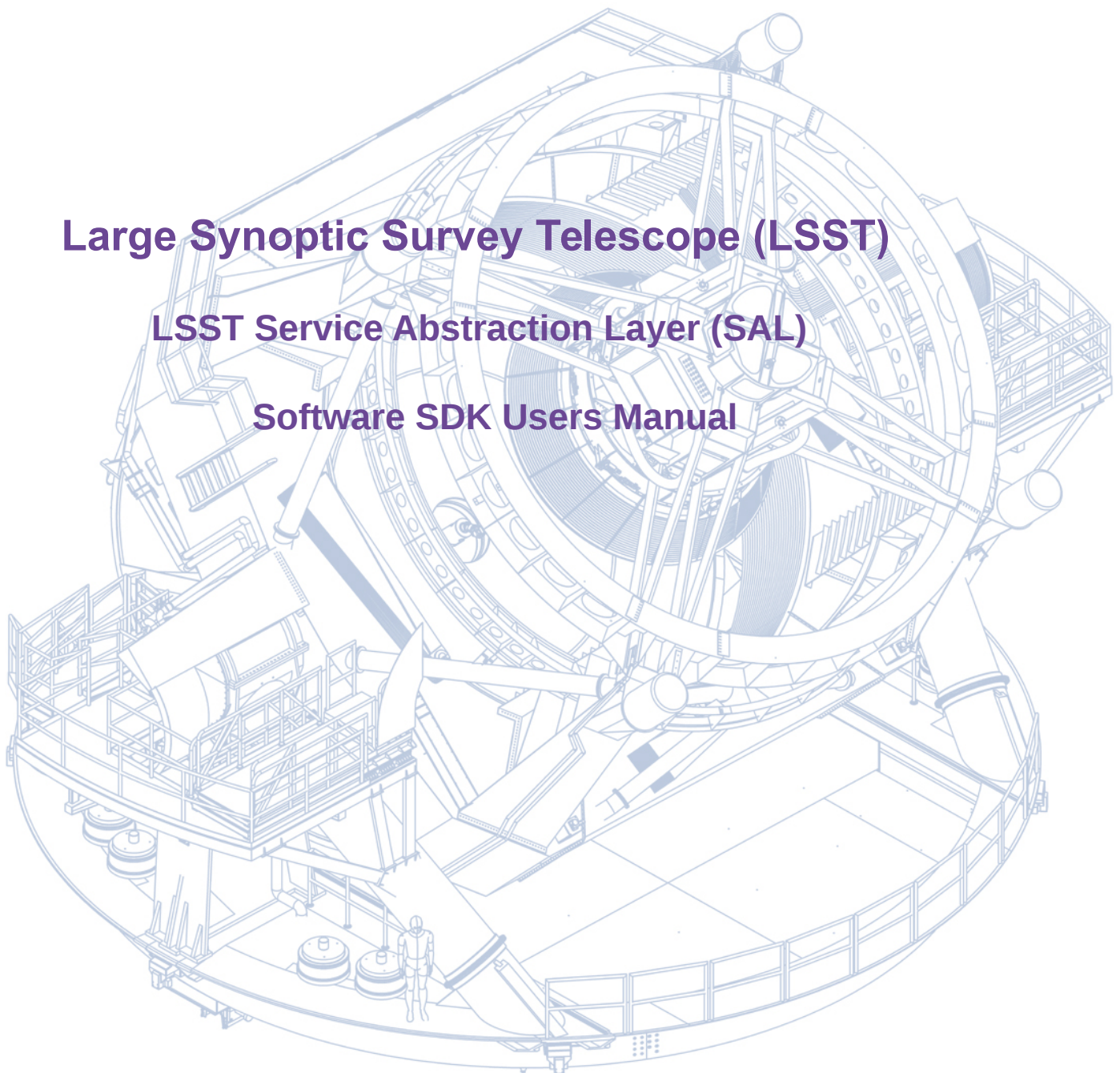


LARGE SYNOPTIC SURVEY TELESCOPE

# Large Synoptic Survey Telescope (LSST)

## LSST Service Abstraction Layer (SAL)

### Software SDK Users Manual



## Change Record

Version	Date	Description	Owner name
V1.0	12/13/2013	Initial Draft	D. Mills
V2.0	06/06/2014	First Release	D. Mills
V2.3	06/05/2015	Added detail about large-file handling	D. Mills
V2.4	07/27/2015	Major revision to document XML schema use	D. Mills

## Table of Contents

1 Introduction.....	4
2. Installation.....	5
2.1 In a Virtual Machine.....	7
3. Data Definition.....	8
3.1 Telemetry Definition.....	8
3.2 Command Definition.....	10
3.3 Log Event Definition.....	11
4. Using the SDK.....	12
4.1 Recommend sequence of operations.....	12
4.1.1 Step 1 – Definition.....	13
4.1.2 Step 2 – Validation.....	13
4.1.3 Step 3 – Update Structure and documentation.....	13
4.1.4 Step 4 – Code Generation.....	14
4.2 salgenerator Options.....	19
4.3 SAL API examples.....	20
5. Testing.....	20
5.1 Environment.....	20
5.2 Telemetry.....	21
5.3 Commands.....	21
5.4 Events.....	23
5.5 TCS pointing simulator.....	24
5.5.1 Simulated Subsystem Controllers.....	27
6. Application programming Interfaces.....	28
6.1. C++.....	28
6.2 Java.....	29
6.3 Python (Boost.python bindings).....	30
7.0 SAL XML Schema.....	33
7.1 Telemetry.....	33
7.1.1 telemetrySetType.....	33
7.1.2 telemetryType.....	33
7.1.3 telemetryItemType.....	34
7.2 Commands.....	35
7.2.1 commandSetType.....	35
7.2.2 commandType.....	35
7.2.3 commandItemType.....	36
7.3 Events.....	37
7.3.1 eventSetType.....	37
7.3.2 eventType.....	37
7.3.3 eventItemType.....	38

# 1 Introduction

This document briefly describes how to use the SAL SDK to generate application level code to utilize the supported services (Commanding, Telemetry and Events).

The SAL SDK should be installed on a modern (x86\_64) Linux computer. The current baseline recommended configuration is 64-bit CentOS 7.0.

The following packages should also be installed prior to working with the SDK (use either the rpm or yum package managers for CentOS, and apt-get , dpkg, or synaptic for Debian based systems). Appropriate rpms can be found in the rpms subdirectory of the unpacked SDK.

- g++
- make
- ncurses-libs
- xterm
- xorg-x11-fonts-misc
- java-1.7.0-openjdk-devel
- boost-python
- boost-python-devel
- maven
- python-devel
- swig
- tk-devel

The distribution includes dedicated versions of the following packages

- OpenSplice

All the services are built upon a framework of OpenSplice DDS. Code may be autogenerated for a variety of compiled and scripting languages, as well as template documentation, and components appropriate for ingest by other software engineering tools.

A comprehensive description of the SAL can be found in doc/LSE74-html, navigate to the directory with a web browser to view the hyper-linked documentation.

e.g.

firefox file:///path-to-installation/doc/LSE74-html/index.html

## 2. Installation

A minimum of 800Mb of disk space is required, and at least 1Gb is recommended to leave some space for building the test programs.

The default OpenSplice configuration requires that certain firewall rules are added, alternatively, shut down the firewall whilst testing.

