Lean Services Architecture Specification

Date:	Thursday, 11 June 2015
Author(s):	Nick Peach
Version:	2
Status:	Release

VERSION HISTORY

Issue	Date	By	Comment
1	04/02/2014	C. Preston	MOD OGL Release
2	11/06/2015	R. Culverwell	Release of updated specification

CONTENTS

1 Introduction

1.1 Specification Objective

This specification defines the Lean Services schemas, messages, architecture and required functionality of a complaint Lean Services Architecture implementation. Appendix A specifies the syntax for Barrier Validation Rules to define content checks to be performed on Lean Services messages by assurance devices. The additional standards that document any extended functionality (and associated Lean Services Definitions and Records) used by a specific implementation must reference and continue to adhere to this standard.

1.2 Structure of the Specification

Section 1 provides the background to the specification.

Section 2 provides an overview of the Lean Services Architecture.

Section 3 specifies the Lean Services message schema syntax.

Section 4 specifies the core Lean Services calls and events.

Section 5 specifies the Lean Services binary message format.

Section 6 specifies the transport protocols to carry Lean Services messages.

Appendix A specifies the Barrier Validation Rules syntax used to content check Lean Services messages.

1.3 Lean Services Architecture Intent

The intent of the Lean Services Architecture (LSA) is to support the following:

- Comprehensive digital interoperability of systems both within and between any tactical military location and platform, including the soldier platform;
- Coalition tactical interoperability;
- Interoperability between the strategic/operational and tactical environments.

1.4 Lean Services Architecture Characteristics and Principles

The LSA is a schema-based request/response and event message protocol and supporting architecture that provides a Services Orientated Architecture (SOA) in the operational and tactical military domain, or other similar environments.

Typical environmental characteristics for a domain using LSA are:

- No central servers available or that can be assumed to be accessible;
- Variable quality communications; low bandwidth, high latency, frequently interrupted;
- Large numbers of frequently changing participating systems and platforms.

The principles behind Lean Services are:

Strong Architecture	Open, published and sufficiently specified to ensure compatible LSA implementations; Adheres to the principles of SOA, with loose coupling between components; Managed forward and backward service compatibility.
Simple	Simple to understand in concept and simple to implement; Familiar concepts that do not require new skills or techniques by programmers.
Software solution	Software only and provides low implementation overhead; Suitable for lean (low powered) computing devices and lean (low bandwidth) communications; Operating system, programming language and transport protocol neutral.
Flexible	Assume on-going core improvements, for example security and safety, through schema definition and functionality updates; New systems and associated services can be quickly and easily added.

1.5 Referenced Standards

Standards referenced in this document are listed in Table 1.

Name	Location
Apache Avro version 1.7.5	http://avro.apache.org/docs/1.7.5/spec.html
Javascript Object Notation (JSON)	Internet Engineering Taskforce RFC 4627
Hypertext Transport Protocol (HTTP) 1.1	http://www.w3.org/Protocols/rfc2616/rfc2616.html

Table 1. Referenced Standards

2 Lean Services Architecture Overview

2.1 Architecture at a Platform Node

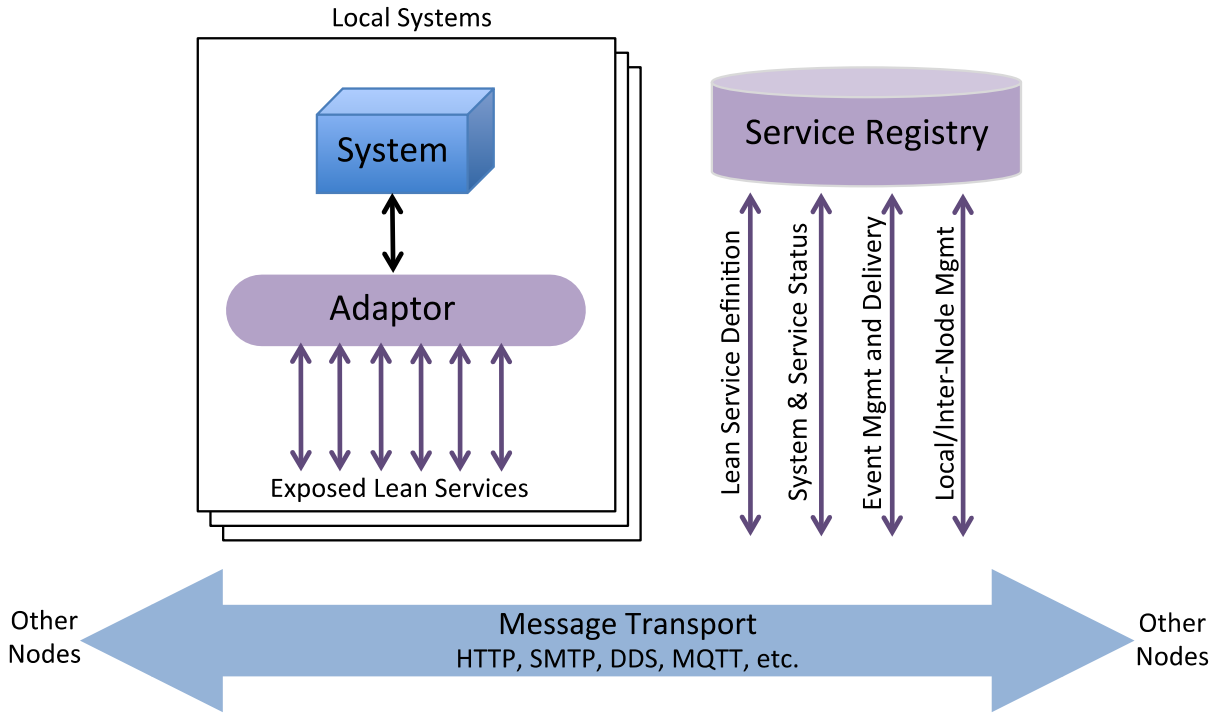

Figure 1. The Lean Services Architecture Overview at a Node

Figure 1 shows a single Lean Services architectural node located on a platform, such as a base, vehicle or soldier. Each node has a Service Registry and the platform can have multiple local systems. Any system can offer an interface using Lean Services, either directly or, as shown in Figure 1, by using a software adaptor. All communications within the Lean Services architecture and between nodes use Lean Services calls and events.

2.2 Service Registry and Event Handler

The Service Registry queries each system for offered services (call and events defined using Lean Services Definitions) and persists this information. As each service status changes, the Service Registry is informed, which persists the updates. Both local and remote Lean Services systems and Service Registries can query a Service Registry for offered services and the status of the services. The Service Registry appropriately distributes all events generated by systems to other systems whose interest matches the registered criteria using an Event Handler. The Event Handler can be integrated with the Service Registry or be a discrete component.

2.3 Lean Services Definition

A Lean Services Definition is a schema that defines the data exchange messages for a service, whether a call or event service. For a Lean Services call, the definition describes a 2-way interaction, consisting of the call request message sent to the service and the response or error message returned. For a Lean Services event, the definition describes the data produced by the service. Version management of Lean Services Definitions is included in this specification to manage both forward and backward compatibility of services through version changes.

The implementation of a Lean Services message uses a binary message format at the service interface.

2.4 Exploiting a Lean Service

To allow the discovery and exploitation of services, the Service Registry persists both the Lean Services Definition for each service and the Uniform Resource Identifier (URI) for the system providing that service. The issuing of a Lean Services message to the URI causes the service to operate.

