef Name	Description
STRS_Access	a type of number used to indicate how reading and/or writing of a file or queue is done.
STRS_Buffer_Size	a type of number used to represent a buffer size in bytes.
STRS_Clock_Kind	a type of number used to represent a kind of clock or timer.
STRS_File_Size	a type of number used to represent a size in bytes.
STRS_HandleID	a type of number used to represent an STRS application, device, file, or queue.
STRS_int8	an 8-bit signed integer
STRS_int16	a 16-bit signed integer
STRS_int32	a 32-bit signed integer
STRS_int64	a 64-bit signed integer
STRS_ISR_Function	used to define static C-style function pointers passed to the STRS_SetISR() method.
STRS_Message	a char array pointer used for messages.
STRS_NumberOfProperties	a type of number used to represent the number of properties in a Properties structure.
STRS_Queue_Type	a type of number used to represent the queue type..
STRS_Priority	a type of number used to represent the priority of a queue.
STRS_Properties	shorthand for “struct Properties”
STRS_Property	shorthand for “struct Property”
STRS_Result	a type of number used to represent a return value, where negative indicates an error.
STRS_TestID	a type of number used to represent the built-in test or ground test to be performed by APP_RunTest or APP_GroundTest, respectively.
STRS_TimeWarp	a representation of a time delay.
STRS_Type	a type of number used to indicate whether a file is text or binary.
STRS_uint8	an 8-bit unsigned integer
STRS_uint16	a 16-bit unsigned integer
STRS_uint32	a 32-bit unsigned integer
STRS_uint64	a 64-bit unsigned integer



STRS C/C++ Predefined Data

Constant Name	Description
STRS_ACCESS_APPEND	writing is allowed preserving previous data written. Corresponds to fopen mode “a”.
STRS_ACCESS_BOTH	both reading and writing are allowed. Corresponds to fopen mode “r+”.
STRS_ACCESS_READ	reading is allowed. Corresponds to fopen mode “r”.
STRS_ACCESS_WRITE	writing is allowed. Corresponds to fopen mode “w”.
STRS_OK	the STRS_Result is valid.
STRS_ERROR	the STRS_Result is invalid such that the component is still usable.
STRS_ERROR_QUEUE	the STRS_HandleID indicates that the log queue is for error messages.
STRS_FATAL	the STRS_Result is invalid such that the component is not usable.
STRS_FATAL_QUEUE	the STRS_HandleID indicates that the log queue is for fatal messages.
STRS_PRIORITY_HIGH	a number representing a high priority queue.
STRS_PRIORITY_MEDIUM	a number representing a medium priority queue.
STRS_PRIORITY_LOW	a number representing a low priority queue.
STRS_QUEUE_PUBSUB	a number representing a Publish/Subscribe queue type.
STRS_QUEUE_SIMPLE	a number representing a simple queue type.
STRS_TELEMETRY_QUEUE	the STRS_HandleID indicates that the log queue is for telemetry data.
STRS_TEST_STATUS	value of type STRS_TestID used as the argument to APP_RunTest and STRS_RunTest so that the state of the STRS application is returned.
STRS_TEST_USER_BASE	value of type STRS_TestID for the lowest numbered user-defined test. Any STRS_TestID values lower than STRS_TEST_USER_BASE are reserved arguments to APP_RunTest. An example of a test type lower than STRS_TEST_USER_BASE is STRS_TEST_STATUS.
STRS_TYPE_BINARY	the value indicating that a file is a binary file.
STRS_TYPE_TEXT	the value indicating that a file is a text file.
STRS_WARNING	the STRS_Result is invalid such that there may be little or no effect on the operation of the component.
STRS_WARNING_QUEUE	the STRS_HandleID is for warning messages.
STRS_APP_FATAL	state indicating that a nonrecoverable error has occurred.
STRS_APP_ERROR	state indicating that a recoverable error has occurred.
STRS_APP_INSTANTIATED	state indicating that the object is instantiated and ready to accept messages.
STRS_APP_RUNNING	state indicating that STRS_Start() has been called.
STRS_APP_STOPPED	state indicating that STRS_Initialize() or STRS_Stop() has been called.



STRS C/C++ Predefined Data

Struct Name	Description
Property	a struct with two character pointer variables: name and value.
Properties	a struct with two variables (nProps and mProps) of type STRS_NumberOfProperties, and an array of Property structures (vProps). The variable nProps contains the number of items in the vProps array. The variable mProps contains the maximum number of items in the vProps array.

STRS C/C++ Predefined Data

Rationale:

- For portability, standard names are defined for various constants and data types, but the implementation of these definitions is mission dependent.

STRS Tutorial 6

STRS Application-provided Application Control API

STRS Application-provided Application Control API

Rationale

- Need methods to control the actions of a waveform or other application, devices, etc.
- Leverage state-of-the-art standards and experience.
- An open standard architecture and interfaces are used to support portability.
- JTRS/SCA and OMG/SWRADIO define similarly named methods. Allows a similar PIM (platform-independent model), with a different PIM to PSM (platform-specific model) transformation.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Application-provided Application Control API

Methods:

- STRS-29 APP_Configure
- STRS-30 APP_GroundTest
- STRS-31 APP_Initialize
- STRS-32 APP_Instance
- STRS-33 APP_Query
- STRS-34 APP_Read
- STRS-35 APP_ReleaseObject
- STRS-36 APP_RunTest
- STRS-37 APP_Start
- STRS-38 APP_Stop
- STRS-39 APP_Write

STRS Application-provided API

- STRS-29 APP_Configure
  - Prototype:
    - STRS_Result APP_Configure(STRS_Properties * propList)
  - Description:
    - Set values for one or more properties in the application (or device or other entity that is configurable).
- STRS-33 APP_Query
  - Prototype:
    - STRS_Result APP_Query(Properties *propList)
  - Description:
    - Obtain values for one or more properties in the application (or device or other entity that is queryable).

STRS Application-provided API

- STRS-31 APP_Initialize
  - Prototype:
    - STRS_Result APP_Initialize()
  - Description:
    - Initialize the application (or device) to a known initial state based on what has been configured previously.
- STRS-32 APP_Instance
  - Prototype:
    - ThisSTRSApplication *APP_Instance(STRS_HandleID handleID, char *name)
  - Description:
    - Set the handle name and identifier (ID). In C++, it is a static method used to call the class constructor for C++.

STRS Application-provided API

- STRS-35 APP_ReleaseObject
  - Prototype:
    - STRS_Result APP_ReleaseObject()
  - Description:
    - Free any resources the application (or device) has acquired. An example would be to close any open files or devices.
- STRS-36 APP_RunTest
  - Prototype:
    - STRS_Result APP_RunTest(STRS testID, STRS_Properties *propList)
  - Description:
    - Test the application (or device or other entity that is testable).
    - The tests provide aid in isolating faults within the application.