For iptables : this can be done (as root) with the following commands

```
/etc/init.d/iptables stop
```

or by editing the

```
/etc/sysconfig/iptables
```

to add the following lines

```
-A INPUT -p udp -m udp --dport 250:251 -j ACCEPT  
-A INPUT -p udp -m udp --dport 7400:7411 -j ACCEPT  
-A OUTPUT -p udp -m udp --dport 250:251 -j ACCEPT  
-A OUTPUT -p udp -m udp --dport 7400:7411 -j ACCEPT
```

The iptables service should then be restarted

```
/etc/init.d/iptables restart
```

For firewalld : this can be done (as root) with the following commands

First, run the following command to find the default zone:

```
firewall-cmd --get-default-zone
```

Next, issue the following commands:

```
firewall-cmd --zone=public --add-port=250-251/udp --permanent  
firewall-cmd --zone=public --add-port=7400-7411/udp --permanent  
firewall-cmd --reload
```

Replace *public* with whatever the default zone says, if it is different.

Unpack the SAL tar archive in a location of choice (/opt is recommended),  
e.g. (in a terminal)

```
cd /opt  
tar xzf [location-of-sdk-archive]/salSDK-2.5.1_x86_64.tgz
```

and then add the SDK setup command.

```
source /opt/setup.env
```

to your bash login profile.

If you chose to install the SDK in a location other than /opt, then you will need to edit the first line of the setup.env script to reflect the actual location.  
e.g.

```
LSST_SDK_INSTALL=/home/saltester
```

The most common SDK usage consists of simple steps :

- 1) Define Telemetry, Command or Log activity (either using the SAL VM, or manually with an ascii text editor). For details of the SAL VM interface , please refer to Document-xxxxx.

The current prototypes for each subsystem can be used as a baseline, eg for the dome subsystem

```
cd $LSST_SDK_INSTALL  
cp lsstsal/scripts/xml-templates/dome/*.xml test/.
```

- 2) Generate the interface code using 'salgenerator'
- 3) Modify the autogenerated sample code to fit the application required.
- 4) Build if necessary, and test the sample programs

## 2.1 In a Virtual Machine

The SDK has been tested in a Virtual Machine environment (VirtualBox).  
To set up a VM appropriately for this usage :

1. In VM configuration , choose Bridged Adaptor for the network device
2. Add a sal user account during OS installation, the user should be an administrator
3. Choose Gnome Desktop + Development tools during OS installation
4. From VM menu , install Guest Additions
5. Once the OS has booted, enable the network
6. Verify the network is ok.
7. `sudo yum install xterm xorg-x11-fonts-misc java-1.7.0-openjdk-devel boost-python  
boost-python-devel maven python-devel tk-devel`
8. Configure (or disable) iptables and firewalld  
eg `systemctl disable iptables  
systemctl disable firewalld  
system stop iptables  
system stop firewalld`

## 3. Data Definition

### 3.1 Telemetry Definition

A very simple XML schema is used to define a telemetry topic. The topic is the smallest unit of information which can be exchanged using the SAL mechanisms.

The following Reserved words may NOT be used in names and will flag an error at the validation phase (once the SAL System Dictionary is finalized, the item names will also be validated for compliance with the dictionary).

Reserved words : *bstract any attribute boolean case char component const consumes context custom default double emits enum eventtype exception factory false finder fixed float getraises home import in inout interface local long module multiple native object octet oneway out primarykey private provides public publishes raises readonly sequence setraises short string struct supports switch true truncatable typedef typeid typeprefix union unsigned uses valuebase valuetype void wchar wstring*



e.g.

```
<SALTelemetry>
<Subsystem>hexapod</Subsystem>
<Version>2.5</Version>
<Author>A Developer</Author>
<EFDB_Topic>hexapod_LimitSensors</EFDB_Topic>
  <item>
    <EFDB_Name>liftoff</EFDB_Name>
    <Description></Description>
    <Frequency>0.054</Frequency>
    <IDL_Type>short</IDL_Type>
    <Units></Units>
    <Conversion></Conversion>
    <Count>18</Count>
  </item>
  <item>
    <EFDB_Name>limit</EFDB_Name>
    <Description></Description>
    <Frequency>0.054</Frequency>
    <IDL_Type>short</IDL_Type>
    <Units></Units>
    <Count>18</Count>
  </item>
</SALTelemetry>
```

## 3.2 Command Definition

The process of defining supported commands is similar to Telemetry using XML. The command aliases correspond to the ones listed in the relevant subsystem ICD. e.g.

```
<SALCommand>
  <Subsystem>hexapod</Subsystem>
  <Version>2.5</Version>
  <Author>salgenerator</Author>
  <EFDB_Topic>hexapod_command_configureAcceleration</EFDB_Topic>
  <Alias>configureAcceleration</Alias>
  <Device>drive</Device>
  <Property>acceleration</Property>
  <Action></Action>
  <Value></Value>
  <Explanation>http://sal.lsst.org/SAL/Commands/hexapod\_command\_configureAcceleration.html</Explanation>
  <item>
    <EFDB_Name>xmin</EFDB_Name>
    <Description> </Description>
    <IDL_Type>double</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>xmax</EFDB_Name>
    <Description> </Description>
    <IDL_Type>double</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>ymin</EFDB_Name>
    <Description> </Description>
    <IDL_Type>double</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>ymax</EFDB_Name>
    <Description> </Description>
    <IDL_Type>double</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
```

</SALCommand>

### 3.3 Log Event Definition

Events are defined in a similar fashion to commands. e.g

The Log Event aliases are as defined in the relevant ICD.

e.g.

```
<SALEvent>
  <Subsystem>hexapod</Subsystem>
  <Version>2.4</Version>
  <Author>salgenerator</Author>
  <EFDB_Topic>hexapod_logevent_limit</EFDB_Topic>
  <Alias>limit</Alias>
  <Explanation>http://sal.lsst.org/SAL/Events/hexapod\_logevent\_limit.html</Explanation>
  <item>
    <EFDB_Name>priority</EFDB_Name>
    <Description>Severity of the event</Description>
    <IDL_Type>long</IDL_Type>
    <Units>NA</Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>axis</EFDB_Name>
    <Description> </Description>
    <IDL_Type>string</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>limit</EFDB_Name>
    <Description> </Description>
    <IDL_Type>string</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
  <item>
    <EFDB_Name>type</EFDB_Name>
    <Description> </Description>
    <IDL_Type>string</IDL_Type>
    <Units> </Units>
    <Count>1</Count>
  </item>
</SALEvent>
```

## 4. Using the SDK

Once Telemetry/Command/Events have been defined , either using the SAL VM or hand edited,

e.g. for *skycam*, interface code and usage samples can be generated using the *salgenerator* tool. e.g.

```
salgenerator skycam validate  
salgenerator skycam sal cpp
```

would generate the c++ communications libraries to be linked with any user code which needs to interface with the **skycam** subsystem.

The "sal" keyword indicates SAL code generation is the required operation, the selected wrapper is cpp (GNU G++ compatible code is generated, other options are java, isocpp and python).

C++ code generation produces a shared library for type support and another for the SAL API. It also produces test executables to publish and subscribe to all defined Telemetry streams, and to send all defined Commands and log Events.

Java code generation produces a .jar class library for type support and another for the SAL API. It also produces .jar libraries to test publishing and subscribing to all defined Telemetry streams, and to send all defined Commands and log Events.

The Python option generates an import'able library. Simple example scripts to perform the major functions can be found later in this document.