2.5 Transport Agnostic

A Lean Services message is carried using the transport protocol defined in the service URI. To provide a minimum guarantee of interoperability between implementations, the only mandated transport protocol for Lean Services implementations is HTTP. To add additional transport protocols, either the Service Registry and system/adaptor implementations are all functionally extended to natively support additional protocols or a Gateway is incorporated to convert between protocols.

2.6 Binary format

The proven and open Apache Avro (http://avro.apache.org) MUST be used as the data serialisation technique for Lean Services to enable messages to be sent in binary format (to significantly reduce footprint). Avro is dynamically typed so does not mandate specially generated code to serialise and de-serialise each type of message. The messages can be created and read in a scripted environment, allowing unfamiliar messages to be read with general-purpose software (as long as the schema is known). This accelerates the implementation of interoperability. Avro produces compact binary files appropriate for transmission over tactical radio systems and in lean environments.

3 Lean Services Schema Specification

There are 2 types of Lean Services schema used to define a service, LS Definition and LS Record.

A LS Definition defines the data exchange supported by each lean service. The LS Definition can define the data exchange as a Call, which is request/response (two-way), or an Event (one-way).

A LS Record defines a re-usable data structure that can be used by an LS Definition and other LS Records. To support consistency and normalisation of data handling, a data dictionary of common data elements can be converted into reusable LS Records for use by a specific implementation.

This section defines the:

- LS Definition schema syntax
- LS Record schema syntax
- Rules for creating a Lean Services schema

3.1 Lean Services Schema Syntax

A Lean Services schema is described using JavaScript Object Notation (JSON), RFC 4627[1]. A Lean Services schema is a JSON-object that can contain JSON-objects, JSON-names and JSON-arrays.

A Lean Services attribute uses a JSON-name to describe a name/value pair.

3.1.1 Lean Services Schema Header Attributes

The Lean Services schema header attributes are the first attributes in a Lean Services schema.

```
"type"      : "lsschematype",
"version"   : "lsschemaversion"
"namespace" : "namespace",
"name"      : "name"
```

lsschematype is either "lsdefinition" or "lsrecord" for service definitions.

The ***lsschemaversion*** in this document is "1.0".

namespace is a string stating the namespace of the service.

name is a string stating the name and version of the service.

3.1.2 Lean Services Definition Attributes

The Lean Services Definition attributes follow the Lean Services header schema attributes.

```
"lsservicetype"   : "servicetype",
"parameters"      : [dataparameters],
"response"        : [dataparameters],
"error"           : [dataparameters]
```

servicetype is either "CALL" for a Lean Services call or "EVENT" for a Lean Services event.

The `parameters`, `response` and `error` attributes contain one or more **dataparameters**, or are null.

[1] http://tools.ietf.org/html/rfc4627

An LS Definition schema for a CALL:

```
{
"type"            : "lsdefinition",
"version"         : "1.0"
"namespace"       : "namespace",
"name"            : "name",
"lsservicetype"   : "CALL",
"parameters"      : [dataparameters],
"response"        : [dataparameters],
"error"           : [dataparameters]
}
```

An LS Definition schema for an EVENT:

```
{
"type"            : "lsdefinition",
"version"         : "1.0"
"namespace"       : "namespace",
"name"            : "name",
"lsservicetype"   : "EVENT",
"parameters"      : [dataparameters]
}
```

For a **servicetype** of "EVENT", if response and error attributes are present, they are to be ignored by the implementation.

3.1.3 Lean Services Record Attributes

The Lean Services Record attributes follow the Lean Services header attributes.

```
"fields"          : [dataparameters]
```

The `fields` attribute contains one or more **dataparameters**.

A LS Record schema:

```
{
"type"            : "lsrecord",
"version"         : "1.0"
"namespace"       : "namespace",
"name"            : "name",
"fields"          : [dataparameters]
}
```

3.1.4 Lean Services Data Parameter

A Lean Services data parameter is a JSON-object containing one or more attributes. The first attribute MUST be a Lean Services parameter name/parameter type attribute. LS Data Parameters syntax:

```
{"parametername": "parametertype", ... further attributes ... }
```

3.2 Lean Services Data Parameter Types

3.2.1 Primitive Data Parameters

Function: To define a parameter type as a primitive.

Syntax example:

{"age": "int"}

Primitive types are:

- null: no value
- boolean: a binary value
- int: 32-bit signed integer
- long: 64-bit signed integer
- float: single precision (32-bit) IEEE 754 floating point number
- double: double precision (64-bit) IEEE 754 floating point number
- bytes: sequence of unsigned 8-bit bytes
- string: Unicode character sequence

3.2.2 Lean Services Record Data Parameter

Function: To define a parameter type as a LS Record.

Syntax example:

```
{"homeAddress": "ls.acmecorp.commontypes.addressrecord_v1_0"}
```

3.2.3 Enum Data Parameter

Function: To define a parameter type as enumerated.

The parameter type is "enum".

Further attributes:

- symbols: JSON-array of JSON-strings, mandatory. Duplicate names are prohibited.

Syntax example:

```
{"Seasons": "enum", "symbols": ["WINTER", "SPRING", "SUMMER", "AUTUMN"]}
```

3.2.4 Fixed Data Parameter

Function: To define a parameter type as a bytes field of fixed size.

The parameter type is "fixed".

Further attributes:

- size: integer specifying the number of bytes, mandatory.

Syntax example:

```
{"employeeid": "fixed", "size": 8}
```

3.2.5 Lean Services List Data Parameter

Function: To define a parameter type as an array of a primitive, LS Record or fixed data type.

The syntax of "list" is used followed by the parameter type enclosed in "<>" bracketing.

```
{"parametername": "list<parametertype>"}
```

A list may not immediately contain other lists.

Syntax examples:

```
{"daysoftheweek": "list<string>"}
{"listOfPackageDeliveries": "list<ls.acmecorp.commontypes.addressrecord_v1_0>"}
```

3.3 Rules for Lean Services Schemas

3.3.1 Overview

This section describes the rules to be followed for Lean Services schemas. The proper use of name and namespaces is required to ensure the intended function of Lean Services by correctly identifying and processing each Lean Services schema. The proper management of versioning is required to ensure predictable backward and forward version compatibility.

3.3.2 Namespace and Name Rules

The following rules are mandated:

- A namespace is a series of names separated by a dot (.). The only permitted characters used in a Lean Services schema namespace and schema name are a to z, 0 to 9. The underscore (_) is used for versioning only.
- All namespace and name characters are treated as case insensitive in Lean Services and must be converted to and processed as lower case by all implementation software.
- The first name in a Lean Services schema namespace must be ‘ls’.
- The namespace hierarchy starting ‘ls.messages’ is reserved for use in this specification only.
- ‘ls.messages.core’ is used for well-known calls.
- It is the responsibility of the schema owner to ensure that the namespace-name combination is unique.

The following is advisory:

- It is suggested that a schema owner uses the name of the organisation they represent as the second name in a namespace, e.g. ‘ls.moduk’ or ‘ls.companyname’. Further names can be added to refine the namespace as required by the organisation.
- The names of the Lean Services schemas listed in this document are reserved for use in this specification and should not be reused elsewhere.

3.3.3 Versioning Rules For Schemas, Calls and Events

3.3.3.1 Lean Services Schema Structure

The second attribute in a Lean Services schema is ‘version’. This describes the version of the structure of the LS Definition and LS Record schema. This is defined as "1.0" by this document.

3.3.3.2 Lean Services Definition Schemas

The version of a defined ls call and ls event is appended at the end of the service name.

The version format is Underscore (_) + ‘v’ + major release number + Underscore (_) + minor release number, with no spaces. If no version is present, the implementation treats as ‘_v1_0’.