STRS Application-provided API

- STRS-30 APP_GroundTest
  - Prototype:
    - STRS_Result APP_GroundTest(STRS_TestID testID, STRS_Properties *propList)
  - Description:
    - Perform unit and system testing usually done on ground before deployment. The testing may include calibration.

STRS Application-provided API

- STRS-37 APP_Start
  - Prototype:
    - STRS_Result APP_Start()
  - Description:
    - Begin normal application processing.
- STRS-38 APP_Stop
  - Prototype:
    - STRS_Result APP_Stop()
  - Description:
    - End normal application processing.

STRS Application-provided API

- STRS-34 APP_Read
  - Prototype:
    - STRS_Result APP_Read(STRS_Message buffer, STRS_Buffer_Size nb)
  - Description:
    - Method is used to obtain data from the application (or device or other entity) that is a source of data to the infrastructure.
- STRS-39 APP_Write
  - Prototype:
    - STRS_Result APP_Write(STRS_Message buffer, STRS_Buffer_Size nb)
  - Description:
    - Method used to send data to the application (or device or other entity) that is a sink receiving data from the infrastructure.

STRS Tutorial 7

STRS Infrastructure-provided Application Control API

STRS Infrastructure-provided Application Control API

Requirement	Method	Description
STRS-40	STRS_Configure	Set values for one or more properties in the application (or device).
STRS-41	STRS_GroundTest	Perform unit and system testing usually done on the ground before deployment. The testing may include calibration.
STRS-42	STRS_Initialize	Initialize the target component.
STRS-43	STRS_Query	Obtain values for one or more properties in the target component.
STRS-44	STRS_ReleaseObject	Free any resources the application has acquired.
STRS-45	STRS_RunTest	Test the target component.
STRS-46	STRS_Start	Begin normal application processing.
STRS-47	STRS_Stop	End normal application processing.

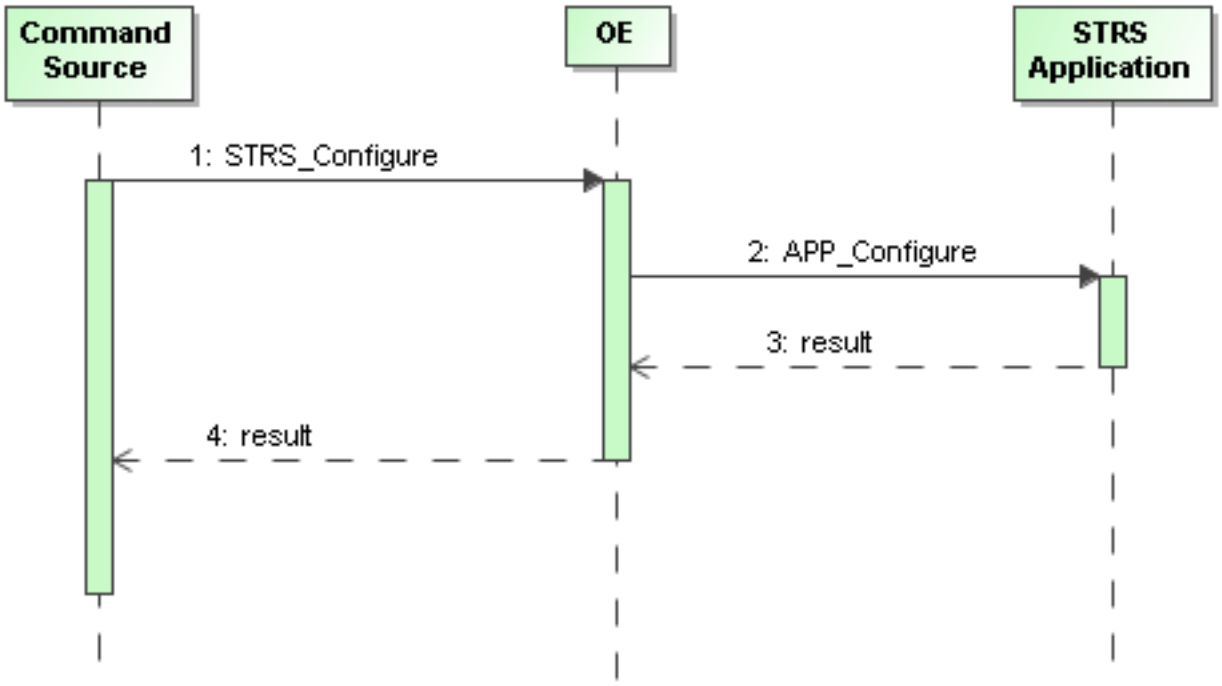
STRS Infrastructure-provided Application Control API

Rationale

- Need methods to control the actions of a waveform or other application, devices, etc.
- Because a C language compatible interface was required, the corresponding methods had a different calling sequence from the STRS Application-provided Application Control API.
- An open standard architecture and interfaces are used to support portability.
- JTRS/SCA and OMG/SWRADIO define similarly named methods. Allows a similar PIM (platform-independent model), with a different PIM to PSM (platform-specific model) transformation.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Infrastructure-provided Application Control API

- Many of these methods are implemented as shown below for STRS_Configure (and corresponding APP_Configure):



STRS Infrastructure-provided Application Control API

- Example of use:

```
struct {
    STRS_NumberOfProperties nProps;
    STRS_NumberOfProperties mProps;
    STRS_Property  vProps[MAX_PROPS];
} propList;
propList.nProps = 2;
propList.mProps = MAX_PROPS;
propList.vProps[0].name  = "A";
propList.vProps[0].value = "5";  /* Set A=5. */
propList.vProps[1].name  = "B";
propList.vProps[1].value = "27"; /* Set B=27. */
STRS_Result rtn = STRS_Configure(fromWF,toWF,
                                 (STRS_Properties *) &propList);
if ( ! STRS_IsOK(rtn)) {
    STRS_Buffer_Size nb = strlen(  "STRS_Configure fails.");
    STRS_Log(fromWF, STRS_ERROR_QUEUE, "STRS_Configure fails.", nb);
}
```

