Certification Test Report

OpenSSRF Software v3.1.0 Release Candidate
Submitted April 13, 2015

This document is the final certification test report for the OpenSSRF software implementation of the SSRF v.3.1.0 data exchange specification. This document details the approach, findings and results of various tests executed against the OpenSSRF software library described in the *Standard Spectrum Resource Format (SSRF) Certification Test Requirements* (Requirements) and *Standard Spectrum Resource Format (SSRF) Certification Test Plan* (Plan) document.

The Standard Spectrum Resource Format (SSRF) defines a set of standardized data elements for automated exchange of radio-frequency (RF) spectrum-related data. The SSRF specification document (Specification) is edited and curated under the authority of DOD Directive 5100.35, and the Military Command, Control, Communications, and Computers Executive Board (MC4EB). The SSRF specification is in the public domain and is royalty-free and license-free.

OpenSSRF is a open source, royalty-free, license-free reference software implementation of the most recent SSRF specification; currently v3.1.0. By "license-free" we mean there is no licensing fee to use the software: the OpenSSRF software code is licensed under the Apache License, Version 2.0.

The OpenSSRF software refernced in this document is available in the download section of http://openssrf.org .

OpenSSRF software is developed and maintained by Key Bridge LLC in collaboration with members of the The Wireless Innovation Forum Spectrum Innovation Committee and with unofficial advisory support from the Defense Spectrum Organization.

100% of the OpenSSRF software passes 100% of the certification tests identified in the Requirements and described in the Test Plan documents. In the author's estimation the OpenSSRF software implementation appears to be 100% SSRF standards compliant: the software passes all certification tests and generates SSRF schema compliant XML documents for all Common data types. Accordingly the authors request that the OpenSSRF v3.1.0 release candidate software is formally certified as a reference implementation.

Contents

Test Result Summary

The entire certification process consisted of exactly 100 tests of which there were zero (0) failures and zero (0) errors. The test results summary is shown in tabular format below with details provided for each data type in their respective sections.

	Maximum Fill		Minimum Fill	
Common Type	**Positive**	**Negative**	**Positive**	**Negative**
Administrative	Pass	Pass	Pass	Pass
Allotment	Pass	Pass	Pass	Pass
Antenna	Pass	Pass	Pass	Pass
AsgnFreqBase	Pass	Pass	Pass	Pass
Assignment	Pass	Pass	Pass	Pass
ChannelPlan	Pass	Pass	Pass	Pass
Contact	Pass	Pass	Pass	Pass
DetailedFunction	Pass	Pass	Pass	Pass
ExternalReference	Pass	Pass	Pass	Pass
FEDeployment	Pass	Pass	Pass	Pass
ForceElement	Pass	Pass	Pass	Pass
IntfReport	Pass	Pass	Pass	Pass
JRFL	Pass	Pass	Pass	Pass
Loadset	Pass	Pass	Pass	Pass
Location	Pass	Pass	Pass	Pass
Message	Pass	Pass	Pass	Pass
Note	Pass	Pass	Pass	Pass
Organisation	Pass	Pass	Pass	Pass
RadiationPlan	Pass	Pass	Pass	Pass
Receiver	Pass	Pass	Pass	Pass
RFSystem	Pass	Pass	Pass	Pass
Role	Pass	Pass	Pass	Pass
Satellite	Pass	Pass	Pass	Pass
SchemaRoot	Pass	Pass	Pass	Pass
SSReply	Pass	Pass	Pass	Pass
SSRequest	Pass	Pass	Pass	Pass
TOA	Pass	Pass	Pass	Pass
Transmitter	Pass	Pass	Pass	Pass

Test Strategy

SSRF is a flexible data formatting and exchange specification. To promote system and service interoperability and compatibility all software implementations of the SSRF specification must, at minimum, produce (write) and accept (read) valid SSRF XML documents. There are two primary categories of SSRF XML document validation:

- XML Document Format to validate that the SSRF document is well formed XML and passes all standard XML schema validation checks using the SSRF XML Schema Definition (XSD) document.
- XML Document Semantics to validate that, within a SSRF Document, all Datasets are correctly configured according to the SSRF XML Schema Definition (XSD) and specification document.