A minor LS Definition release increment MUST maintain forward and backward compatibility with all implementations using the same major release number.

If distribution of up to date schemas cannot be guaranteed within an implementation, then it is only permitted to add attributes after existing attributes in the ‘parameters’, ‘response’ and ‘error’ sections of an LS Definition for a minor release increment. This ensures that older schemas can be used to

read the message correctly because the extra attributes bytes only appear after the legacy message when the message data is read.

If distribution of up to date schemas to **all participants can be guaranteed**, then more flexibility can be offered using the minor increment versioning rules for a specific implementation.

A major release increment of an LS Definition can be made without consideration for compatibility with previous version releases.

3.3.3.3 Lean Services Record Schemas

The version of a defined LS Record is appended at the end of the name.

The version format is Underscore (_) + 'v' + major release number + Underscore (_) + minor release number, with no spaces. If no version is present, the implementation treats as '_v1_0'.

If up to date schema distributions cannot be guaranteed within an implementation, it is only permitted to add attributes after existing attributes in the 'fields' section for a minor release increment.

Advisory: LS Records are expected to be long lasting data types within any implementation. If changes are required to an LS Record, it is usually expected to be a major version change only. Such a change would also typically require all calls or events incorporating the new record type to also be advanced by a major version change.

If a change is a minor version upgrade, and calls and events that use it are also provided with a minor version upgrade only to incorporate then new type, then careful consideration to backward compatible of the services must be given. This is because the new LS Record may insert extra data into a message in such a way that older calls or event schemas may misread the message. Be aware that it maybe necessary to distribute the new minor version call and event schemas to all participants to ensure expected behaviour.

If distribution of up to date schemas to **all participants can be guaranteed**, then more flexibility can be offered using the minor increment versioning rules for a specific implementation.

3.3.4 Service Fullname

The syntax for service fullname is namespace+(.)+name+version. E.g. The fullname for the 'systemregistration' service is '`ls.messages.core.systemregistration_v1_0`'.

4 Well-Known ls calls and ls events

4.1 Introduction

This section lists the core ls calls and ls events that are mandated for an implementation. They are used in the interoperability relationship between systems on a platform[2] and that platform's Service Registry. They are also used to achieve inter-platform interoperability. These are referred to as 'well-known' and their schemas, as well as a description of their functional behaviour, are specified in this section.

4.2 Mandated ls calls and ls events

The following section explains the mandated ls calls and ls events that must be supported in the Lean Services Architecture. The correct use of each call and event is explained in the next section. The values used to populate parameters in some well known calls and events are informed by the needs of specific implementations, so are not defined in this standard.

Table 1 lists mandated ls calls and whether a system, or the Service Registry (SR), or both, must support the call.

LS Call	System	SR	Description
NodeRegistration	X		Inform a system of the URI for the Service Registry and the URI for events communications[3]
RegisterSystem		X	Add a system entry to the Service Registry
DeRegisterSystem		X	Remove a system entry from the Service Registry
ReturnSystemStatus	X	X	Return the status of a system
ReturnAllServicesOverview	X	X	Return the summary list of services available
ReturnServiceDetail	X	X	Return the details of a specific service
RegisterService		X	Add a service entry to the Service Registry. Can be used for adding transient services, such as short-lived video streams.
DeregisterService		X	Remove a service entry from the Service Registry
ReturnServiceStatus	X	X	Return the status of a service
RegisterInterestInEvent		X	A system registers to receive events matching selection criteria
DeregisterInterestInEvent		X	A system de-registers event interest
ReturnEventsofInterest		X	Return all event-service associations
ReturnInterestInEvent		X	List all systems receiving events of a particular type

Table 1. Mandated Lean Services Calls

Table 2 lists the mandated Lean Services Events that must be issued by a system.

LS Event Name	Description
SystemStatusUpdate	Issued when a system has a status change
ServiceStatusUpdate	Issued when a service has a status change

Table 2. Mandated System Issued Lean Services Events

[2] Platform in this document is a computational platform (e.g. a computer) and not necessarily a military platform (e.g. a vehicle). It is considered that there should be only one Service Registry per platform.

[3] A component in an implementation must issue this call. This may or may not be the Service Registry.

Table 3 lists the mandated Lean Services Events that must be both issued and processed by the Service Registry.

LS Event Name	Description
PlatformAnnouncement	Issued when a platform has a status change
SystemStatusUpdate	Issued when a system has a status change
ServiceStatusUpdate	Issued when a service has a status change

Table 3. Mandated Service Registry Events

The Service Registry issues system and status updates regarding its own state in the same manner as any other system and will respond to a ReturnSystemStatus and ReturnServiceStatus about its own status as it would for any other system.

4.3 Operation of the Architecture

4.3.1 Service Registry

Node Registration

Using persisted information, on the startup of the Service Registry, the Service Registry issues a "noderegistration" call to each system to inform them about the node to use. The system is passed the Service Registry URI and the URI of the Event Handler to use when issuing events. When a Service Registry becomes aware of further systems, it makes a "noderegistration" call to each.

Note: the use of an automated system discovery technique is specific to an implementation. Such a technique could automatically inform the Service Registry when a new system has connect with or left the local network.

Populate the Service Registry

Next, to populate its database with the services provided by the systems, the Service Registry issues the ls calls "returnservicesoverview" and "returnservicedetail to each system and persists the returned data.

A system reports status changes in its operation and the status change of each of its call and event services using events "systemstatusupdate" and "servicestatusupdate". The Service Registry receives these events from the Event Handler and the status is persisted. A third party needing to know the status of a system or service uses the call "returnsystemstatus" or "returnservicestatus" on the Service Registry, which returns the last known information.

Note: rules regarding stale system and service information are specific to an implementation.

Remote Platform Registration

A Service Registry is informed of the presence of other nodes through the receipt of a "platformannouncement" event. A platform Service Registry can query another remote Service Registry for available services in the manner described above.

Note: rules regarding exchange of services information and regarding stale platform information are specific to an implementation.

4.3.2 The Event Handler and Lean Services Events

A system sends events to the URI supplied in the "noderegistration" call. A logical function called the Events Handler receives events. An event is re-distributed by the Events Handler to the URI for each system that has made a "RegisterInterestInEvent" call that matches the event characteristics. The Events Handler persists registrations and on re-start will continue to distribute events until receiving a "DeRegisterInterestInEvent" call.

Note: rules regarding stale event registration information are specific to an implementation.

4.3.3 Designing and Implementing Services Suitable for Lean Environments

The design and implementation of a specific Lean Services Architecture should be appropriate to its deployed environment and will vary. However the mandated calls and events MUST be supported from the perspective of a system at a particular node. The loosely coupled nature of Lean Services makes many existing techniques available for a system architect to exploit, including URI re-writing, re-direction and proxies. A system architect can insert architectural components into an implementation to provide these functions and other similar techniques.

The design of implementation-specific services should be appropriate to suit its deployed environment. For example in highly bandwidth-constrained environments, the design of the services should attempt to avoid verbose messages. Where it is unavoidable, it should use techniques to deliver such messages in a manner that appears transparent to systems.

The method to fulfil the mandated services will vary due to the environment. For example, obtaining the services provided by a remote Service Registries over low bandwidth links could be replaced with a custom solution, to avoid carrying the volume of data. A custom solution could use the 'platform type' parameter from the "platformannouncement" event, and knowing the services on that type of platform, could populate the local Service Registry without having to use unnecessary calls to the remote platform.