### 4.1 Recommend sequence of operations

1. Create the XML Telemetry , Command, and Event definitions
2. Use the *salgenerator validate* operation
3. Use the *salgenerator html* operation
4. Use the *salgenerator sal* operation

5. Verify test programs run correctly
6. Build the SAL shared library / JAR for the subsystem
7. Begin simulation/implementation and testing

#### **4.1.1 Step 1 – Definition**

Use an XML editor to create/modify the set of subsystem xml files. Each file should be appropriately named and consists of a either Telemetry, Command, or Event definitions. The current prototypes for each subsystem can be found at [http://stash.lsstcorp.org/projects/TS/repos/ts\\_xml/browse](http://stash.lsstcorp.org/projects/TS/repos/ts_xml/browse).

#### **4.1.2 Step 2 – Validation**

Run the salgenerator tool validate option for the appropriate subsystem.

e.g. `salgenerator mount validate`

The successful completion of the validation phase results in the creation of the following files and directories.

- idl-templates – Corresponding IDL DDS topic definitions
- idl-templates/validated – validated and standardized idl
- idl-templates/validated/sal – idl modules for use with OpenSplice
- sql – database table definitions for telemetry
- xml – XML versions of the all telemetry definitions

#### **4.1.3 Step 3 – Update Structure and documentation**

Run the salgenerator html option for the appropriate subsystem.

e.g. `salgenerator mount html`

The successful completion of the html phase results in the creation of the following files and directories which may be used to update the SAL online configuration website. (See SAL VM documentation for upload details).

html – a set of directories, one per .idl file, with web forms for editing online  
a set of index-dbsimulate web page forms  
a set of index-simulate web page forms  
a set of sal-generator web page forms

#### 4.1.4 Step 4 – Code Generation

Run the salgenerator tool using the sal option for the appropriate subsystem.  
The sal option requires at least one target language to also be specified.  
The current target languages are cpp, isocpp, java and python.

Depending upon the target language , successful completion of the code generation results in the following output directories (e.g for mount)

e.g. salgenerator mount sal cpp

cpp -

mount:                - *common mount support files*

cpp  
isocpp  
java

mount/cpp:

ccpp_sal_mount.h	- main include file
libsacpp_mount_types.so	- dds type support library
Makefile.sacpp_mount_types	- type support makefile
sal_mount.cpp	- item access support
sal_mountDcps_impl.cpp	- type class implementation
sal_mount.idl	- type definition idl
sal_mountDcps.cpp	- type support interface
sal_mountDcps_impl.h	- type implementation headers
sal_mountSplDcps.cpp	- type support I/O
sal_mountDcps.h	- type interface headers
sal_mount.h	- type support class
sal_mountSplDcps.h	- type I/O headers

src

mount/cpp/src:

CheckStatus.cpp	- test dds status returns
CheckStatus.h	- test dds status headers
mountCommander.cpp	- command generator
mountController.cpp	- command processor
mountEvent.cpp	- event generator
mountEventLogger.cpp	- event logger
Makefile.sacpp_mount_cmd	- command support makefile
Makefile.sacpp_mount_event	- event support makefile
sacpp_mount_cmd	- <i>test program</i>
sacpp_mount_ctl	- <i>test program</i>
sacpp_mount_event	- <i>test program</i>
sacpp_mount_eventlog	- <i>test program</i>
sal_mount.h	- SAL class headers
sal_mountC.h	- SAL C support
sal_mount.cpp	- SAL class

mount\_TC:      - *specific to particular telemetry stream*

cpp  
isocpp  
java  
python

mount\_TC/cpp:

src  
standalone

mount\_TC/cpp/src:

CheckStatus.cpp	- check dds status class
CheckStatus.h	- check dds status header

mount\_TCDDataPublisher.cpp  
mount\_TCDDataSubscriber.cpp

- Actuators data publisher  
- Actuators data subscriber

mount\_TC/cpp/standalone:

Makefile  
Makefile.sacpp\_mount\_TC\_sub  
Makefile.sacpp\_mount\_TC\_pub  
sacpp\_mount\_sub  
sacpp\_mount\_pub  
src

- subscriber makefile  
- publisher makefile  
- *test program*  
- *test program*

mount\_TC/cpp/standalone/src:

e.g. salgenerator mount sal java

java -

mount/java:

classes  
mount  
Makefile.saj\_mount\_types  
saj\_mount\_types.jar  
sal\_mount.idl  
src

- compiled type classes  
- generated java types  
- makefile for types  
- type support classes  
- validated sal idl

mount/java/classes:

full set of java .class type support files

mount saj\_mount\_types.manifest

mount/java/classes/mount:

full set of .java type support files



mount/java/mount:

mount/java/src :

ErrorHandler.java	
mount_cmdctl.run	- run command tester
mount_event.run	- run event tester
mountCommander.java	- commander source
mountController.java	- command processor source
mountEvent.java	- event generator source
mount_EventLogger.java	- event logger source
Makefile.saj_mount_cmdctl	- command class makefile
Makefile.saj_mount_event	- event class makefile
sal_mount_cmdctl.jar	- command class source
sal_mount_event.jar	- event class source

mount\_TC/java: - *specific to particular telemetry stream*

Makefile  
src  
standalone

mount\_TC/java/src:

ErrorHandler.java	- error handler class source
mount_TCDataPublisher.java	- publisher class source
mount_TCDataSubscriber.java	- subscriber class source
org	

mount\_TC/java/src/org:

lsst

mount\_TC/java/src/org/lsst:

sal

mount\_TC/java/src/org/lsst/sal:

sal\_mount.java

- sal class for mount

mount\_TC/java/src/org/lsst/sal/mount:

Actuators

mount\_TC/java/src/org/lsst/sal/mount/Actuators:

mount\_TC/java/standalone:

mount\_TC.run

- *run test programs*

Makefile

Makefile.saj\_mount\_TC\_pub

- publication class makefile

Makefile.saj\_mount\_TC\_sub

- subscription class makefile

saj\_mount\_TC\_pub.jar

- telemetry publication class

saj\_mount\_TC\_sub.jar

- telemetry subscription class

e.g. salgenerator mount sal python

mount/cpp/src :

Makefile\_sacpp\_mount\_python

SALPY\_mount.cpp

- Boost.python wrapper

SALPY\_mount.so

- import'able python library

## 4.2 salgenerator Options

The salgenerator executes a variety of processes, depending upon the options selected.

validate	- check the XML files, generate validated IDL
html	- generate web form interfaces and documentation
labview	- generate Labview interface
sal [lang]	- generate SAL C++, Java, or Python wrappers
lib	- generate the SAL shared library for a subsystem
simd	- generate simd wrappers (deprecated)
shmem	- generate shared memory interface
sim	- generate simulation configuration
tcl	- generate tcl interface
icd	- generate ICD document
maven	- generate a maven project (per subsystem)
verbose	- be more verbose ;-)
db	- generate telemetry database table

for db the arguments required are

db start-time end-time interval

where the times are formatted like “2008-11-12 16:20:01”  
and the interval is in seconds

## 4.3 SAL API examples

The SAL code generation process also generates a comprehensive set of test programs so that correct operation of the interfaces can be verified.

Sample code is generated for the C++, Java, and Python target languages currently.

The sample code provides a simple command line test for

- publishing and subscription for each defined Telemetry type

- issuing and receiving each defined Command type

- generating and logging for each defined Event type.

In addition , GUI interfaces are provided to simplify the launching of Command and Event tests.

## 5. Testing

### 5.1 Environment

To check that the OpenSplice environment has been correctly initialized ; in a terminal, type

```
ipcs -a (lists shared memory segments)
```

```
idlpp (tests availability of idl processor)
```

To check that the SAL environment has been correctly initialized; in a terminal type

```
salgenerator (tests availability of sal processor/generator)
```

Verify that the network interface is configured and operating correctly.