The XML document format and semantics are verified using two techinques:

1. XML document format is verified by inspecting the data

Data validation is an internal function of the OpenSSRF sofware. The OpenSSRF software includes a comprehensive set of validation procedures that prevent the generation of XML documents that contain invalid data configurations. Only a valid SSRF software object tree configuration may be successfully written to an XML document.

Prior to writing a SSRF software object tree to an XML document each SSRF software object field configuration is validated according to its respective field parameter constraints. For example, a field may have a numerical range contraint (maximum or minimum value, significant digits, etc.) a string contraint (maximum or minimum length, expected pattern, etc.) or other type-specific constraint.

2. XML document semantics are verified by inspecting the document

OpenSSRF software does not include a schema validation algorithm and cannot detect or prevent non-conforming XML documents or other subtle schema violation errors.

An automated schema validation routine was therefore developed to evaluate the SSRF XML documents, which always contain valid field data, against the SSRF schema.

Schema errors typically result from configuration logic constraints that are difficult to enforce in software. Common schema validation identifies errors include, for example, invalid collection lengths, missing or null entries from a choice selection, etc.

Test Procedure

SSRF, when viewed as a data structure, represent a simple, standard message container into which a collections of 25 different data types may be stored. In this context a SSRF software objec t is simply an enumerated collection of collections, where each collection is a set of one of 25 enumerated software object types.

All 25 different data types are derived from a common data structure, appropriately called **Common**. Hereafter the term “Common” is used to reference a non-specific data type.

Because the top level (SSRF) software object is merely a simple container of collections of Common instantiations, certification testing therefore focused on each of the 25 Common types, which themselves represent the useful information in a SSRF document.

Each of the Common types was tested independently. That is: a SSRF software object tree and XML document under test only contained one Common type of software object. Different Common types were not co-mingled or tested simulataneously.

A software program called **SSRFTestUtility** was developed to automate the generation of various SSRF configurations. The SSRFTestUtility produces a SSRF software object tree by reading each field’s respective schema constraints and generating a compliant field value as follows:

- Numerical fields are populated with random values that vary through the entire allowed numerical range.
- String fields are populated with a maximum-length sequence containing the character “R” for each required character and the hash character “#” for each option character.
- Enumerated fields, including string fields where an enumerated value is desired or recommended, are populated with a randomly selected value from the corresponding enumerated list.
- Collections are populated with a random number of instatiated types, the number ranging from one to 10. For example, a field for a collection Remarks may contain between one to 10 Remarks entries.

Four independet tests were developed and executed against each Common type. The four tests are designed to evaluate the full range of OpenSSRF software capability and Common type configuration:

A “Positive” test indicates that all field are intentionally configured with known good (i.e. valid) data, wheras a “Negative” test indicates that one or more fields have been intentionally configured with known bad (e.g. invalid) data.

A Positive test should therefore always generate an XML document while a Negative test should never pass basic validation nor generate an XML document.

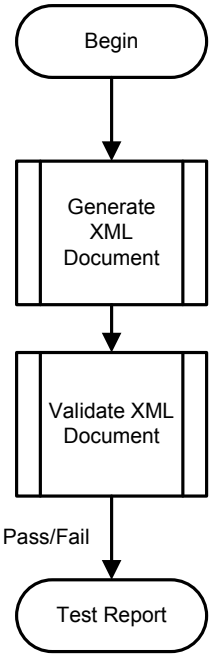
To accomplish the test strategy and procedures described above an automated software program called **SSRFTestUtility** was developed to perform the various tests described in the Plan document.