4.4 Mandated Services

4.4.1 {noderegistration}

Description:

Sent by the Service Registry at a node to inform a system of the URI of the Service Registry and the URI for events communications.

Schema:

```
{
"type"            : "lsdefinition",
"version"         : "1.0"
"namespace"       : "ls.messages.core",
"name"            : "noderegistration",
"lsservicetype"   : "CALL",
"parameters"      : [
                    {"sruri" : "string"},
                    {"srguid": "string"},
                    {"eventsuri": "string"},

                    ],
"response"        : null,
"error"           :
      [
            {"error" : "ls.messages.core.lserror"}
      ]
}
```

Parameter Name	Usage
sruri	The URI of the Service Registry.
srguid	The Service Registry may optionally provide a Globally Unique Identifier (GUID).
eventsuri	All events issued by the system are to be sent to the URI in eventsuri.

4.4.2 {registersystem}

Description:

Registers a system with the Service Registry. Following the registration, the Service Registry can query the system for services. Each system MUST have a unique URI.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "registersystem",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"system" : "ls.messages.core.systeminfo"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
      [
            {"error" : "ls.messages.core.lserror"}
      ]
}
```

Parameter Name	Usage
system	Systeminfo record
success	Success or failure of registration

{ls.messages.core.systeminfo}

```
{
   "type":"lsrecord",
   "version":"1.0",
   "namespace":"ls.messages.core",
   "name":"systeminfo",
   "fields":[
      {"uri"             : "string"},
      {"systemtype"      : "string"},
      {"name"            : "string"},
      {"description"     : "string"}
   ]
}
```

Parameter Name	Usage
uri	The URI of the system
systemtype	Type of system. The value is informed by the implementation.
name	Name of the system
description	Succinct human readable description of the system's primary business function

4.4.3 {deregistersystem}

Description:

Deregisters a system with the Service Registry. The service entries for the system can be removed from the Service Registry.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "deregistersystem",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"uri" : "string"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
      [
            {"error" : "ls.messages.core.lserror"}
      ]
}
```

Parameter Name	Usage
uri	The URI of the system

4.4.4 {returnsystemstatus}

Description:

Describes the current state of a system by returning a list of named states with true/false values and an optional textual message.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "returnsystemstatus",
  "lsservicetype" : "CALL",
  "parameters": null,
  "response":
      [
            {"statuses" : "List<ls.messages.core.genericstatusupdate>"}
      ],
  "error":
      [
            {"error" : "ls.messages.core.lserror"}
      ]
}
```

{genericstatusupdate}:

```
{
   "type":"lsrecord",
   "version":"1.0",
   "namespace":"ls.messages.core",
   "name":"genericstatusupdate",
   "fields":[
      {"statusname"           :     "string"},
      {"stringdata"           :     "string"},
      {"booleandata"          :     "boolean"}
   ]
}
```

Parameter Name	Usage
statusname	The name of the status
booleandata	Whether the status described in statusname is true or false
stringdata	Optional. A general-purpose message regarding the state, intended for human consumption.

The genericstatusupdate record is a status name and boolean pairing, with the Boolean value indicating whether the status is true or false.

There are two mandated statusname values to be supported by a Lean Services implementation, "systemactive" and "systemmessage".

"systemactive", with a booleandata of true or false, is used to communicate the operational availability of the primary business function of the system.

"systemmessage" is used to return a succinct human readable message. When used, booleandata is set to true and the stringdata parameter is populated with a human readable message.

The use of additional statusname values is informed by the implementation.

4.4.5 {returnallservicesoverview}

Description:

Return a summary list of services.

Schema:

```
{
"type"            : "lsdefinition",
"version"         : "1.0"
"namespace"       : "ls.messages.core",
"name"            : "returnallservicesoverview",
"lsservicetype"   : "CALL",
"parameters"      : null,
"response"        : [
                    {"services" : "list<ls.messages.core.serviceoverview>"}
                    ],
"error"           :
                    [
                    {"error" : "ls.messages.core.lserror"}
                    ]
}
```

{serviceoverview} record:

```
{
"type"              : "lsrecord",
"version"           : "1.0"
"namespace"         : "ls.messages.core",
"name"              : "serviceoverview",
"fields"            : [
                      {"servicefullname" : "string"},
                      {"uri"             : "string"},
                      {"servicetype"     : "string"},
                      ],
}
```

Parameter Name	Usage
servicefullname	The fullname of the service
uri	The URI of the service
servicetype	A definition of the type of service. The value is informed by the implementation

4.4.6 {returnservicedetail}

Description:

Return the detail of a specific service.

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "returnservicedetail",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"servicefullname" : "string"},
            {"uri" : "string"}
      ],
  "response":
      [
            {"servicedetail" : "ls.messages.core.servicedetail"}
      ],
  "error":
      [
            {"error" : "ls.messages.core.lserror"}
      ]
}
```

{servicedetail} record

```
{
   "type":"lsrecord",
   "version":"1.0",
   "namespace":"ls.messages.core",
   "name":"servicedetail",
   "fields":[
      {"servicefullname"      :     "string"},
      {"servicetype"          :     "string"},
      {"systemtype"           :     "string"},
      {"description"          :     "string"},
      {"uri"                  :     "string"},
      {"luid"                 :     "string"},
      {"schemafullname"       :     "string"},
      {"servicedefinition"    :     "string"}
   ]
}
```

Parameter Name	Usage
servicefullname	The fullname of the service
servicetype	A definition of the type of service. The value is informed by the implementation
systemtype	A definition of the primary business function of the system providing the service. The value is informed by the implementation
description	A succinct human-readable description of the service
uri	The URI of the service
luid	The luid (locally unique ID) parameter is reserved for future use.
schemafullname	The full name of the schema
servicedefinition	The schema

4.4.7 {registerservice}

Description:

Add a service entry to the Service Registry. Available for adding transient services, such as short-lived video streams.

Duplicate service URIs must be rejected by the Service Registry.

Schema:

```
{

  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "registerservice",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"servicefullname": "string"},
            {"uri"            : "string"},
            {"servicetype"    : "string"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]
}
```

Parameter Name	Usage
servicefullname	The fullname of the service
uri	The URI of the service
servicetype	A definition of the type of service. The value is informed by the implementation

4.4.8 {deregisterservice}

Description:

Remove a service entry from the Service Registry.

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "deregisterservice",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"servicefullname"      : "string"},
            {"uri"                  : "string"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]

}
```

Parameter Name	Usage
servicefullname	The fullname of the service
uri	The URI of the service

4.4.9 {returnservicestatus}

Description:

Return the status of a service.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "returnservicestatus",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"servicefullname" : "string"},
            {"uri" : "string"}
      ],
  "response":
      [
            {"status" : "ls.messages.core.servicestatus"}
      ],
  "error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]
}
```

{servicestatus}

```
{
   "type":"lsrecord"
   "version":"1.0",
   "namespace":"ls.messages.core",
   "name":"servicestatus",
   "fields":[
      {"servicefullname"      :          "string"},
      {"uri"                        :     "string"},
      {"status"                     :     "string"},
      {"statustext"                 :     "string"}
   ]
}
```

Parameter Name	Usage
servicefullname	The fullname of the service
uri	The URI of the service
status	A summary status. Informed by the implementation.
statustext	Optional. Human readable and succinct.

4.4.10 {registerinterestinevent}

Description:

Request to receive subsequent events matching the event interest parameters. The requestor is the source in the wrapper.