STRS Tutorial 8

STRS Infrastructure Application Setup API

STRS Infrastructure Application Setup API

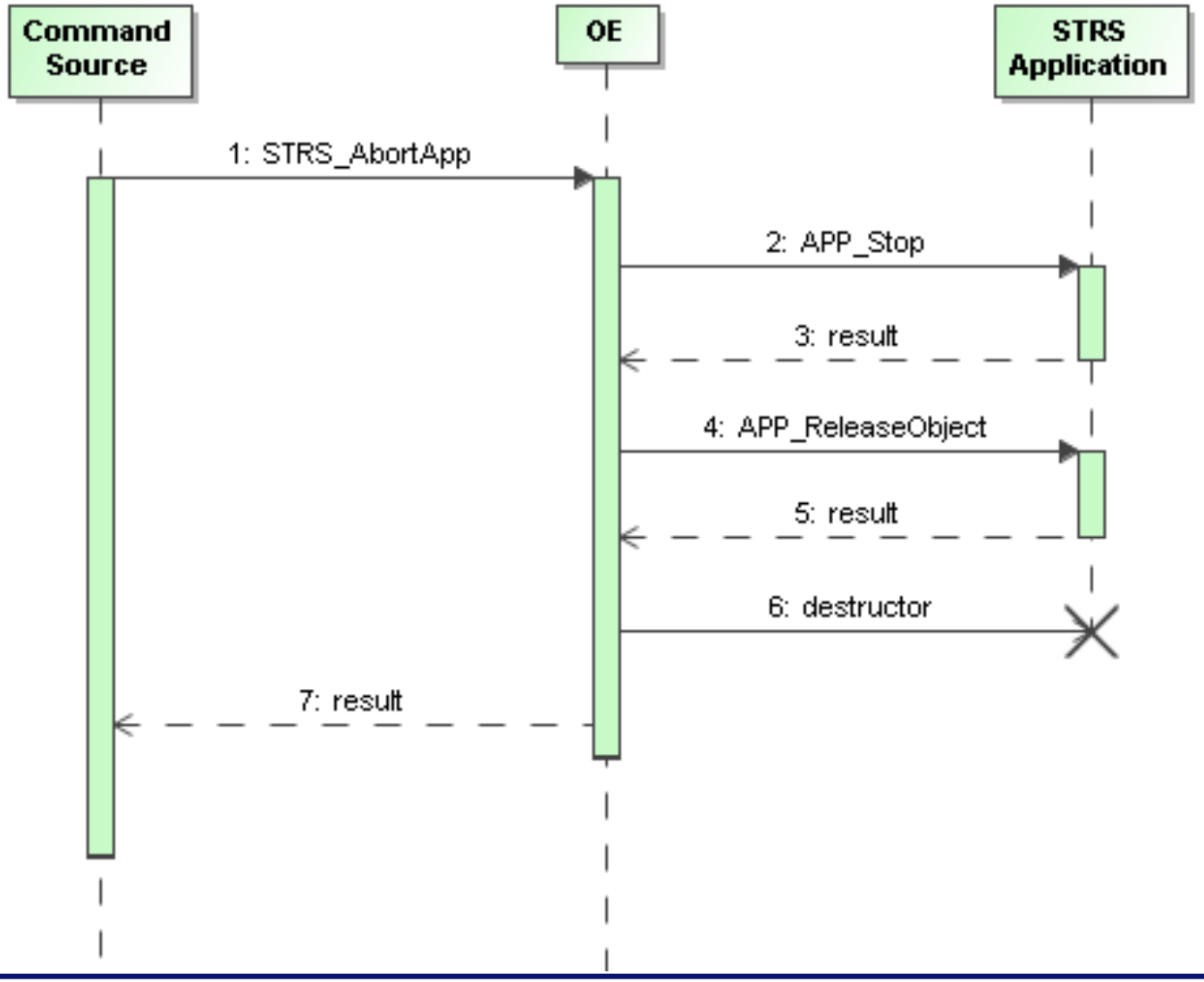
Requirement	Method	Description
STRS-48	STRS_AbortApp	Abort an application or service.
STRS-49	STRS_GetErrorQueue	Transform an error status into an error queue.
STRS-50	STRS_HandleRequest	The table of object names is searched for the given name and an index is returned as a handle ID.
STRS-51	STRS_InstantiateApp	Instantiate an application, service or device and perform any operations by the configuration file.
STRS-52	STRS_IsOK	Return true, if return value of previous call is not an error status.
STRS-53	STRS_Log	Send log message for distribution as appropriate. Time stamp is added automatically.
STRS-54	STRS_Log	When an STRS application has a non-fatal error, the STRS application shall use the *STRS_Log* method with a target handle ID of constant STRS_ERROR_QUEUE.
STRS-55	STRS_Log	When an STRS application has a fatal error, the STRS application shall use the *STRS_Log* method with a target handle ID of constant STRS_FATAL_QUEUE.
STRS-56	STRS_Log	When an STRS application has a warning condition, the STRS application shall use the *STRS_Log* method with a target handle ID of constant STRS_WARNING_QUEUE.
STRS-57	STRS_Log	When an STRS application needs to send telemetry, the STRS application shall use the *STRS_Log* method with a target handle ID of constant STRS_TELEMETRY_QUEUE.

STRS Infrastructure Application Setup API

Rationale:

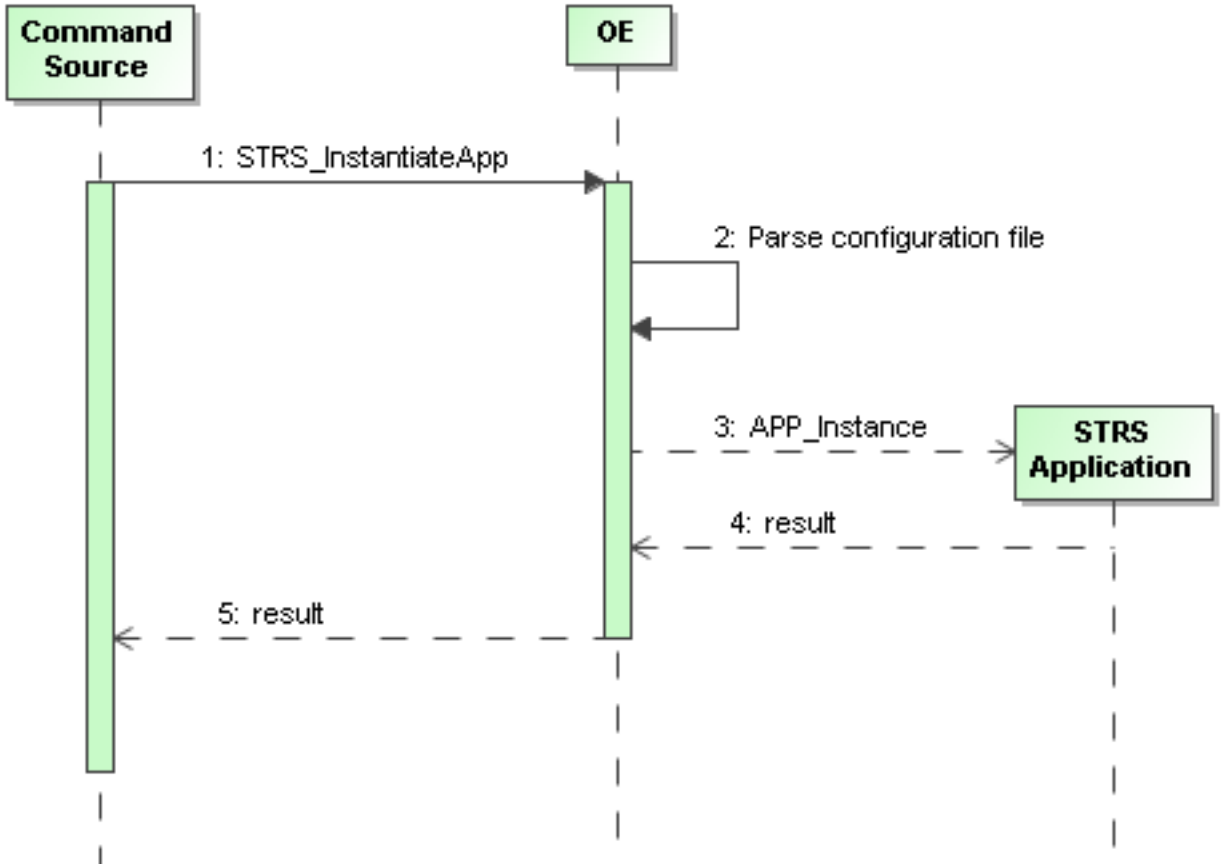
- The methods for these requirements are used to support the STRS Infrastructure-provided Application Control API.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS-48 STRS_AbortApp