Each Common type was tested as follows:

1. A Common type was selected
2. SSRFTestUtility was called to populate the Common type with either a minimum or maximum configuation. This produced a Common type configuration with an expected good configuration.
3. The single populated Common type was inserted into a SSRF container. This SSRF container then served as the input software object tree for all four test types (max/min positive/negative fill).
    a. If a negative fill test was desired the SSRFTestUtility was called again to corrupt a random selection of configured data fields in the SSRF container were automatically selected and corrupted. The selected field(s) were recorded for later comparison.
4. The SSRF container was attempted written to XML. For positive tests this attempt must succeed and for negative attempts this attempt must fail.
    a. For positive tests the resultant XML files was validated against the SSRF schema.
    b. For negative tests the SSRF container was evaluated for errors with the discovered set of errors compared against the selected fields recorded earlier. The two sets (discovered and expected) should matched exactly.

Positive Fill Test

A “Positive” fill test is where fields are intentionally configured with known good (i.e. valid) data. In a positive fill test a software object tree is configured and then written to an XML document. The XML document is then validated against the specification schema. This concept is illustrated to the right.



In a minimum positive fill test only fields and data items identified as required by the schema are populated whereas in a maximum positive fill test all possible fields and data items are populated.

A minimum fill test is useful to inspect and evaluate basic functionality and expected bahavior. The maximum positive fill test is essentially a stress test to evaluate the extent, range and capability of a Common type configuration.

The minimum fill test produces the absolute minimum valid configuration of a Common data type and the smallest viable SSRF XML document. The maximum fill test produces a very large configuration and XML document. For example, the Transmitter maximum configuration XML document can exceed 18 MBytes and over 41,000 lines.

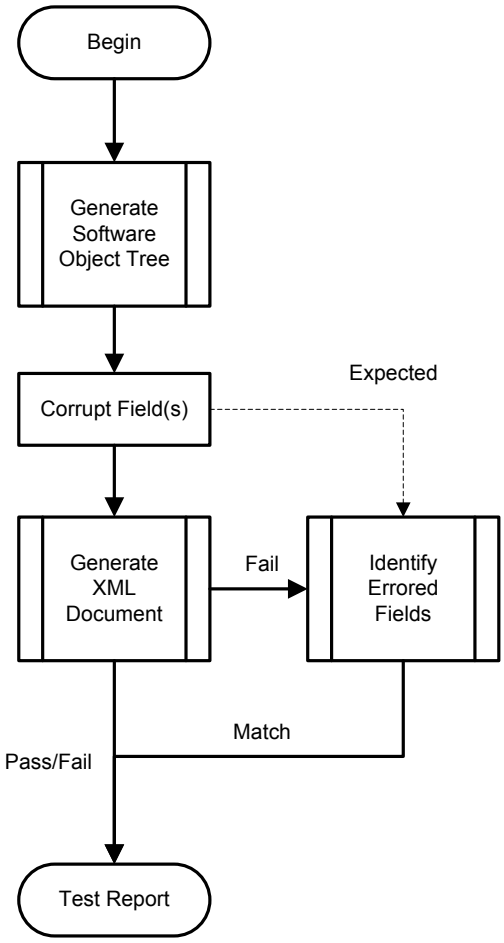
In a minimum fill test all required collection are filled with a single entry. A maximum fill test populates all collections with a randomly selected number of entries.

A successful minimum or maximum fill test requires two criteria:

1. The SSRF software object tree must validate and write to an XML document
2. The XML document must validate against the SSRF schema

Negative Fill Test

A “Negative” test indicates that one or more fields have been intentionally configured with known bad (e.g. invalid) data. In a negative fill test a software object tree is created then purposefully corrupted. The corrupted software object tree is then written to an XML document, a process which is expected to fail. The software object tree is then inspected for errors and the set of identfified errors is matched against the expected set of known corrupted fields. This concept is illustrated to the right.