Make sure that IPTABLES/Firewalld are properly configured (or disabled by issuing *systemctl stop iptables* and *systemctl stop firewalld* commands as root).

## 5.2 Telemetry

Once the salgenerator has been used to validate the definition files and generate the support libraries, there will be automatically built test programs available.

In all cases , log and diagnostic output from OpenSplice will be written to the files

ospl-info.log and ospl-error.log

in the directory where the test is run.

The following locations assume code has been built for the skycam subsystem support, there will be separate subdirectories for each Telemetry stream type.

For C++

skycam\_<telemetryType>/cpp/standalone/sacpp\_skycam\_<telemetryType>\_pub - publisher  
skycam\_<telemetryType>/cpp/standalone/sacpp\_skycam\_<telemetryType>\_sub - subscriber

For java

skycam\_<telemetryType>/java/standalone/skycam\_<telemetryType>.run  
- start publisher and subscriber

## 5.3 Commands

The following locations assume code has been built for mount subsystem support

For C++

mount/cpp/src/sacpp\_mount\_cmd - to send commands  
mount/cpp/src/sacpp\_mount\_ctrl - to process commands

For java

mount/java/src/mount\_cmdctl.run - starts command processor

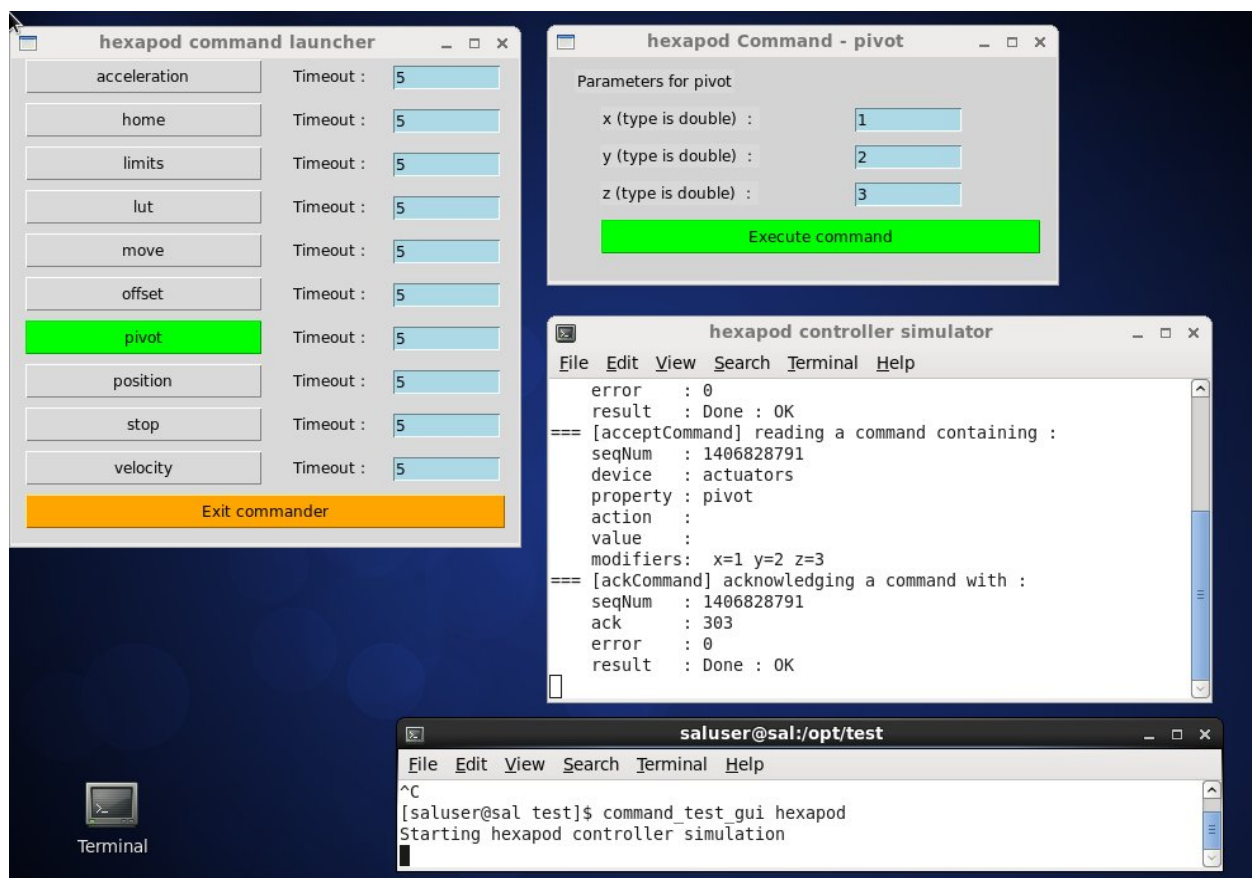
In addition a gui can be used to send all supported subsystem commands (with an associated processor to demonstrate reception of same). To start the gui e.g. for hexapod subsystem

For C++

## command\_test\_gui hexapod

The gui provides a window to select the command to run. If a command has optional values /modifiers, then a subwindow will open to allow their values to be entered.

A terminal window show the messages from a demo command processor which simply prints the contents of commands as they are received.



## 5.4 Events

The following locations assume code has been built for mount subsystem support

For C++

```
mount/cpp/src/sacpp_mount_event    - to generate events
mount/cpp/src/sacpp_mount_eventlog - to log the events
```

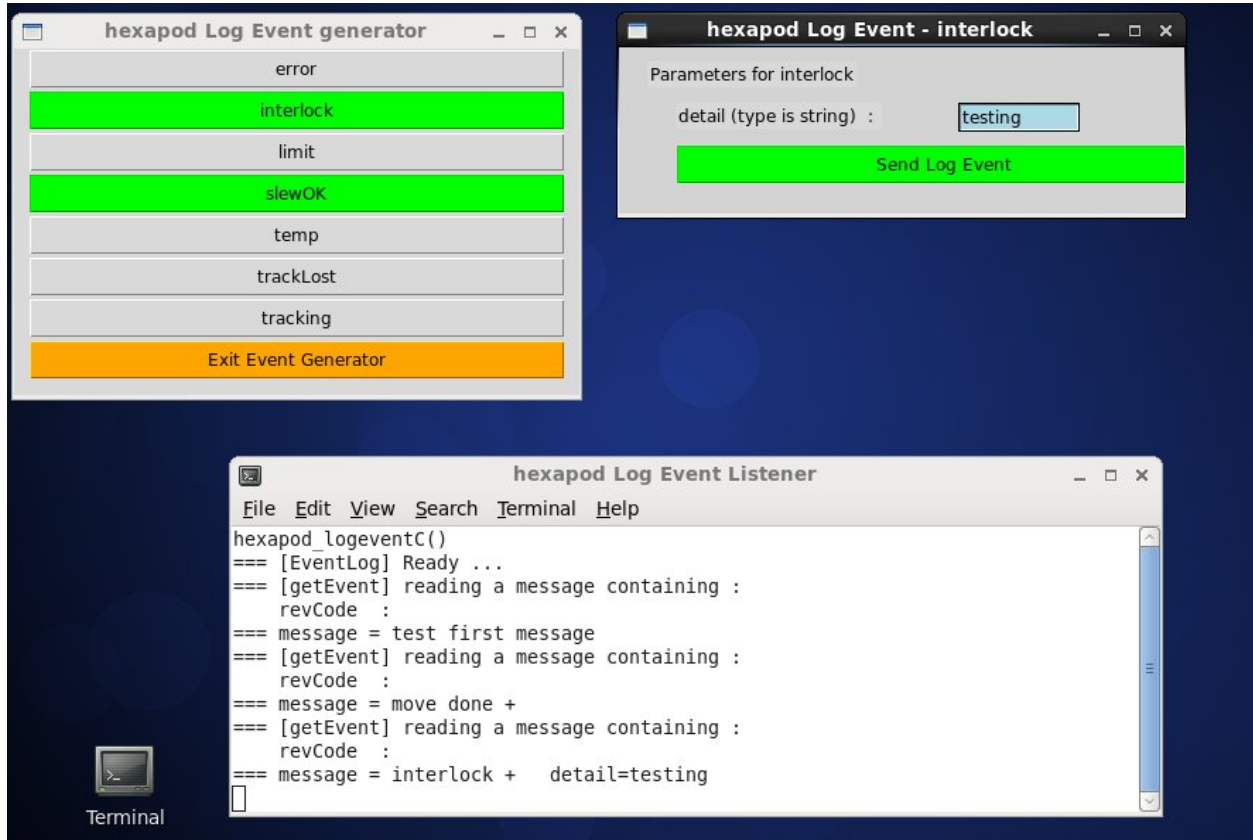
For java

```
mount/java/src/mount_events.run    - starts events processor
```