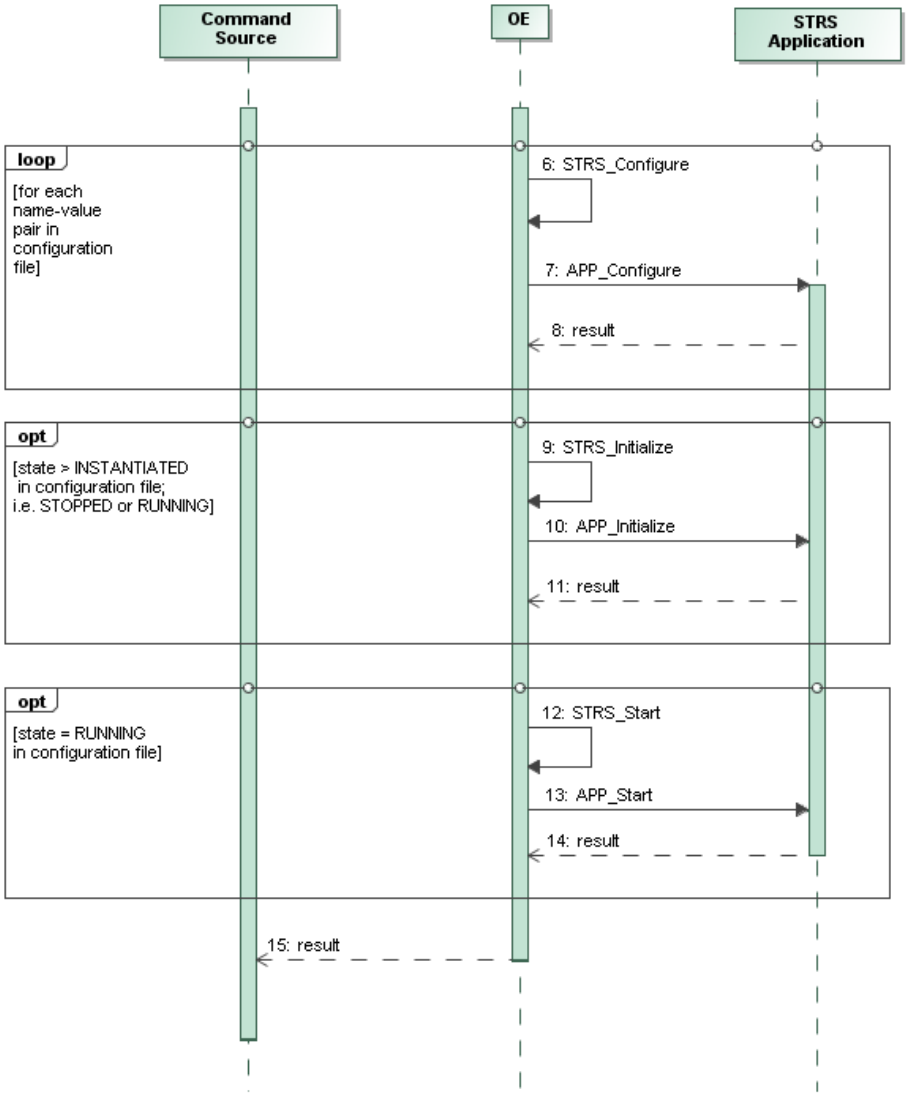
STRS-51 STRS_InstantiateApp

First part, instantiation:



STRS-51 STRS_InstantiateApp

Second part, configuration:



(Configuration)

(State: STOPPED)

(State: RUNNING)

STRS-51 STRS_InstantiateApp

Example:

```
char toWF[MAX_PATH_LENGTH];
strcpy(toWF,"/path/STRS_WFxxx.cfg");
STRS_HandleID wfID = STRS_InstantiateApp(fromWF,toWF);
if (wfID < 0) {
   STRS_Buffer_Size nb = strlen(
       "InstantiateApp fails.");
   STRS_Log(fromWF,
        STRS_GetErrorQueue((STRS_Result)wfID),
        "InstantiateApp fails.", nb);
} else {
   cout << "Successful instantiation for " << toWF
        << ": " << wfID << std::endl;
}
```

STRS Tutorial 9

STRS Infrastructure
Data Sink & Data Source

STRS Infrastructure
Data Sink & Data Source

Requirement	Method	Description
(STRS-58)	STRS_Write	Method used to send data to a sink (application, device, queue, or file). APP_Write is implemented within a sink application when data is to be sent to and used by that application.
(STRS-59)	STRS_Read	Method used to obtain data from a source or supplier (application, device, queue, or file). APP_Read is implemented within a source application when data is to be obtained from that application and used elsewhere.

STRS Infrastructure
Data Sink & Data Source

Rationale

- Sinks and sources are needed to allow a generic interface for I/O.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Infrastructure
Data Sink & Data Source

Example:

```
char buffer[32];
STRS_Buffer_Size nb = 32;
STRS_Result rtn = STRS_Read(fromWF,pullID,buffer,nb);
if (STRS_IsOK(rtn)) {
    cout << "Read " << rtn << " bytes." << std::endl;
    nb = rtn;
    STRS_Result rtn = STRS_Write(fromWF,toID,buffer,nb);
    if (STRS_IsOK(rtn)) {
        cout << "Wrote " << rtn << " bytes." << std::endl;
    } else {
        nb = strlen("Error writing.");
        STRS_Log(fromWF, STRS_GetErrorQueue(wfID), "Error writing.", nb);
    }
} else {
    nb = strlen("Error reading.");
    STRS_Log(fromWF, STRS_GetErrorQueue(wfID), "Error reading.", nb);
}
```

STRS Tutorial 10

STRS Infrastructure Device Control API

STRS Infrastructure Device Control API

Requirements	Method	Description
STRS-60	(many)	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.