A minimum negative fill test is logically an extension of the minimum positive fill test where a known good minimum configuration is established and then one or more field values are purposefully corrupted. A maximum negative fill test, similar to the minimum negative fill test, is logically an extension of the maximum positive fill test where a known good maximum configuration is established and then one or more field values are purposefully corrupted.

The purpose of a negative fill test is to confirm that the OpenSSRF software fails to write an XML document containing data known to be invalid. A second optional goal is to confirm that the OpenSSRF software detects and correctly indentifies all of the corrupted or invalidated fields.

A successful negative fill test requires just one criteria:

1. The SSRF software object tree must not validate nor write to an XML document

The validity of a negative fill test is improved if the SSRF validation procedure correctly identifies all invalid fields. The negative fill test was therefore expanded to evaluate the OpenSSRF validation capability: a random selection of configured fields was selected, recorded and purposefully corrupted. The selection was then compared with the validation results.

A successful field validatation test adds a second optional criteria to the negative fill test:

2. The SSRF validation algorithm should correctly identify all invalid fields

Test Result Detail

Four tests (max/min positive/negative fill) were conducted against each Common type. The entire certification process consisted of exactly 100 tests of which there were zero (0) failures and zero (0) errors.

The resultant XML documents generated from each positive fill test is included with this report in the "results" subdirectory.

All 25 different SSRF data types derive from a common data structure called Common. Common fields were included in each Common type instance test. Individual the test results are detailed below for each Common type.

Common Test Notes

Within the Common definition and therefore within all Common types the SecurityClass .Downgrade collection is limited to maximum 3 entries. In the maximum positive and negative fill tests this field was not configured with a random number of entries but rather was configured with the maximum allowable set size.

Unconstrained fields were configured as follows:

- String fields were populated with an eight (8) character string of hash "#" symbols
- Non-specific Serial references were populated with either a Location or Contact serial number, randomly selected
- Decimal fields were populated with a randomly generated double-precision number ranging between 1.0 and 1,024,000
- Integer fields were populated with a randomly generated double-precision number ranging between 1 and 999,999
- Date and DateTime fields were populated with the current date and date time, respectively

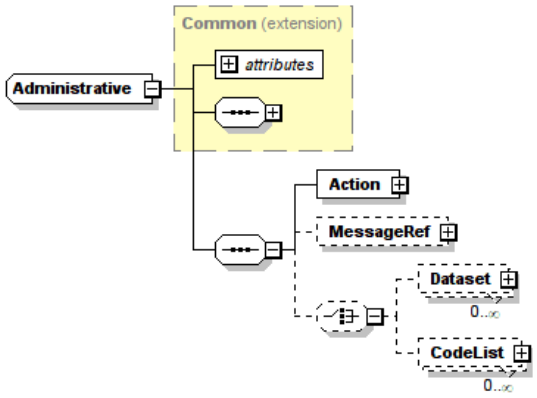
Other Common type-specific test notes are included in each respective section below.

Administrative

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

As illustrated in the schema diagram: in the Administrative data type the Dataset and Codelist collections are a mutually exclusive choice. In all tests only the Dataset type was implemented.



Allotment

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Emission.EmsClass element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

Antenna

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Assignment

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The MINSEC fields (Seconds andMinutes) require a pattern formatted string describing one or more numeratical values in the range [0, 59]. This string field was always configured with the value “30” and not a randomly generated numerical value or string sequence.

The Hours field requires a pattern formatted string describing one or more numeratical values in the range [0, 53]. This string field was always configured with the value “8” and not a randomly generated numerical value or string sequence.

The DaysOfWeek field requires a pattern formatted string describing one or more numeratical values in the range [0, 7] where 0 and 7 are for Sunday, 1 for Monday, etc. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The DaysOfMonth field requires a pattern formatted string describing one or more numeratical values in the range [1, 31]. This string field was always configured with the value “15” and not a randomly generated numerical value or string sequence.