In addition a gui can be used to send all supported subsystem commands (with an associated processor to demonstrate reception of same). To start the gui e.g. for hexapod subsystem

For C++

```
logevent_test_gui hexapod
```



The gui provides a window to select the event to generate.. If an event has optional values /modifiers, then a subwindow will open to allow their values to be entered.  
A terminal window shows the messages from a demo event processor which simply prints the contents of events as they are received.

## 5.5 TCS pointing simulator

The SDK includes a TCS pointing kernel simulation, with associated gui's and data files.

This can be found in the

`$LSST_SDK_INSTALL/test/tcs/tcs`

directory tree.

The simulation consists of the following elements, all of which communicate using the SAL layer (C++).



- a). TCS pointing kernel with GUI and command line
- b). Opsim database log , used as input
- c). Mount controller simulator
- d). Camera controller simulator
- e). Hexapod controller simulators
- f). Dome controller simulator
- g). Rotator controller simulator

The simulation is started by

```
cd $LSST_SDK_INSTALL/test/tcs/tcs/bin  
./startdemo
```

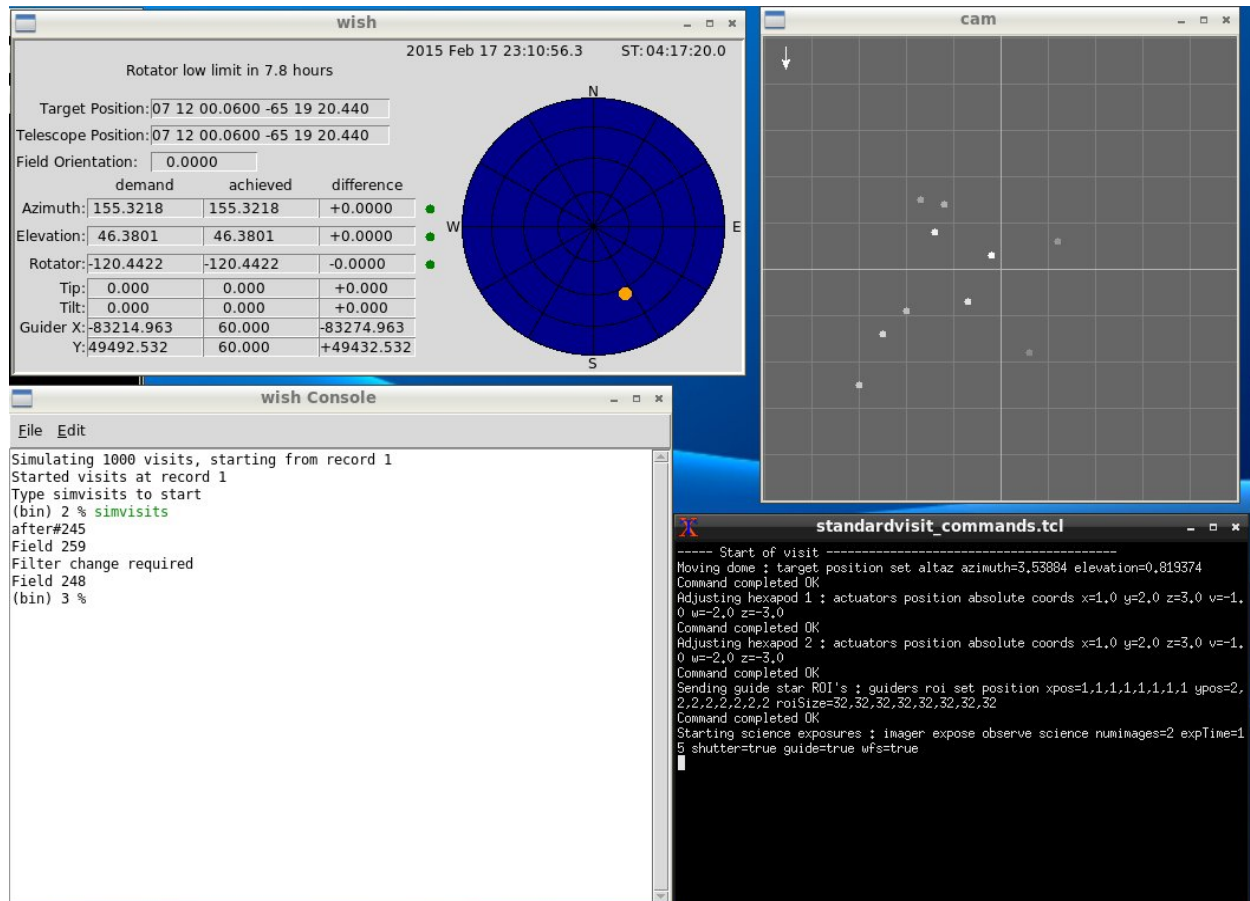
Once all the windows have deployed, the tcs simulator will automatically slew to the default target. Once it arrives (watch the GUI to follow it's progress), locate the command line interface window and type

```
simvisits
```

to start the simulated set of visits.

For each new visit, the simulator will send appropriately timed commands to each of the subsystem controller simulators.

## TCS Simulation GUI



## Standard Visit window

### 5.5.1 Simulated Subsystem Controllers

Customized controller simulators can also be used by specifying their location via environment variables

e.g.

```
export LSST_DOME_SIMULATOR /home/saldev/bin/dome_controller_test
```

would change any subsequent “startdemo” invocations to use the specified executable for the dome controller instead of the default one.