STRS Infrastructure Device Control API

Requirements	Method	Description
STRS-61	STRS_DeviceClose	Close the device.
STRS-62	STRS_DeviceFlush	Send any buffered data immediately to the underlying hardware and clear the buffers.
STRS-63	STRS_DeviceLoad	Load a binary image to the device.
STRS-64	STRS_DeviceOpen	Open the device.
STRS-65	STRS_DeviceReset	Reinitialize the device.
STRS-66	STRS_DeviceStart	Start the device. This is recommended to keep the device from only starting when it is loaded.
STRS-67	STRS_DeviceStop	Stop the device. This is recommended to keep the device from being unloaded to just pause since most devices stop when they are unloaded or there is no data to process.
STRS-68	STRS_DeviceUnload	Unload the device.
STRS-69	STRS_SetISR	Set the Interrupt Service Routine for the device.

STRS Infrastructure Device Control API

Rationale:

- STRS Device Control methods are needed to add additional functionality to an STRS application.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Infrastructure Device Control API

- **Example:**

```
char *msg; = NULL
STRS_Result rtn = STRS_DeviceLoad(fromWF,toDev, "/path/WF1.FPGA.bit");
if ( ! STRS_IsOK(rtn)) {  msg = "DeviceLoad fails.";
} else {
 STRS_Result rtn = STRS_DeviceOpen(fromWF,toDev);
 if ( ! STRS_IsOK(rtn)) { msg = "DeviceOpen fails.";
 } else {
  STRS_Result rtn = STRS_DeviceStart(fromWF,toDev);
  if ( ! STRS_IsOK(rtn)) {         msg = "DeviceStart fails.";
  } else { . . . }}}
 . . .
STRS_Result rtn = STRS_DeviceStop(fromWF,toDev);
if ( ! STRS_IsOK(rtn)) {  msg = "DeviceStop fails.";
} else {
 STRS_Result rtn = STRS_DeviceClose(fromWF,toDev);
 if ( ! STRS_IsOK(rtn)) { msg = "DeviceClose fails.";
 } else {
  STRS_Result rtn = STRS_DeviceUnload(fromWF,toDev);
  if ( ! STRS_IsOK(rtn)) {         msg = "DeviceUnload fails.";
  } else { . . . }}}
```

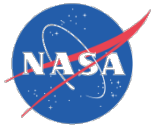
STRS Infrastructure Device Control API

Notes:

- The use of the device functions is not well defined since STRS Devices are not required. This is a potential problem.
- Is the usual order:
  - open/load/start/stop/unload/close
  - load/open/start/stop/close/unload
- STRS_DeviceStart = STRS_Start?
- STRS_DeviceStop = STRS_Stop?

STRS Tutorial 11

STRS Infrastructure File Control API



STRS Infrastructure File Control API

Requirement	Method	Description
STRS-70	STRS_FileClose	Close the file. STRS_FileClose is used to close a file that has been opened by STRS_FileOpen.
STRS-71	STRS_FileGetFreeSpace	Get total size of free space available for file storage.
STRS-72	STRS_FileGetSize	Get the size of the specified file.
STRS-73	STRS_FileGetStreamPointer	Get the file stream pointer for the file associated with the STRS handle ID. This is normally not used because either the common functions are build in to STRS or the entire file manipulation is local to one application or device.
STRS-74	STRS_FileOpen	Open the file. This method is used to obtain an STRS handle ID when the file manipulation is either built in to STRS or distributed over more than one application or device or the STRS infrastructure.
STRS-75	STRS_FileRemove	Remove the file.
STRS-76	STRS_FileRename	Rename the file.

STRS Infrastructure File Control API

Rationale:

- Since a space platform may or may not have a file system, the word “file” was abstracted to mean a named storage area regardless of the existence of a file system.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Infrastructure File Control API

Example:

```
STRS_Buffer_Size nb;
char* msg = NULL;
STRS_File_Size size = STRS_FileGetSize(fromWF,"/path/WF1.FPGA.bit");
if (size <= 0) {
      nb = strlen("FileGetSize fails.");
      STRS_Log(fromWF, STRS_ERROR_QUEUE, "FileGetSize fails.", nb);
}
STRS_HandleID frd = STRS_FileOpen(fromWF,filename,
                                        STRS_ACCESS_READ, STRS_TYPE_TEXT);
if (frd < 0) { msg = "FileOpen fails.";
} else {
 STRS_Result rtn;
 char buffer[32];
 nb = 32;
 rtn = STRS_Read(fromWF,frd, buffer,nb);
 if ( ! STRS_IsOK(rtn)) {                    msg = "Read fails.";
 } else {
  rtn = STRS_FileClose(fromWF,frd);
  if ( ! STRS_IsOK(rtn)) {                   msg = "FileClose fails.";
  } else { . . .
  }}}
```

STRS Tutorial 12

STRS Infrastructure Messaging Control API

STRS Infrastructure Messaging Control API

Requirements	Method	Description
STRS-77	(many)	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..
STRS-78	STRS_QueueCreate	Create a queue (FIFO).
STRS-79	STRS_QueueDelete	Delete a queue.
STRS-80	STRS_Register	Register an association between a publisher and subscriber. Disallow adding an association such that the subscriber has another association back to the publisher because this would cause an infinite loop.
STRS-81	STRS_Unregister	Remove an association between a publisher and subscriber.

STRS Infrastructure Messaging Control API

Rationale:

- The STRS Infrastructure Messaging methods are used to send messages between components with a single queue handle ID.
- The ability for applications, services, devices, or files to communicate with other STRS applications, services, devices, or files is crucial for the separation of radio functionality among independent asynchronous components.
  - For example, the receive and transmit telecommunication functionality can be separated between two applications where the final destination of a message is not necessarily known to the producer of the message..
  - Another example is when commands or log messages come from several independent sources and have to be merged appropriately.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

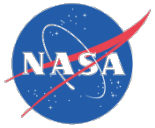
STRS Infrastructure Messaging Control API

Example:

```
STRS_HandleID qX = STRS_QueueCreate(myQ, "QX",
      STRS_QUEUE_SIMPLE, STRS_PRIORITY_MEDIUM);
if (qX < 0) {
   STRS_Buffer_Size nb = strlen("Can't create queue.");
   STRS_Log(fromWF,STRS_ERROR_QUEUE, "Can't create queue.", nb).
   return STRS_ERROR;
}
rtn = STRS_Write(myQ, qX, "This is the message.", strlen("This is the message."));
if (! STRS_IsOK(rtn)) {
   STRS_Buffer_Size nb = strlen("Can't write queue.");
   STRS_Log(fromWF,STRS_ERROR_QUEUE, "Can't write queue.", nb);
}
. . .
STRS_Result rtn = STRS_QueueDelete(myQ,qX);
if (! STRS_IsOK(rtn)) {
   STRS_Buffer_Size nb = strlen("Can't delete queue.");
   STRS_Log(fromWF,STRS_ERROR_QUEUE, "Can't delete queue.", nb);
}
```

STRS Tutorial 13

STRS Infrastructure
Time Control API



STRS Infrastructure Time Control API

Requirement	Method	Description
STRS-82		Any portion of the STRS Applications on the GPP needing time control shall use the STRS Infrastructure Time Control methods to access the hardware and software timers.
STRS-83	STRS_GetNanoseconds	Get the number of nanoseconds from the STRS_TimeWarp object.
STRS-84	STRS_GetSeconds	Get the number of seconds from the STRS_TimeWarp object.
STRS-85	STRS_GetTime	Get the current base time and the corresponding time of a specified type. The base clock/timer is a hardware timer. The interval between two non-base times of different kinds only makes sense if they are in the same frame of reference. To compute the interval between two non-base times in the same frame of reference, the function is called twice and the interval is modified by the difference between the two base times.
STRS-86	STRS_GetTimewarp	Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.
STRS-87	STRS_SetTime	Set the current time in the specified clock/timer by adjusting the time offset.
STRS-88	STRS_Synch	Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external.