The return Boolean value is true if processed successfully.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "registerinterestinevent",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"assoc"   : "ls.messages.core.eventinterest"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]
}
```

{eventinterest} schema:

```
{
   "type":"lsrecord",
   "version":"1.0",
   "namespace":"ls.messages.core",
   "name":"eventinterest",
   "fields":[
      {"eventfullname"        :     "string"},
      {"eventuri"             :     "string"}
            ]
}
```

Parameter Name	Usage
eventfullname	The fullname of the event
eventuri	URI of the event source

4.4.11 {deregisterinterestinevent}

Description:

Cancels a register interest request.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "deregisterinterestinevent",
  "lsservicetype" : "CALL",
  "parameters":
      [
            {"assoc"   : "ls.messages.core.eventinterest"}
      ],
  "response":
      [
            {"success" : "boolean"}
      ],
  "error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]
}
```

4.4.12 {returneventsofinterest}

Description:

Return all event-to-service associations.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "returneventsofinterest",
  "lsservicetype" : "CALL",
  "parameters": null,
  "response":
      [
            {"associations" : "List<ls.messages.core.eventinterest>"}
      ],
"error":
            [
            {"error" : "ls.messages.core.lserror"}
            ]
}
```

4.4.13 {platformannouncement}

Description:

Event issued to inform of the presence or change of status of a node. On receipt of a platform announcement, a platform announcement event is returned to the source URI.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "platformannouncement",
  "lsservicetype" : "EVENT",
  "parameters":
      [
            {"nodeid"         : "string"},
            {"platformtype"   : "string"},
            {"nodeuri"        : "string"},
            {"status"         : "string"}
      ]
}
```

Parameter Name	Usage
nodeid	The id of the platform. Must be unique for each platform. Informed by the implementation.
platformtype	The type of platform. Informed by the implementation.
nodeuri	The URI of the platform. This represents a URI for the platform's Service Registry, to allow service information to be remotely queried.
status	A summary status for the platform. Informed by the implementation

4.4.14 {lserror}

Description:

Error message used within the core architecture.

Schema:

```
{
"type"            : "lsrecord",
"version"         : "1.0"
"namespace"       : "ls.messages.core",
"name"            : "lserror",
"fields"          : [
                        {"errortype" : "enum", "symbols":
                    ["NOTSUPPORTED","CALLERROR","SYSTEMERROR","DELIVERYFAILURE"]},
                      {"message" : "string"}
                    ]
}
```

Parameter Name	Usage
errortype	The type of error. NOTSUPPORTED – The call is not supported by the device. Includes calls made using the wrong major version number. CALLERROR – The system has reported an error while processing the call SYSTEMERROR – The architecture reports that the call was delivered to the system, but the system was unavailable DELIVERYFAILURE – The architecture reports that the message could not be delivered.
message	Optional. A succinct summary of the error for human consumption

4.4.15 {systemstatusupdate}

Description:

Issued when a system status changes.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "systemstatusupdate",
  "lsservicetype" : "EVENT",
  "parameters":
      [
            {"systemuri" : "string"},
            {"statuslist" : "list<ls.messages.core.genericstatusupdate>"}
      ]
}
```

Parameter Name	Usage
systemuri	The URI of the system issuing the status update.
statuslist	A list of genericstatusupdate records that describe the current system status

4.4.16 {servicestatusupdate}

Description:

Issued when a service status changes.

Schema:

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.messages.core",
  "name" : "servicestatusupdate",
  "lsservicetype" : "EVENT",
  "parameters":
      [
            {"status" : "ls.messages.core.servicestatus"}
      ]
}
```

Parameter Name	Usage
status	A service status record

5 Lean Services Message Generation

The technique used to exchange data with a Lean Services interface is a Lean Services message contained in a Lean Services wrapper.

The Lean Services wrapper provides the source, destination and return URI information and time of message assembly to allow delivery systems to carry the message without needing knowledge of the content. The carried Lean Services messages are then simpler in structure, as well as providing a future upgrade path to incorporate features such as security and safety features within a wrapper message.

All generated Lean Services messages use the schema type `lsbase`, and MUST use the serialisation method specified by Apache Avro version 1.7.5.

This section specifies the `lsbase` schemas for the Lean Services wrapper, call and event, and defines the transformation from a `lsbase` schema to a valid Avro schema.

5.1 lsbase Schema for Lean Services Call

The Lean Services call request, response and error messages are generated using the following `lsbase` schema.

Schema:

```
{
  "type"      : "lsbase",
  "version"   : "1.0",
  "namespace" : "ls.messages.base",
  "name"      : "lscall",
  "parameters":
      [
            {"servicefullname"         : "string"},
            {"type": "enum", "symbols" : ["EVENT", "REQUEST", "RESPONSE",
            "ERROR"]},
            {"callcontext"             : "string"},
            {"parameters"              : dataparameters}
      ]
}
```

servicefullname is populated using the LS Definition schema namespace and name attributes.

`lsmessagetype` defines the type of message being carried. This MUST be set to either REQUEST for a call request, RESPONSE for a call response or ERROR for a call error message. EVENT is not selected for a call but must be included here to allow generation and reading of an event binary message to occur correctly.

`callcontext` is populated when preparing a Lean Services call request, to allow the caller to associate a subsequently received RESPONSE or ERROR message with the original call REQUEST message. All Lean Services call responders MUST populate the RESPONSE or ERROR message with the same `callContext` provided by the issuer in the REQUEST message.

dataparameters is populated from the LS Definition, using the parameters attributes for a REQUEST, the response attributes for a RESPONSE or the error attributes for an ERROR.

A call with a schema response of null MUST return a response. If no response is required to a message, then an event would be used instead.

5.2 lsbase Schema for Lean Services Event

The schema for a Lean Services event is almost identical to the call, except there is no ‘callcontext’ attribute.

Schema:

```
{
  "type"      : "lsbase",
  "version"   : "1.0",
  "namespace" : "ls.messages.base",
  "name"      : "lsevent",
  "parameters":
      [
            {"servicefullname"         : "string"},
            {"type": "enum", "symbols" : ["EVENT"]},
            {"parameters"              : dataparameters}
      ]
}
```

servicefullname is populated using the LS Definition schema namespace and name attributes.

`lsmessagetype` is set to EVENT.

`dataparameters` is populated from the parameters attributes of the LS Definition.

5.3 lsbase Schema for Lean Services Wrapper

A Lean Services message is passed to a Lean Services interface inside a Lean Services wrapper.

Schema:

```
{
  "type"      : "lsbase",
  "version"   : "1.0",
  "namespace" : "ls.messages.base",
  "name"      : "lswrapper",
  "parameters":
      [
            {"messagetype": "enum", "symbols" : ["LSWRAPPER", "LSCALL",
"LSEVENT"]},
            {"zulutime-iso8601compact"       : "string"},
            {"sourceURI"                     : "string"},
            {"destinationURI"                : "string"},
            {"returnURI"                     : "string"},
            {"message"                       : "bytes"},
      ]
}
```

The `messagetype` parameter MUST match the message type being carried.

A Lean Services wrapper may contain a Lean Services wrapper to allow future specification implementation flexibility in delivery and provide available extensions for security and safety. This matches architectural patterns used elsewhere.

The assembly time of the wrapper message MUST be included in the Lean Services wrapper and is always recorded as Zulu time (Greenwich Mean Time). The presentation of the time uses UTC standard ISO 8601[4], with dashes and colons removed, i.e., YYYYMMDDHHMMSS.

For a Lean Services call REQUEST, the URI of the service is the destination, the return URI is where the call response or error is to be sent (usually the originator of the call). The Source URI is the originator of the call or event.