## 6. Application programming Interfaces

### 6.1. C++

Includes :

```
#include <string>
#include <sstream>
#include <iostream>
#include "SAL_mount.h"
#include "ccpp_sal_mount.h"
#include "os.h"
#include "example_main.h"
using namespace DDS;
using namespace <subsystem>; // substitute the actual subsystem name here
```

Public :

int putSample(<subsystem::telemetryType> data);	- publish telemetry sample
int getSample(<subsystem::telemetryTypeSeq> data);	- read next telemetry sample
int putSample_<telemetryType>(<subsystem::telemetryTypeC>*data);	- publish telemetry sample (C)
int getSample_<telemetryType>(<subsystem::telemetryTypeC>*data);	- read next telemetry sample (C)
void salTypeSupport(char *topicName);	- initialize type support
void salTelemetryPub(char *topicName);	- create telemetry publisher
void salTelemetrySub(char *topicName);	- create telemetry subscriber
void salEvent(char *topicName);	- create event object
int getResponse(<subsystem>::ackcmdSeq data);	- read command ack
int getEvent(<subsystem>::logeventSeq data);	- read event data
void salShutdown();	- tidyup
void salCommand();	- create command object
void salProcessor();	- create command processor object
int issueCommand(<subsystem>::command data);	- send a command
int issueCommandC(<subsystem>_commandC *data);	- send a command (C)
int ackCommand( int cmdSeqNum, long ack, long error, char *result );	- acknowledge a command
int acceptCommand( <subsystem>::commandSeq data);	- read next command
int acceptCommandC( <subsystem>_commandC *data);	- read next command (C)
int checkCommand( int cmdSeqNum );	- check command status
int cancelCommand( int cmdSeqNum );	- cancel command
int abortCommand( int cmdSeqNum );	- abort all commands
int waitForCompletion( int cmdSeqNum ,unsigned int timeout );	- wait for command to complete
int setDebugLevel( int level );	- change debug info level
int getDebugLevel( int level );	- get current debug info level
int getOrigin();	- get origin descriptor
int getProperty(stringproperty, stringvalue);	- get configuration item
int setProperty(stringproperty, stringvalue);	- set configuration item
int getPolicy(stringpolicy, stringvalue);	- get middleware policy item
int setPolicy(stringpolicy, stringvalue);	- set middleware policy item
void logError(int status);	- log middleware error
salTIME currentTime();	- get current timestamp
int logEvent( char *message, int priority );	- generate a log event

## 6.2 Java

Includes :

```
import <subsystem>.*;           //substitute actual subsystem name here
import org.lsst.sal.<SAL_subsystem>; //substitute actual subsystem name here
```

Public :

public void salTypeSupport(String topicName)	- initialize type support
public int putSample(<telemetryType> data)	- publish a telemetry sample
public int getSample(<telemetryType> data)	- read next telemetry sample
public void salTelemetryPub(String topicName)	- create telemetry publisher
public void salTelemetrySub(String topicName)	- create telemetry subscriber
public void logError(int status)	- log middleware error
public SAL_<subsystem>()	- create SAL object
public int issueCommand( command data )	- send a command
public int ackCommand( int cmdId, int ack, int error, String result )	- acknowledge a command
public int acceptCommand( <subsystem>.command data )	- read next command
public int checkCommand( int cmdSeqNum )	- check command status
public int getResponse(ackcmdSeqHolder data)	- read command ack
public int cancelCommand( int cmdSeqNum )	- cancel a command
public int abortCommand( int cmdSeqNum )	- abort all commands
public int waitForCompletion( int cmdSeqNum , int timeout )	- wait for command to complete
public int getEvent(logeventSeqHolder data)	- read next event data
public int logEvent( String message, int priority )	- generate an event
public int setDebugLevel( int level )	- set debug info level
public int getDebugLevel( int level )	- get debug info level
public int getOrigin()	- get origin descriptor
public int getProperty(String property, String value)	- get configuration item
public int setProperty(String property, String value)	- set configuration item
public void salCommand()	- create a command object
public void salProcessor()	- create command processor object
public void salShutdown()	- tidyup
public void salEvent(String topicName)	- create event object

## 6.3 Python (Boost.python bindings)

```
BOOST_PYTHON_MODULE(SALPY_mount){
    namespace bp = boost::python;

    bp::class_<subsystem_TelemetryTypeC>("subsystem_TelemetryTypeC")
        .add_property("telemetryItem", make_array(&<subsystem::TelemetryTypeC>::telemetryItem))
    bp::class_<SAL_subsystem>("SAL_subsystem", bp::init<int>())
        .def(bp::init<int>())
        .def(
            "abortCommand"
            , (::int ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::abortCommand )
            , ( bp::arg("cmdSeqNum") ) )
        .def(
            "acceptCommand"
            , (::int ( ::SAL_subsystem::* )( ::mount_commandC ) )( &::SAL_subsystem::acceptCommandC )
            , ( bp::arg("data") ) )
        .def(
            "ackCommand"
            , (::int ( ::SAL_subsystem::* )( int,::long,::long,char * ) )( &::SAL_subsystem::ackCommand )
            , ( bp::arg("cmdSeqNum"), bp::arg("ack"), bp::arg("error"), bp::arg("result") ) )
        .def(
            "cancelCommand"
            , (::int ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::cancelCommand )
            , ( bp::arg("cmdSeqNum") ) )
        .def(
            "checkCommand"
            , (::int ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::checkCommand )
            , ( bp::arg("cmdSeqNum") ) )
        .def(
            "currentTime"
            , (::salTIME ( ::SAL_subsystem::* )( ) )( &::SAL_subsystem::currentTime ) )
        .def(
            "getDebugLevel"
            , (int ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::getDebugLevel )
            , ( bp::arg("level") ) )
        .def(
            "getEvent"
            , (::int ( ::SAL_subsystem::* )( ::subsystem_logeventC ) )( &::SAL_subsystem::getEvent )
            , ( bp::arg("data") ) )
        .def(
            "getOrigin"
            , (int ( ::SAL_subsystem::* )( ) )( &::SAL_subsystem::getOrigin ) )

        .def(
            "getProperty"
            , (int ( ::SAL_subsystem::* )( char *,char * ) )( &::SAL_subsystem::getProperty )
            , ( bp::arg("property"), bp::arg("value") ) )
    }
```

```

.def(
    "getResponse"
    , (::int ( ::SAL_subsystem::* )( ::subsystem_ackcmdC ) )( &::SAL_subsystem::getResponse )
    , ( bp::arg("data") ) )
.def(
    "issueCommand"
    , (int ( ::SAL_subsystem::* )( ::subsystem_commandC ) )( &::SAL_subsystem::issueCommandC )
    , ( bp::arg("data") ) )
.def(
    "logError"
    , (void ( ::SAL_subsystem::* )( ::int ) )( &::SAL_subsystem::logError )
    , ( bp::arg("status") ) )
.def(
    "logEvent"
    , (::int ( ::SAL_subsystem::* )( char *,int ) )( &::SAL_subsystem::logEvent )
    , ( bp::arg("message"), bp::arg("priority") ) )
.def(
    "salCommand"
    , (void ( ::SAL_subsystem::* )( ) )( &::SAL_subsystem::salCommand ) )
.def(
    "salProcessor"
    , (void ( ::SAL_subsystem::* )( ) )( &::SAL_subsystem::salProcessor ) )
.def(
    "salShutdown"
    , (void ( ::SAL_subsystem::* )( ) )( &::SAL_subsystem::salShutdown ) )
.def(
    "salTelemetryPub"
    , (void ( ::SAL_subsystem::* )( char * ) )( &::SAL_subsystem::salTelemetryPub )
    , ( bp::arg("topicName") ) )
.def(
    "salTelemetrySub"
    , (void ( ::SAL_subsystem::* )( char * ) )( &::SAL_subsystem::salTelemetrySub )
    , ( bp::arg("topicName") ) )
.def(
    "salTypeSupport"
    , (void ( ::SAL_subsystem::* )( char * ) )( &::SAL_subsystem::salTypeSupport )
    , ( bp::arg("topicName") ) )