STRS Infrastructure Time Control API

Rationale:

- The *STRS Infrastructure Time Control* methods are used to access the hardware and software timers.
- These methods include conversion of time between seconds and nanoseconds and some implementation-specific object.
  - Although nanoseconds are specified, that does not imply that the resolution is nanoseconds, nor that the underlying STRS_TimeWarp object contains its data in nanoseconds.
  - These timers are expected to be used for relatively low accuracy timing such as time stamps, timed events, and time constraints.
  - The timers are expected to be used for signal processing in the GPP when the GPP becomes fast enough.
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

STRS Infrastructure Time Control API

Example:

```
STRS_TimeWarp b1,b2,t1,t2,diff;
STRS_int32 isec,nsec;
STRS_Result rtn;
STRS_Clock_Kind k1 = 1;
STRS_Clock_Kind k2 = 2;
rtn = STRS_GetTime(fromWF,toDev,*b1,k1,*t1);
rtn = STRS_GetTime(fromWF,toDev,*b2,k2,*t2);
/* The time difference between timer k1 and timer k2 is computed by obtaining
 *  the two times, t1 and t2, and adjusting for the time difference between
 *  the two base times, b2 and b1:  */
isec =  STRS_GetSeconds(t2) -
        (STRS_GetSeconds(t1) +
        (STRS_GetSeconds(b2) -
         STRS_GetSeconds(b1)));
nsec = STRS_GetNanoseconds(t2) -
        (STRS_GetNanoseconds(t1) +
        (STRS_GetNanoseconds(b2) -
         STRS_GetNanoseconds(b1)));
diff = STRS_GetTimeWarp(isec,nsec);
```

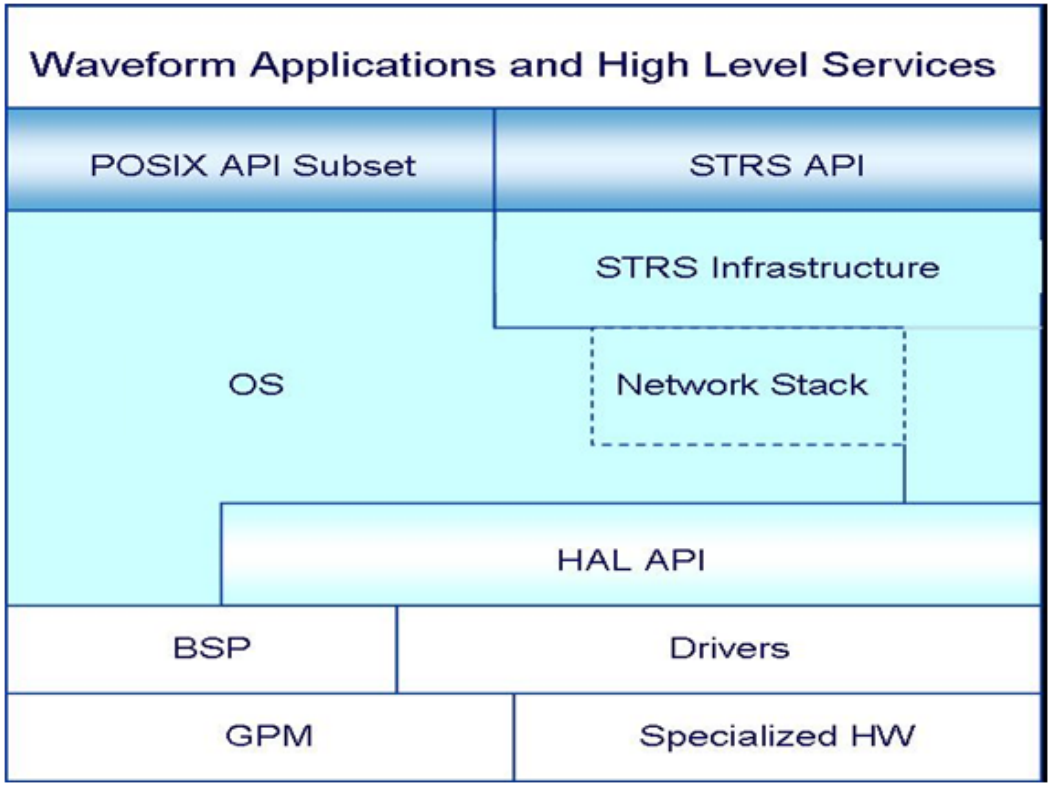
STRS Tutorial 14

POSIX

Portable Operating System Interface

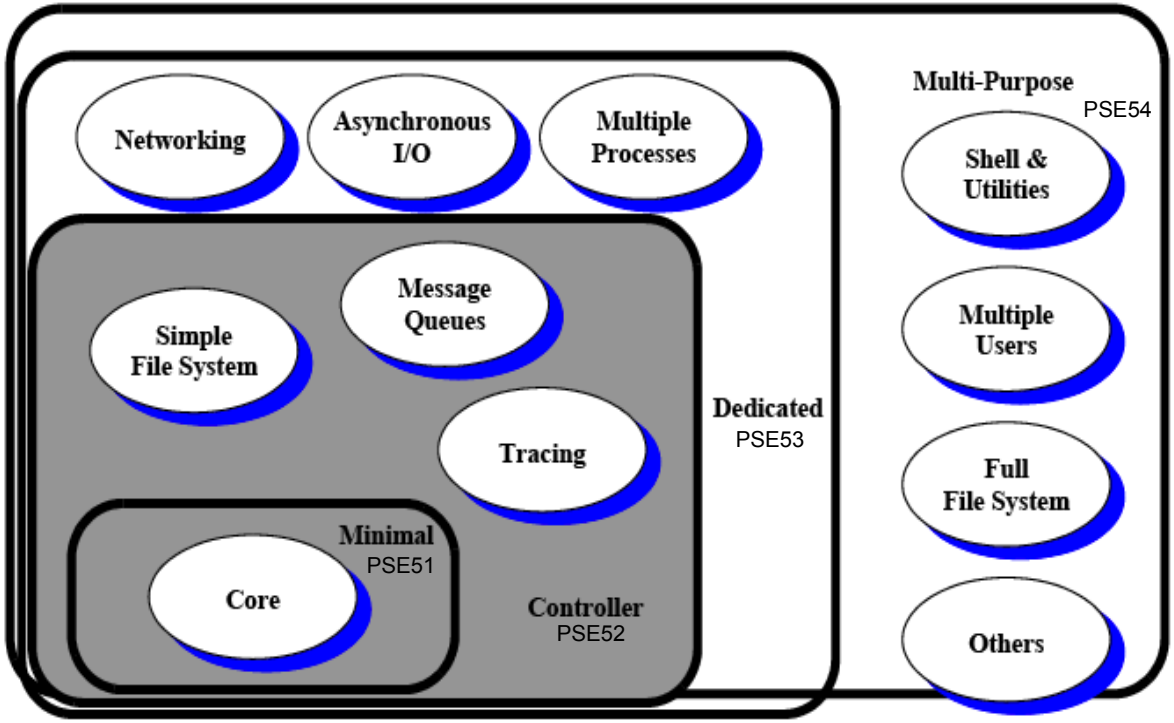
POSIX

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



POSIX

- (STRS-90) The STRS Operating Environment shall provide the interfaces described in POSIX IEEE Standard 1003.13-2003 profile PSE51.