For a Lean Services call RESPONSE and ERROR, the URI of destination is the return URI from the original call request. The Return URI is not set. The Source URI is the responding URI.

[4] http://www.iso.org/iso/catalogue_detail?csnumber=40874

For a Lean Services event, the destination URI is the recipient's URI. The return URI is not set. The Source URI is the originator of the event.

`message` contains a generated Lean Services message or wrapper.

5.4 Lean Services Schema to Avro Schema Translation

To use an Avro-compatible software library to generate and/or de-serialise binary Lean Services messages, the substitutions and attribute syntax conversions from a Lean Services schema to a valid Avro schema are listed in this section.

5.4.1 Lean Services lsbase Schema Header

Lean Services Schema Representation

`"type":"lsbase",`
`"version":"1.0",`
`"namespace":"`***`namespace`***`",`
`"name":"`***`name`***`",`
`"parameters":`

Avro Schema Representation

`"type":"record",`
`"name":"`***`namespace+name`***`",`
`"fields":`

5.4.2 Lean Services Record Schema Header

Lean Services Schema Representation

`"type":"lsrecord",`
`"version":"1.0",`
`"namespace":"`***`namespace`***`",`
`"name":"`***`name`***`",`
`"fields":`

Avro Schema Representation

`"type":"record",`
`"name":"`***`namespace+name`***`",`
`"fields":`

5.4.3 Lean Services Name/Type Attribute

Lean Services Schema Representation
`"`***`parametername`***`": "`***`parametertype`***`"`
Avro Schema Representation
`"name": "`***`parametername`***`", "type": "`***`parametertype`***`"`

5.4.4 Lean Services Name/Type - List Attribute

Lean Services Schema Representation
`"`***`parametername`***`": "list<`***`parametertype`***`>"`
Avro Schema Representation
`"name": "`***`parametername`***`", "type": "array", "items": "`***`parametertype`***`"`

5.4.5 Lean Services Null Type Attribute

Lean Services Schema Representation
`"`***`parametername`***`": null`
Avro Schema Representation
`"`***`parametername`***`": null`

5.4.6 Lean Services Enum Type - Symbols Attribute

Lean Services Schema Representation
"symbols": [***JSON Array of JSON Strings***]
Avro Schema Representation
"symbols": [***JSON Array of JSON Strings***]

5.4.7 Lean Services Fixed Type - Size Attribute

Lean Services Schema Representation
"size": ***numberofbytes***
Avro Schema Representation
"size": ***numberofbytes***

The Lean Services ***namespace*** and ***name*** are concatenated with a dot (.) into the name attribute of the Avro record.

A Lean Services Definition may comprise of attributes that contain one or more LS Records. An Avro schema is always a single file, so the primitive attributes of all LS Records are concatenated (once transformations and substitutions listed above are applied), along with the individual primitive and list attributes in the definition. This creates a single set of attributes in the fields section of the final Avro schema. The attributes must be in the same sequence as the original Lean Services Definition to be processed correctly.

The Avro types of Map and Union are not used by Lean Services.

6 Transport Protocols

6.1 Mandated Transport Protocols

Lean services are transport protocol agnostic. However HTTP is the mandated transport protocol that MUST be supported by an implementation. It is used to ensure that interoperability is always possible between implementations, regardless of the chosen transport protocols used within implementations.

HTTP interoperability can be provided either by making every component support HTTP as well as each of the other implementation-specific chosen transport protocols, or by providing Gateways to convert between protocols.

6.2 Uniform Resource Indicator Syntax

The Uniform Resource Indicator (URI) is a string that describes the means to access a resource. The URI scheme name defines the scheme protocol to be used, followed by a hierarchy that uniquely identifies the resource for the protocol.

The URI for each service is persisted in the Service Registry along with a Lean Services Definition, so defining how to access the service and the data exchange with the service.

6.3 HTTP Transport Protocol

6.3.1 HTTP URI Specification

The scheme name is ‘http://’.

The hierarchy is the host name or IP address (optional port number), followed by a path to the resource, e.g. *http://192.168.0.55:80/ls/handling/local*.

6.3.2 HTTP Usage with Lean Services

The HTTP implementation MUST support HTTP 1.1[5] as a minimum, using MIME Base64 encoding for Lean Services messages in the HTTP message body. There are two communications patterns using HTTP with Lean Services. One is the Lean Services call mechanism used between systems and the other is the Lean Services event mechanism, which distributes Lean Services events to systems.

6.3.3 Call Mechanism

The calling system is the HTTP client and the recipient system is the HTTP server.

A Lean Services call request is sent as an HTTP POST message that includes the HTTP header fields of ‘Content-Length’, ‘Content-Type’ and ‘Host’, and the message body. Content length is set to the length of the encoded Lean Services message in the message body. Content type is set to ‘application/x-ls’. Host is set to the URI provided by the Service Registry.

A reply that returns a Lean Services call response or error uses an HTTP response status code of ‘200 OK’. Content length is set to the length of the encoded Lean Services message in the message body. Content type is set to ‘application/x-ls’.

[5] https://tools.ietf.org/html/rfc7230

6.3.4 Event Mechanism – From System to Event Handler

The Event Handler is a component that distributes events. It receives events issued by systems and distributes them to subscribing systems.

The Event Handler is the HTTP client and the system issuing a Lean Services event is the HTTP server. The event mechanism uses HTTP long polling.

The Event Handler initiates a long polling session sent as an HTTP POST containing the text ‘StartLongPolling’ in the message body and content type set to ‘text/plain’.

When the system has a Lean Services event to issue, the system uses an HTTP response status code of ‘200 OK’, with content Length set to the length of the encoded Lean Services event in the message body and content type set to ‘application/x-ls’.

If no event message is returned after a long polling timeout of 20 seconds, the client closes the HTTP session and immediately re-initiates a long polling session.

6.3.5 Event Mechanism –From Event Handler to System

The Event Handler is the HTTP client and the system receiving the Lean Services event is the HTTP server.

The event is sent by the Event Handler to the system using the HTTP POST as described in Section 6.3.3. The system replies using an HTTP response status code of ‘200 OK’ containing an empty body and content type set to ‘application/x-ls’.

1 Appendix A – Barrier Validation Rules

1.1 Overview

The Lean Services Architecture (LSA) is a schema-based request/response and event message protocol and supporting architecture designed to provide the features of a Services Orientated Architecture (SOA) in the operational and tactical military domain, or other similar environments using an implementation more suited to such conditions than existing solutions.

Different Trust Domains are domains that are either at different security classifications, or domains that have different management and ownership. To allow Lean Services messages to cross between Trust Domains it maybe necessary to check the message content in detail. This is to ensure that malicious data does not enter a domain and that unapproved data does not leave a domain.

A Barrier performs the checks. This specification defines the syntax that describes the checks that must be performed by the Barrier on each Lean Services message that is being moved between Trust Domains.

Two syntaxes are defined.

1. LPath, a language to allow a Lean Services message to be queried to return the data held by a specific field, and;
2. Barrier Validation Rules, an XML-based syntax defining the exact checks to be performed on a specific field.

A Barrier running Barrier Validation Rules can comprehensively check and validate the content of defined Lean Services messages to prevent malicious or inappropriate data from crossing between domains.

1.2 Barrier Validation Rules Summary

A Lean Services message is either a call or an event. Both Lean Services calls and events are defined using a schema called a Lean Services Definition. The schema namespace plus name plus version number provides a unique fully qualified name for every Lean Services message, allowing the Barrier to identify the message and apply the correct checks.