```

```

.def(
    "setDebugLevel"
    , (::int ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::setDebugLevel )
    , ( bp::arg("level") ) )
.def(
    "setProperty"
    , (::int ( ::SAL_subsystem::* )( char *,char * ) )( &::SAL_subsystem::setProperty )
    , ( bp::arg("property"), bp::arg("value") ) )
.def(
    "waitForCompletion"
    , (::int ( ::SAL_subsystem::* )( int,int ) )( &::SAL_subsystem::waitForCompletion )
    , ( bp::arg("cmdSeqNum"), bp::arg("timeout") ) )
.def(
    "get<TelemetryType" , &::SAL_subsystem::<getSampleTelemetryType> )
.def(
    "put<TelemetryType" , &::SAL_subsystem::<putSampleTelemetryType> )

bp::class_< subsystem_ackcmdC >( "subsystem_ackcmdC" )
    .def_readwrite( "ack", &subsystem_ackcmdC::ack )
    .def_readwrite( "error", &subsystem_ackcmdC::error )
    .def_readwrite( "result", &subsystem_ackcmdC::result )
    ;

bp::class_< subsystem_commandC >( "subsystem_commandC" )
    .def_readwrite( "device", &subsystem_commandC::device )
    .def_readwrite( "property", &subsystem_commandC::property )
    .def_readwrite( "action", &subsystem_commandC::action )
    .def_readwrite( "value", &subsystem_commandC::value )
    .def_readwrite( "modifiers", &subsystem_commandC::modifiers )
    ;
bp::class_< subsystem_logeventC >( "subsystem_logeventC" )
    .def_readwrite( "message", &subsystem_logeventC::message )
    ;