POSIX

- (STRS-91) STRS Applications shall use POSIX methods except for the unsafe functions listed in table 59, Replacements for Unsafe Functions.

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

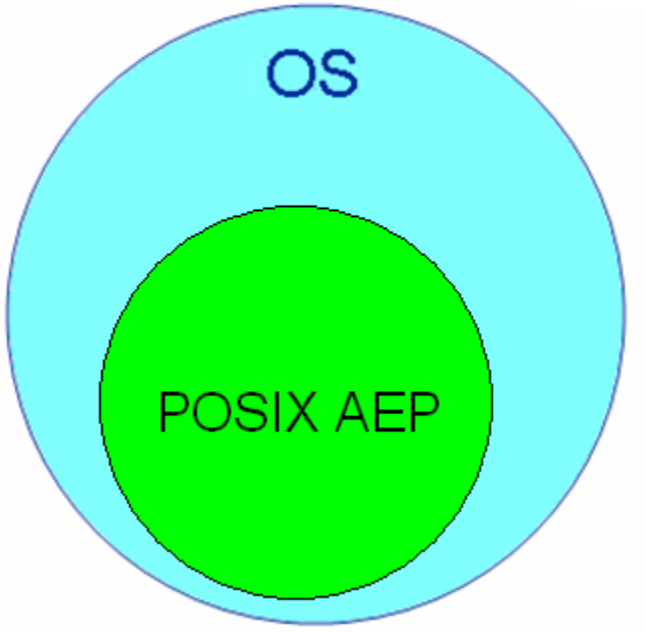
POSIX

Rationale:

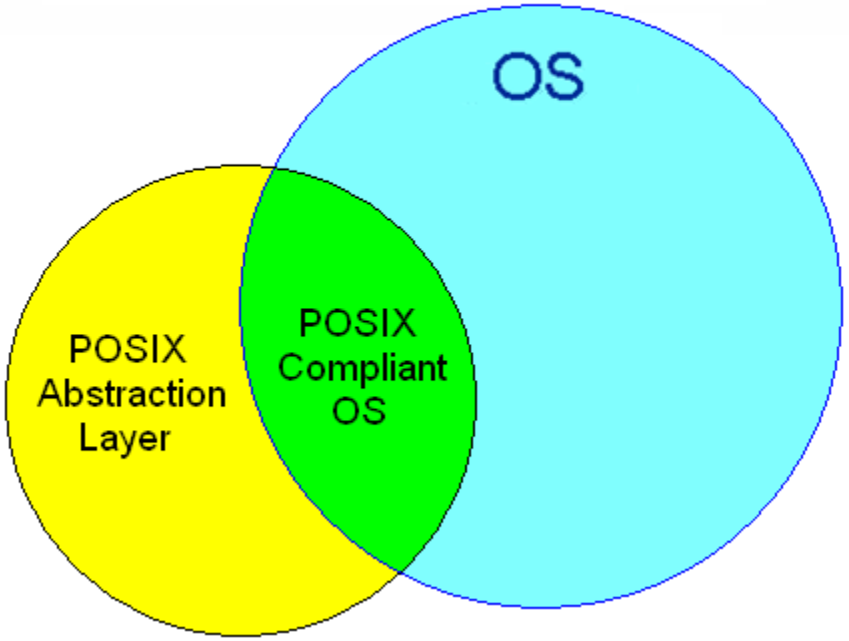
- A POSIX interface was selected because:
  - most operating systems implement POSIX
  - most additional methods needed were available in POSIX
  - POSIX was already an IEEE standard (1003.x)
- An open standard architecture and interfaces are used to support portability.
- Scalable, flexible, reliable, extensible, adaptable, portable.
- Layered architecture used to isolate waveform applications from hardware specific implementations.

POSIX Abstraction Layer

POSIX Conformant OS:



POSIX Compliant OS:



An STRS operating environment can either use an OS that conforms with 1003.13 PSE51 or provide a POSIX abstraction layer that provides missing PSE51 interfaces. For constrained resource platforms, the POSIX requirement is based on waveform requirements so that the waveforms are upward compatible (require POSIX methods).

POSIX Tailoring

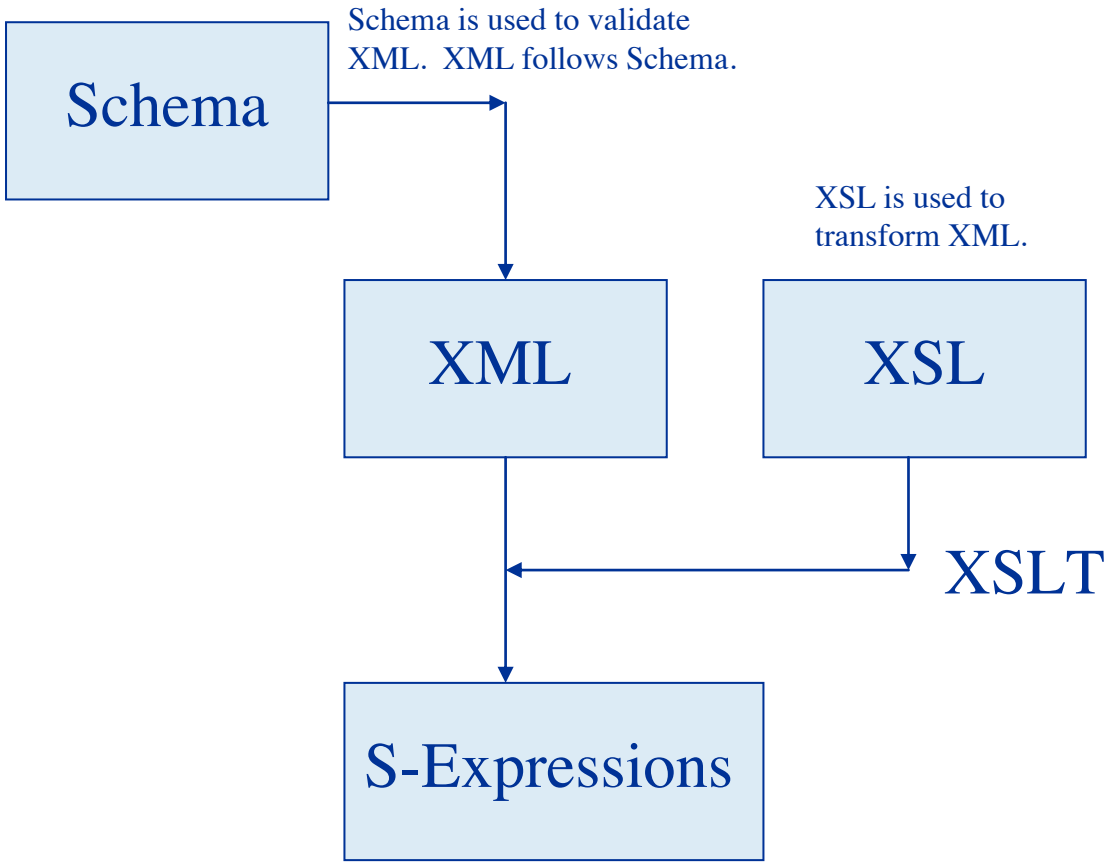
- If a POSIX implementation does not have some required methods, a POSIX abstraction layer should be implemented in the infrastructure for those methods.
- For large platforms, try to stick with PSE51, if possible.
- For constrained resource platforms, with limited software evolutionary capability, where the waveform signal processing is implemented in specialized hardware, the supplier may request a waiver to only implement a subset of POSIX PSE51 as required by the portion of the waveforms residing on the GPP. The applications created for this platform must be upward compatible to a larger platform containing POSIX PSE51. The POSIX API is grouped into units of functionality. If none of the applications for a constrained resource platform use any of the interfaces in a unit of functionality, then the supplier may request a waiver to eliminate that entire unit of functionality.