Barrier Validation Rules (BVR) are loaded onto the Barrier. There is one BVR for each Lean Services message schema used. The Barrier also has a copy of each Lean Services schema used. If a Lean Services message arrives that has no corresponding BVR and/or schema, then the message is rejected.

The BVR defines the checks to be performed on the content of the received message. Using both the schema and the BVR, the Barrier can determine if the data received:

- Is a correctly formatted Lean Services message as defined by the schema
- Has contents within the approved parameters of the BVR

A successful check allows the message to progress. A failed check will quarantine the message or perform other business activities as required by the accreditation.

1.3 Objective of this Specification

This specification defines the LPath and BVR syntax required to provide content check solutions, and Barrier systems, with sufficient information to allow comprehensive and robust content checking to occur. The specification follows the spirit of Lean Services in that it must be easy to read and understand and will require no unusual computer programming knowledge to implement.

The syntax is openly described to allow for use with any content check solution. A particular content check solution has 3 different ways to use BVRs:

- The BVR is used natively by the content checker as its content check syntax, allowing the BVR to be loaded directly on a Barrier and used without further conversion, allowing highest dynamism;
- The BVR is converted into a another content check syntax (requiring a post-process function) before being loaded onto a Barrier;
- The BVR is converted directly into software code, providing a static solution.

2 LPath Language

2.1 Context of Operation

LPath is a Lean Services message query language based on the principles of XPath[6], which is used to query XML documents. It uses the root position in a particular Lean Services schema as its starting point, follows a path through one or more schemas to its destination and returns the value. The language syntax can be extended as needed in the future. For the current specification, it is only necessary to return the value of a single field at a time.

2.2 Using LPath

To identify and extract a field value from a message, an LPath processor, which performs the LPath query, requires:

- A generated Lean Services message (a binary serialised message)
- The Lean Services schemas defining the contents of the message
- An LPath statement to identify a field to be read.

A processor utilises LPath to extract a field value from a Lean Services message by reading a path and using the appropriate Lean Services schema(s) to follow the path to its intended destination and returning the value.

A request, response, error and event message are delivered in a wrapper. If the message is still in the wrapper when being processed, then the path, starting at the root position of the wrapper, needs the wrapper and the message schemas, as well as any further schemas used by the message.

If the message has been extracted from the wrapper, then the path must start at the message schema root.

For the purposes of BVRs, the root value of the Definition is the start of the path of an event or call, so the top level is ‘parameters’, ‘response’ or ‘error’. The path then extends from that point down the hierarchy of the Lean Services schema(s) until it reads the required field.

2.2.1 LPath Syntax to Read a Primitive Field

For the Lean Services definition in Example 1, to access the integer of value1 or 2 requires a very short path, e.g., “parameters/value1”, which will return the integer value of value1.

[6] http://www.w3.org/TR/xpath/

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.2ic.exp",
  "name" : "exampleeventschema_v1_0",
  "lsservicetype" : "EVENT",
  "parameters":
       [
        {"value1"    : "int"},
        {"value2"    : "int"},
        {"person"    : "ls.2ic.exp.record.person_v1_0"}
       ]
}
```

Example 1. Sample Lean Services Event Definition.

If the type of a field is Lean Services Record, such as the 'person' field, then to read a primitive field in the record requires extending the path statement. The processor also needs to have the relevant Lean Services Record schema (see Example 2).

```
{
  "type" : "lsrecord",
  "version" : "1.0",
  "namespace" : "ls.2ic.exp.record",
  "name" : "person_v1_0",
  "fields":
       [
        {"firstname"    : "string"},
        {"lastname"     : "string"},
        {"age/years"    : "int"}
       ]
}
```

Example 2 Sample Lean Services Record 1.

E.g., "parameters/person/lastname". The lastname string value will be returned.

If the field name includes a forward slash then the field name MUST be placed in single quotes in the LPath statement.

E.g., " parameters/person/'age/years' " will return the integer value of the field "age/years".

2.2.2 LPath Syntax to Read a Record Field

To return a record field, the path to the record is used. E.g. "parameters/person". The return is the binary message containing the populated person record. This binary can then be checked using the record field type checks.

2.2.3 LPath Syntax to Read a List Field

To access a list field, the same approach is used as for a record field. See Example 3.

```
{
  "type" : "lsdefinition",
  "version" : "1.0",
  "namespace" : "ls.2ic.exp.call",
  "name" : "fetchListOfStaffatLocation_V1_0",
  "lsservicetype" : "CALL",
  "parameters":
       [
        {"Location" : "string"}
       ],
  "response":
       [
        {"staff"    : "list<ls.2ic.exp.record.person_v1_0>"}
       ],
  "error": null

}
```

Example 3 Sample Lean Services Call Definition.

E.g. “response/staff” will return the binary containing the list of person records meeting the request parameter. The binary can then be checked using list field type checks.

3 Barrier Validation Rule Syntax

3.1 Overview

The principles used to design the BVR syntax:

1) BVR syntax is simple to read.
    a. Lean Services messages should be compact, tactical and ‘lean’ in design, the messages should rarely be very complex, since that would place a load on the tactical communications network, processing computers and the software design and implementations. As such, the syntax of the BVR definition should be similarly simple & compact and allow a human to review the majority of BVRs without tool assistance. This will assist accreditation cases.
    b. A simple XML format is used to define BVRs and is defined in this specification.
2) Version 1 of BVRs will support single field validation only.
    a. Lean Services message design should tend to the approach that rather than overloading an existing Lean Services message with additional fields for a new situation, instead a new service should be designed. As such, optional fields should rarely, if ever, be required, since their use can introduce ambiguity in interpretation for implementations. The introduction of a new service in Lean Services, and SOAs in general, is a trivial act and much more straightforward than introducing consistency rules about the contents of a message. However a cross-field validation syntax extension can be subsequently introduced if required.
    b. *Note: it is possible that the Lean Services Architecture Specification may require additional syntax describing both optional fields and cross-field data completion rules in the future. Developing a cross-field validation syntax leveraging such work could provide greater continuity.*
3) The syntax must avoid any proprietary or patented techniques, allowing open use by any party.

3.2 Sections of the BVR

3.2.1 Header

A BVR is made up of:

- A header; identifies the BVR, issue number and the ls schema it is used against.
- Checks; containing one or more individual checks each against individual lean services parameters.

Example 4 shows the BVR header.

```
<bvr>
	<syntaxversion>1.0</syntaxversion>
	<schemafullname>ls.2ic.dop.inprogress.dop.2ic.exp.locationpos.m1.3</schemafullname>
	<bvrzulutimeofissue>20140912130752</bvrzulutimeofissue>
	<bvrissuenumber>3</bvrissuenumber>
	<messageminsize>178</messageminsize>
	<messagemaxsize>260</messagemaxsize>
```

Example 4 BVR Header.

Name	Usage
syntaxversion	Set to 1.0 for the released specification
schemafullname	The fully qualified name of the Lean Services schema that the BVR checks. This is namespace plus name plus (optional) version number
bvrzulutimeofissue	ISO 8601 time with spaces removed, no Z required at end. The date and time of the BVR release
bvrissuenumber	The issue number of the BVR. This is incremented each time a new BVR for a particular schema is issued. The term “version” has been deliberately avoided. A version number implies you should always use the latest, which may not be the case for a BVR.
messageminsize	The minimum number of bytes for the entire message
messagemaxsize	The maximum number of bytes for the entire message

Table 1. BVR Header Description.

3.2.2 Checks

3.2.3 Checks Header