```



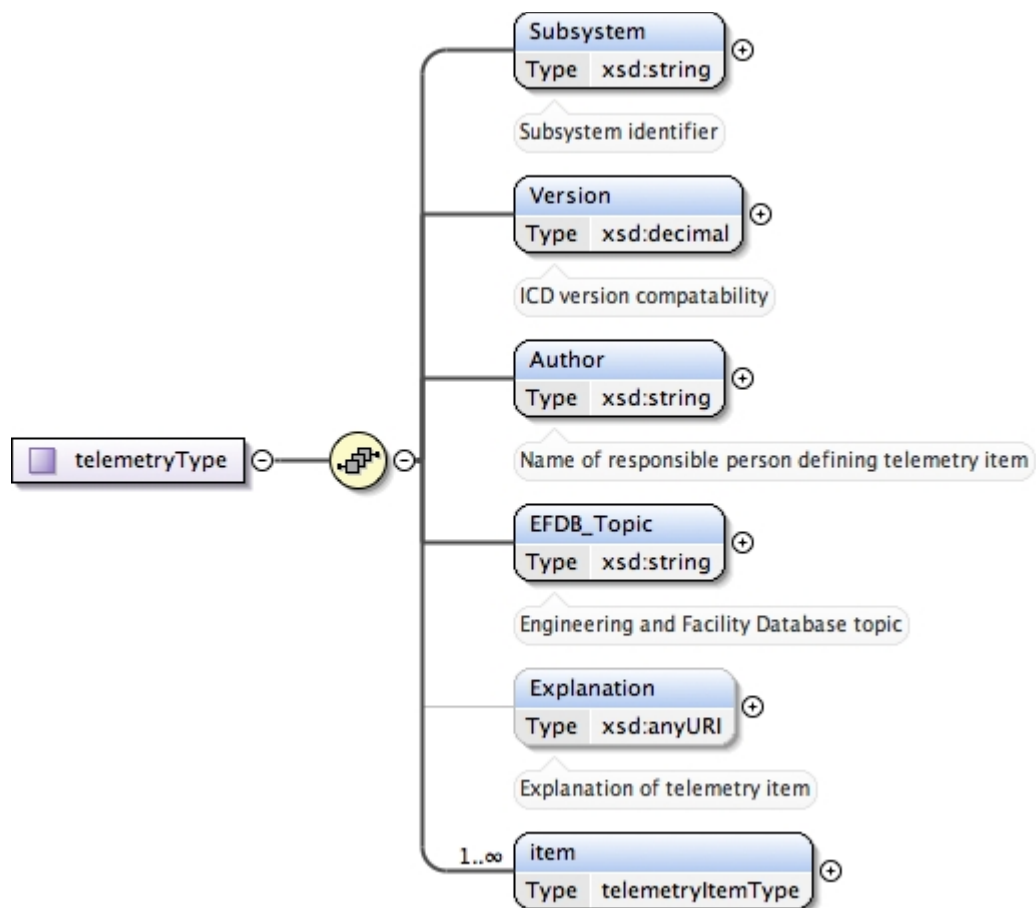
## 7.0 SAL XML Schema

### 7.1 Telemetry

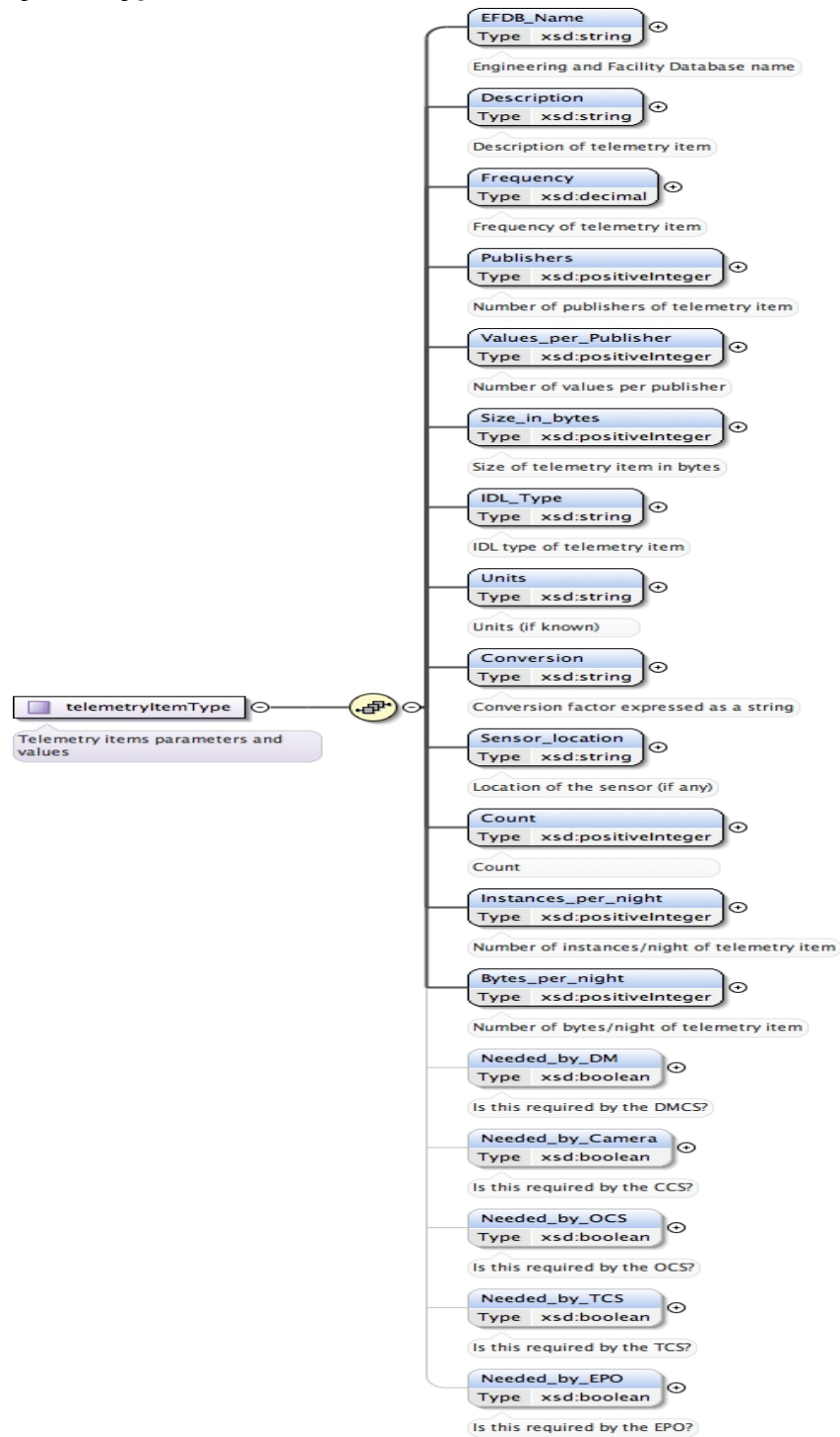
#### 7.1.1 telemetrySetType



#### 7.1.2 telemetryType

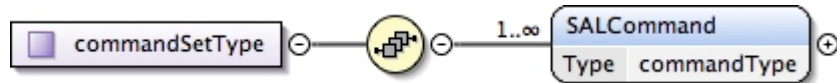


### 7.1.3 telemetryItemType

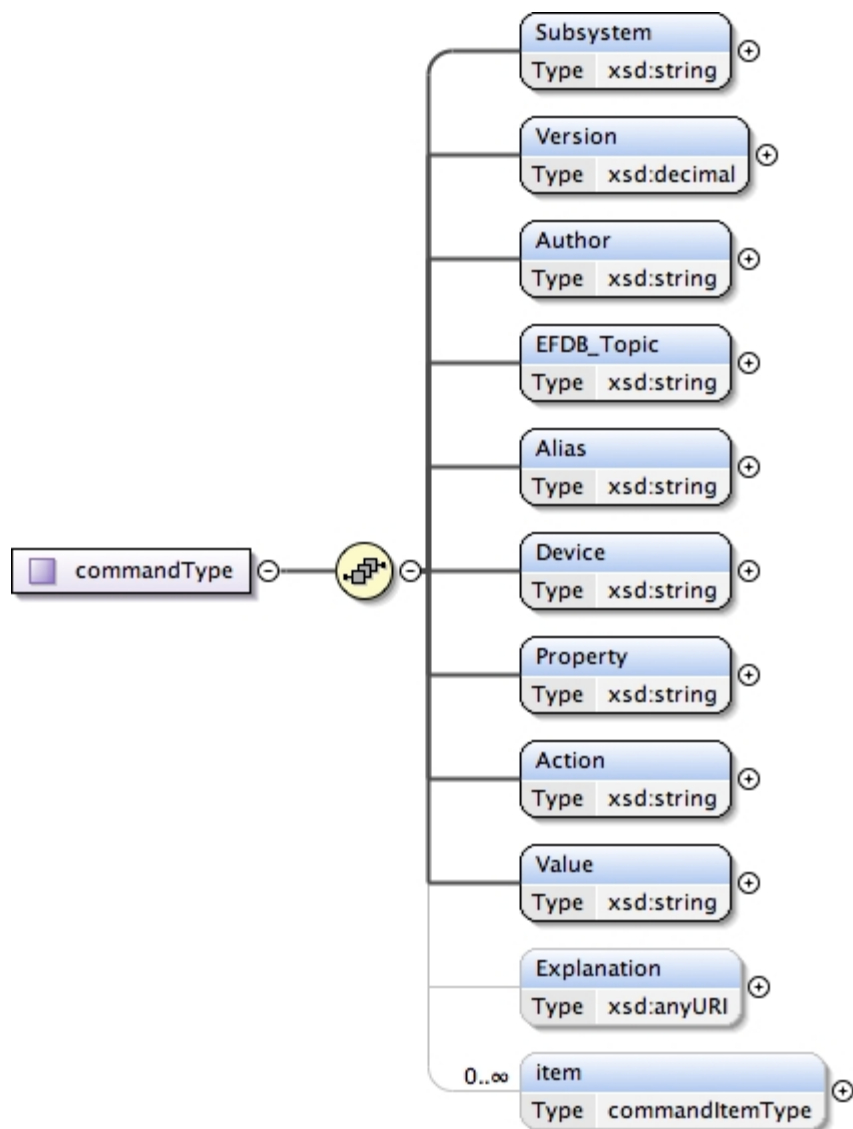


## 7.2 Commands

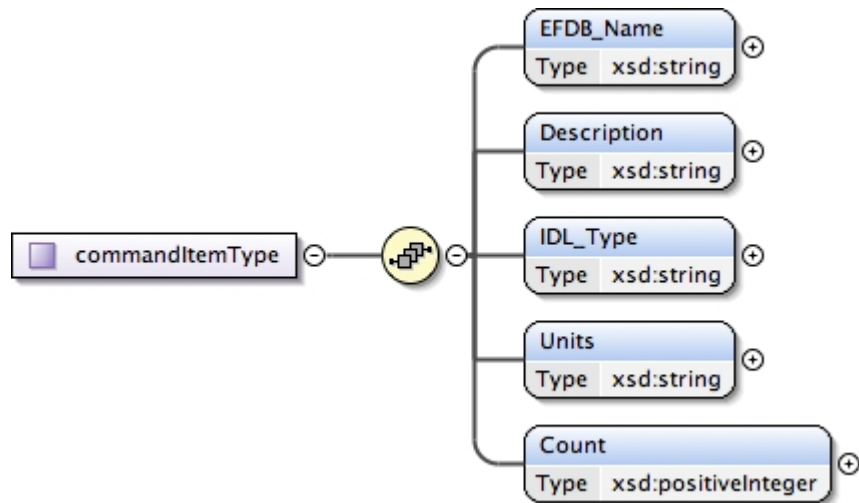
### 7.2.1 commandSetType



### 7.2.2 commandType



### 7.2.3 commandItemType

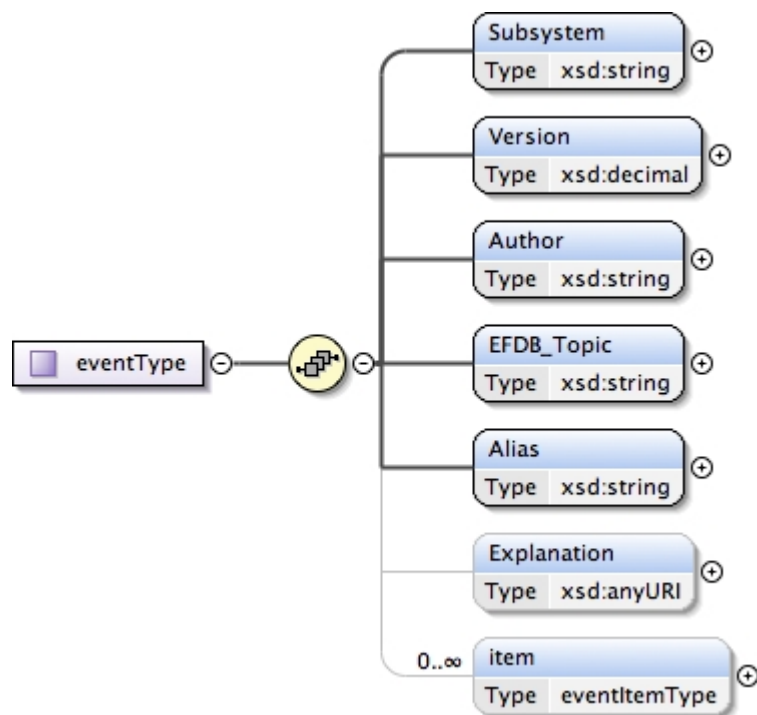


## 7.3 Events

### 7.3.1 eventSetType



### 7.3.2 eventType



### 7.3.3 eventItemType