STRS Tutorial 15

Application Configuration Files

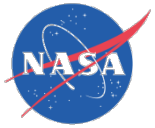
Application Configuration Files

XML + Schema + XSL Relationship



Application Configuration Files

Requirement	Who	Shall
STRS-98	Platform provider	Document the necessary platform information (including a sample file) to develop a predeployed application configuration file in XML 1.0.
STRS-99	Application developer	Document the necessary application information to develop a predeployed application configuration file in XML 1.0.
STRS-100	STRS integrator	Provide a predeployed application configuration file in XML 1.0.
STRS-101	Platform provider & integrator	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.
STRS-102	Platform provider	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.
STRS-103	Platform provider	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.
STRS-104	STRS integrator	Provide a deployed STRS application configuration file for the STRS infrastructure to place the STRS application in the specified state.



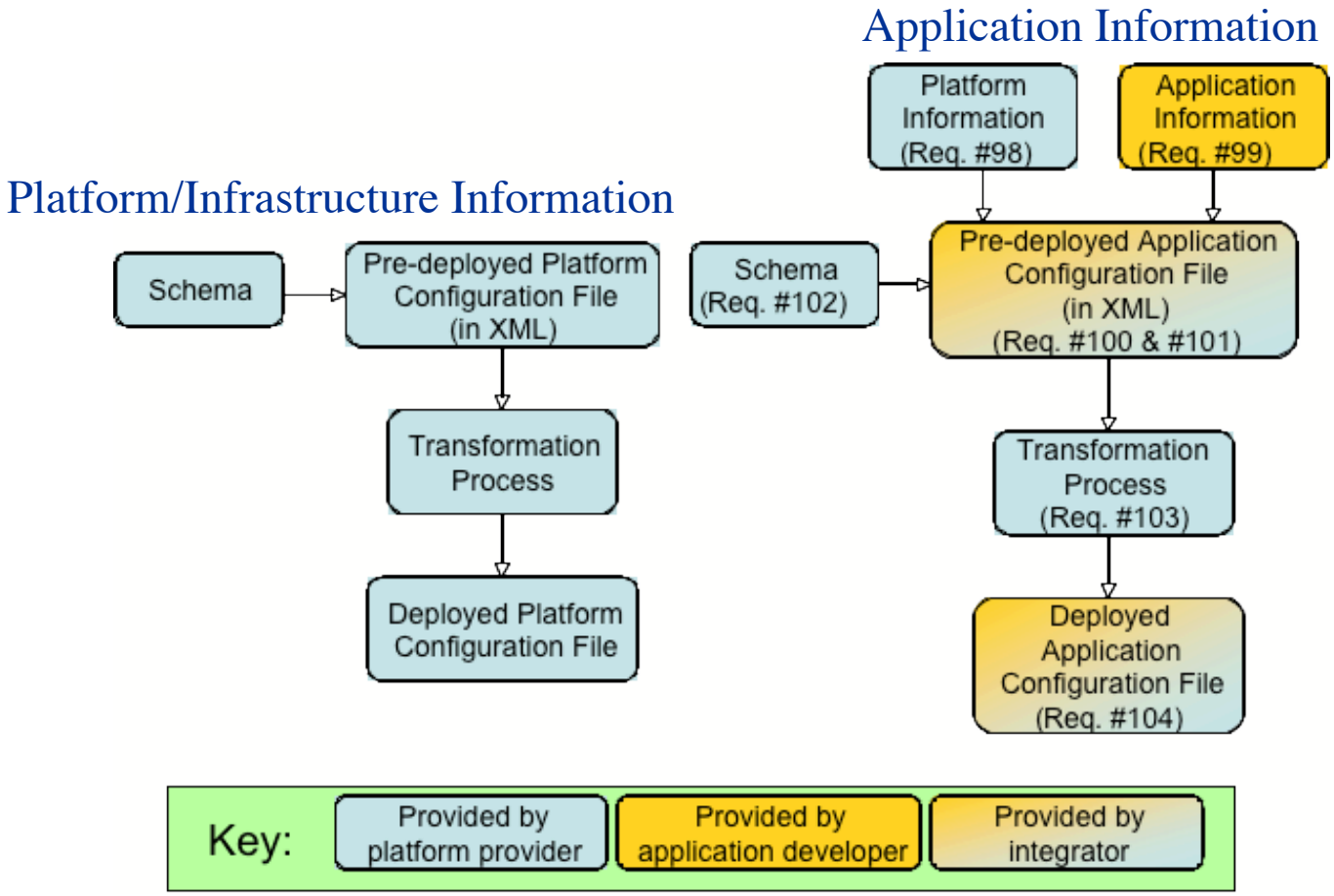
Application Configuration Files

Rationale:

- The use of XML (Extensible Markup Language) version 1.0 allows STRS application developers to have the ability to identify configuration information in a standard (see http://www.w3.org/XML/), human-legible, precise, flexible, and adaptable method.
- The XML configuration file is expected to be pre-parsed, with additional error checking performed prior to transmission.
- Scalable, flexible, reliable, extensible, adaptable, portable.

Application Configuration Files

Configuration File Development Process:




NASA