The Months field requires a pattern formatted string describing one or more numeratical values in the range [1, 12]. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The Years field requires a pattern formatted string describing one or more numeratical values in the range [1900, 2100]. This string field was always configured with the value “2007” and not a randomly generated numerical value or string sequence.

The Configuration.ConfigID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The Configuration.ConfigEmission element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

The Assigned.NavAidChannel field requires a string patten describing the allocated pair of radio frequencies. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “123X” was always assigned to this field.

The Assigned.NetNum field requires a detailed string patten describing assigned link network parameter details. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A12325” was always assigned to this field.

The Station.TSDF field requires a detailed string patten describing time slot duty factor assigned to stations of a time division multiple access (TDMA) system. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “40/20” was always assigned to this field.

The Link.LinkID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was assigned to populate this field.

The Station.StationID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The Link.Tuning.RequestedFreq field accepts any instance of the abstract AsgnFreqBase type. This field was always configured using the Freq and never the FreqOld implementation.

Test Discussion

The Maximum negative fill test failed to identify all modified fields.

- Assignment.DaysOfWeek
- Assignment.Minutes
- Assignment.Years
- Assignment.Hours

This is due to the corrupted field values being overwritten as described in the test notes. All other corrupted fields were detected and the negative fill test is therefore deemed successfully.

ChannelPlan

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Contact

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

ExternalReference

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

FEDeployment

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The MINSEC fields (Seconds andMinutes) require a pattern formatted string describing one or more numeratical values in the range [0, 59]. This string field was always configured with the value “30” and not a randomly generated numerical value or string sequence.

The Hours field requires a pattern formatted string describing one or more numeratical values in the range [0, 53]. This string field was always configured with the value “8” and not a randomly generated numerical value or string sequence.

The DaysOfWeek field requires a pattern formatted string describing one or more numeratical values in the range [0, 7] where 0 and 7 are for Sunday, 1 for Monday, etc. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The DaysOfMonth field requires a pattern formatted string describing one or more numeratical values in the range [1, 31]. This string field was always configured with the value “15” and not a randomly generated numerical value or string sequence.

The Months field requires a pattern formatted string describing one or more numeratical values in the range [1, 12]. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The Years field requires a pattern formatted string describing one or more numeratical values in the range [1900, 2100]. This string field was always configured with the value “2007” and not a randomly generated numerical value or string sequence.

Test Discussion

The Maximum negative fill test failed to identify all modified fields.

- FEDeployment.DaysOfMonth

This is due to the corrupted field values being overwritten as described in the test notes. All other corrupted fields were detected and the negative fill test is therefore deemed successfully.

ForceElement

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

IntfReport

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The SourceEmsClass field requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

JRFL

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

Note: The following customizations apply both to JRFL and JRFLEntry.

The MINSEC fields (Seconds andMinutes) require a pattern formatted string describing one or more numeratical values in the range [0, 59]. This string field was always configured with the value “30” and not a randomly generated numerical value or string sequence.

The Hours field requires a pattern formatted string describing one or more numeratical values in the range [0, 53]. This string field was always configured with the value “8” and not a randomly generated numerical value or string sequence.

The DaysOfWeek field requires a pattern formatted string describing one or more numeratical values in the range [0, 7] where 0 and 7 are for Sunday, 1 for Monday, etc. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The DaysOfMonth field requires a pattern formatted string describing one or more numeratical values in the range [1, 31]. This string field was always configured with the value “15” and not a randomly generated numerical value or string sequence.

The Months field requires a pattern formatted string describing one or more numeratical values in the range [1, 12]. This string field was always configured with the value “1” and not a randomly generated numerical value or string sequence.

The Years field requires a pattern formatted string describing one or more numeratical values in the range [1900, 2100]. This string field was always configured with the value “2007” and not a randomly generated numerical value or string sequence.