```
<checks>
	<check type=”_TYPE_OF_CHECK_”>
		<fieldpath lpath=”/parameters/dopserviceid”>
```

Example 5 BVR Checks Header

Name	Usage
checks	Contains all the individual checks contained in the BVR
check	An individual check
Check options type	The type of field check being applied
Fieldpath	The field to be checked
fieldpath options lpath	The LPath syntax defining the Lean Services field being checked

Table 2. BVR Checks Header Description.

Checks follow the BVR header. Each check applies to a single field and allows a specific type of check to occur against a parameter, whether a primitive, enumerated, record or list type. Checks are added in turn, and behave as AND operators. Every check must pass.

Within a check, one or more field constraints can be applied. The field constraints behave as AND operators, so within a check every field check must pass.

The definitions of the field checks that can be assigned to each field type are listed below.

3.2.4 String

The type of check is ‘string’.

Field Check Name	Description
MinLength	The minimum length of the string
MaxLength	The maximum length of the string
Format	‘ALPHA’ or ‘NUMERIC’ or ‘MIXED’.
Case	‘UPPER’ or ‘LOWER’ or ‘BOTH’. ‘Both’ means that both upper and lower case are permitted.
Permitted options casesensitive	List of permitted words, with the option of casesensitive being ‘TRUE’ or ‘FALSE’
Forbidden options casesensitive	List of forbidden words, with the option of casesensitive being ‘TRUE’ or ‘FALSE’
Regex	A regular expression query.

Table 3. String Type Check Description.

```
<check type=”string”>
      <fieldpath lpath=”/parameters/manufacturerName”>
            <minlength>2</minlength>
            <maxlength>50</maxlength>
            <format>MIXED</format>
            <case>BOTH</case>
      </fieldpath>
</check>
```

Example 6 Checking a string field type.

```
<check type=”string”>
      <fieldpath lpath=”/parameters/powerStatus”>
            <permitted casesensitive=”TRUE”>
                  <value>ON</value>
                  <value>OFF</value>
            </permitted>
      </fieldpath>
</check>
```

Example 7 Checking the permitted values of a string field type.

3.2.5 Int, Long, Float and Double

The type of check is ‘int’ or ‘long’.

Field Check Name	Description
MinValue	The minimum value
MaxValue	The maximum value
Permitted	List of permitted values
Forbidden	List of forbidden values

Table 4. Int, Long, Float and Double Type Check Description.

3.2.6 Boolean

The type of check is ‘boolean’.

Field Check Name	Description
Permitted	The permitted value, either true or false (case insensitive)

Table 5. Boolean Type Check Description.

```
<check type=”boolean”>
	<fieldpath lpath=”/parameters/systemActive”>
		<permitted>TRUE</permitted>
	</fieldpath>
</check>
```

Example 8 **Checking the value of a boolean field type.**

3.2.7 Enumerated

The type of check is ‘enumerated’.

Field Check Name	Description
Permitted	List of permitted values (case insensitive)
Forbidden	List of forbidden values (case insensitive)

Table 6. Enumerated Type Check Description.

```
<check type=”enumerated”>
	<fieldpath lpath=”/parameters/platformType”>
		<permitted>
			<value>SOLDIER</value>
			<value>VEHICLE</value>
			<value>BASE</value>
			<value>UAV</value>
			<value>UGS</value>
		</permitted>
	</fieldpath>
</check>
```

Example 9 Checking the permitted values of an enumerated type.

3.2.8 Bytes

The type of check is ‘bytes’.

Field Check Name	Description
MinSize	The minimum size of the bytes field
MaxSize	The maximum size of the bytes field
ContentType	Validate the bytes content is of the type expected.

Table 7. Bytes Type Check Description.

ContentType informs the Content Checker that the bytes field contains a type of content that requires validation. It is proposed that the type name is held in namespace-style syntax, with an initial broad category with refinement to a precise type for checking.

Type Name	Description
image.jpeg	A JPEG image
image.jpeg.stanag4545	A JPEG image containing STANAG 4545 metadata
image.tiff	A TIFF image
document.pdf	A PDF document

Table 8. Example of Content Type Names.

3.2.9 Lean Services Record Field Type

A record field type contains a Lean Services Record. While using LPath you can define a path that reads down to the individual primitive fields within the record, content checking may wish to check the number of bytes used by the record at field level.

Field Check Name	Description
MinSize	The minimum byte size of the record
MaxSize	The maximum byte size of the record

Table 9. Complex Type Check Description.

3.2.10 List Field Type

The type of check is ‘list’.

A list field type contains a list of primitive or complex types and is used when it is not known at Lean Services schema design time how many items will be present, so cannot be pre-defined using schema fields. A typical use would be a list containing a variable number of items depending on a database query response.

Field Check Name	Description
MinItems	The minimum number of items in the list
MaxItems	The maximum number of items in the list
MinSize	The minimum total byte size of the list
MaxSize	The maximum total byte size of the list
MinItemSize	The minimum byte size of a single item in the list
MinItemSize	The maximum byte size of a single item in the list

Table 10. List Type Check Description

To determine the size of the individual items in the list, the items must be iterated through by the content checker and individually validated. The content check will identify from the schema and BVR that the field is a list. The content checker will iterate through each item in the list, performing the field level check on every item, whether primitive or complex.

```
<check type="list">
      <fieldpath lpath="/response/listofPersons">
            <minitems>1</minitems>
            <maxitems>10</maxitems>
                  <check type="string">
                  <fieldpath lpath="/response/listofPersons/firstname">
                        <minlength>2</minlength>
                        <maxlength>50</maxlength>
                        <alphanumeric>ALPHA</alphanumeric>
                        <case>UPPER</case>
                  </check>
                  <check type="string">
                  <fieldpath lpath="/response/listofPersons/lastname">
                        <minlength>2</minlength>
                        <maxlength>50</maxlength>
                        <alphanumeric>ALPHA</alphanumeric>
                        <case>UPPER</case>
                  </check>
                  <check type="int">
                  <fieldpath lpath="/response/listofPersons/age">
                        <minvalue>0</minvalue>
                        <maxvalue>120</maxvalue>
                  </check>
      </fieldpath>
</check>
```

Example 10 Checking permitted values within a List iteration function.

A future version of LPath could provide a syntax giving greater query control within a list, allowing specified items to be accessed and checked individually.

3.2.11 Complete BVR Example

Example 12 shows a single complete BVR, starting with the BVR header and leading into the individual checks.

```
<bvr>
	<syntaxversion>1.0</syntaxversion>
	<schemafullname>ls.acmecorp.sensorsystem_V1_0</schemafullname>
	<bvrzulutimeofissue>20140912130752</bvrzulutimeofissue>
	<bvrissuenumber>3</bvrissuenumber>
	<checks>
		<check type=”string”>
			<fieldpath lpath=”/parameters/manufacturerName”>
				<minlength>2</minlength>
				<maxlength>50</maxlength>
				<format>MIXED</format>
				<case>UPPER</case>
			</fieldpath>
		</check>
		<check type=”string”>
			<fieldpath lpath=”/parameters/powerStatus”>
				<permitted casesensitive=”TRUE”>
					<value>ON</value>
					<value>OFF</value>
				</permitted>
			</fieldpath>
		</check>
		<check type=”boolean”>
			<fieldpath lpath=”/parameters/systemActive”>
				<permitted>TRUE</permitted>
			</fieldpath>
		</check>
		<check type=”float”>
			<fieldpath lpath=”/parameters/position/latitude”>
				<minvalue>-90</minvalue>
				<maxvalue>90</maxvalue>
			</fieldpath>
		</check>
	</checks>
</bvr>
```

Example 11 A BVR Example.