Test Discussion

The Maximum negative fill test failed to identify all modified fields.

- JRFLEntry.Minutes

- JRFLEntry.Years
- JRFLEntry.Months
- JRFLEntry.DaysOfMonth
- JRFLEntry.Hours
- JRFL.Months
- JRFLEntry.DaysOfWeek
- JRFLEntry.Seconds

This is due to the corrupted field values being overwritten as described in the test notes. All other corrupted fields were detected and the negative fill test is therefore deemed successfully.

Loadset

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The JammingChannelProfile DURATION fields (ObserveTime, TxDuration, RxDuration) require a pattern-formatted String representation of time in hours, minutes and nanoseconds (hhh.mm.sssssssss). DURATION fields were configured with the constant string “123.12.123456789” instead of a randomly generated numerical value.

Location

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Polygon.PolygonPoint collection is constrained to require a minimum of 3 entries. In the maximum positive and negative fill tests this field was not configured with a random number of entries but rather was configured with a fixed set size of ten (10) entries.

Message

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Message Common type contains a collection of DatasetRef elements, which themselves are a simple Serial reference to other Common type elements. When testing the Message type one (for minimum fill) or one or more (for maximum fill) randomly selected Common type element (excluding Message) was created and added to the SSRF software object tree. The Serial field of each non-Message Common type element was then recorded in a DatasetRef element and added to the Message configuration.

Note

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Organisation

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

RadiationPlan

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Receiver

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The RxMode.ModeID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The RxMode.EmsClass element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

RFSystem

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Configuration.ConfigID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The Configuration.ConfigEmission element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

Role

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Satellite

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

SSReply

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Configuration.ConfigID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The Configuration.ConfigEmission element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

SSRequest

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The Configuration.ConfigID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The Configuration.ConfigEmission element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

The SSRequest.State collection is limited to maximum 4 entries. In the maximum positive and negative fill tests this field was not configured with a random number of entries but rather was configured with the maximum allowable set size.

TOA

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

None.

Transmitter

Test	Result
Minimum Positive Fill Test	Pass
Maximum Positive Fill Test	Pass
Minimum Negative Fill Test	Pass
Maximum Negative Fill Test	Pass

Test Notes

The TxMode.ModeID field MUST be unique within the dataset. Instead of the standard generated fill string of “R” and “#” characters a randomly generated UUID sequence was used to populate this field.

The TxMode.EmsClass element requires a detailed string patten describing an emission classification. Instead of the standard generated fill string of “R” and “#” characters the pattern-compliant value “A1AAF” was always assigned to this field.

Conclusion

It is a relatively straightforward process to compile the SSRF schema into a collection of class files. However the result is not necessarily convenient or even usable. The OpenSSRF software library begins with a basic SSRF schema compilation then adds significant, custom developed automation and cross-referencing resources to enable and to simplify SSRF document configuration, assembly and validation, plus XML read and write capabilities.

The OpenSSRF Java software library release candidate that passes 100% of the certification tests identifed in the Requirements required over 1,500 custom modifications and improvements to the software. The final release candidate spans over 750 different software objects and includes over 180,000 lines of code.

The OpenSSRF implementation includes automated utilities to read and write XML documents which may be transmitted via any standard document exchange method. OpenSSRF is specifically developed and intended to be used in a production environment and to support a communications framework and data exchange protocol built using the SSRF data format.

100% of the OpenSSRF software passes 100% of the certification tests. The optional negative fill test error identification also appears to pass 100% when discounting forced configurations as described in the various Common type test notes.

Having a SSRF standards compliant reference implementation is an important component to establishing and ensuring compatible, interoperable SSRF-based systems. The OpenSSRF implementation appears to be 100% SSRF standards compliant: the software generates SSRF schema compliant XML documents for all Common data types.

Accordingly, the authors request that the OpenSSRF v3.1.0 release candidate is formally certified as a reference implementation.

__END