LSST Service Abstraction Layer (SAL) software SDK

Author : Dave Mills
Version : 2.2
Date : 03/20/2015

Contents

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 Logging).

The SAL SDK should install on any modern 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

The distribution includes dedicated versions of the following packages

- apache-maven
- boost
- openjdk
- OpenSplice
- python
- tcl/tk

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:///opt/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.

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-2.2.1_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.

2) Generate the interface code using 'salgenerator'

3) Modify the autogenerated sample code to fit the application required.

4) Build if necessary, and test sample programs

3. Data Definition

3.1 Telemetry Definition

A very simple version of IDL (Interface Definition Language) is used
to define a telemetry topic. The topic is the smallest unit of information which
can be exchanged using the SAL mechanisms.

e.g.

```
#
#  Define the telemetry topic for the IR skycam Application level data
#
#
struct skycam_IR_Application {
  string<16>  site;              // none ;    none ; Site where instrument is located
  long        ref_time;          // seconds ; none ; Reference time for this batch of images
  string<32>  gmt_time;          // none ;    none ; GMT version of REF_TIME
  float       del_t;             // seconds ; none ; Seconds after REF_TIME this image began
  string<16>  htchops;           // none ;    open|closed|moving|fault ; Hatch position
  float       encl_t1;           // degC ;    -10,30 ; Enclosure internal temperature 1, Celsius
  long        filpos;            // none ;    none ; Filter Wheel position
  string<16>  fildes;            // none ;    none ; Filter description
  float       bb_temps[3];       // degC ;    none ; Hatch Blackbody temperatures, Celsius
  float       fpa_t;             // degC ;    -60,0 ; IR camera FPA temperature, Celsius
  float       duration;          // seconds ; none ; Nominal exposure time, sec.
  string<128> imagefile;         // none ;    none ; URL to FITS image
};
```

This example illustrates the major features :

- comment lines have # in the first column

- all topics are named in a hierarchical fashion intended to describe their position within the LSST system.

- individual items in a topic are strongly "typed", and may be of types : string, short, long, float or double. String length is designated using <nnn> and arrays of other types are denoted using [nnn].

- Optional comments can be used to define metadata about items, the format is

    // units ; min,max|enumeration ; Brief descriptive text

The following IDL 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*

Alternatively an XML desription may be used
e.g.

```
<SALTelemetry>
<Subsystem>hexapod</Subsystem>
<Version>2.4</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 very simple. Commands are listed (one per line) in a text command_list file named according to the subsystem. e.g. command_list_dome

```
### COMMANDS
###-----------------------------------------------------------------------------------------
#type      device          property          action   value+modifiers  | alias
#
command  target          position                                    | target
                                                    string  azimuth
                                                    string  elevation
command  track           mode                       string  mode     | track
command  louvers          position set                               | louvers
                                                    double  angle[72]
command  shutter          position open                              | openShutter
command  shutter          position close                             | closeShutter
command  target           position                                   | park
command  target           position                                   | movetoCal
command  test             any               any                      | test
```

The format of a command definition is

```
command    device attribute action                          | alias
```

where value+modifiers are optional and may be primitives (int, string ,etc) or arrays of same. All alias , subsystem , device, property, action, and names must be present in the SAL System Dictionary. Each value/modifier is defined on a single line and is associated with the preceding “command” definition.

The command aliases correspond to the ones listed in the relevant subsystem ICD.

Alternatively an XML description may be used
e.g.

```
<SALCommand>
 <Subsystem>hexapod</Subsystem>
 <Version>2.4</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.htm
l</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 file event_list_dome contains

```
### EVENTS
###-------------------------------------------------------------------------------------
#type     id                  property            parameters                  | alias
###
###
event move                    ready                                           | slewReady
event move                    done                                            | slewOK
event move                    error                                           | slewError
event crawl                   lock                                            | crawling
event crawl                   lost                                            | crawlLost
event track                   lock                                            | tracking
event track                   lost                                            | trackLost
event louvers                 done                                            | lldvOK
event louvers                 error                                           | lldvError
event limit                   windscreen                                      | screenLimit
event limit                   jerk                                            | jerkLimit
event limit                   velocity                                        | VelLimit
event limit                   acceleration                                    | AccLimit
event limit                   position                                        | posLimit
                                                  string    device
                                                  string    limit
                                                  string    type
event temperature                                                             | tempError
                                                  string  device
                                                  long    severity
event power                                                                   | powerError
                                                  string  device
                                                  long    severity
event interlock                                                               | interlock
                                                  string  detail
```

Optional parameters may be associated with each event, one per line, following the particular “event” definition.

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

Alternatively an XML description may be used
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_IR_Application.idl***, interface code and usage samples can be generated using the ***salgenerator*** tool. e.g.

salgenerator skycam_IR_Application.idl validated
salgenerator skycam_IR_Application.idl sal cpp

would generate the c++ communications libraries to be linked with any user code which needs to publish or subscribe to the telemetry stream **skycam_IR_Application**.

The item can be wildcarded, so for example

salgenerator skycam*.idl sal cpp

would generate a library appropriate for all skycam related items as well as test programs for each skycam telemetry stream.

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 IDL and 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. Begin simulation/implementation and testing

4.1.1 Step 1 – Definition

Use a text editor to create a set of .idl files. Each file should be appropriately named and consists of a single telemetry stream definition. The file name should be constructed using the subsystem name, and a descriptive component. e.g. mount_TC.idl

4.1.2 Step 2 – Validation

Run the salgenerator tool on each .idl file using the validate option.

e.g. salgenerator mount_TC.idl validate

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

idl-templates – copy of current input id
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 telemetry definitions

4.1.3 Step 3 – Update Strucuture and documentation

Run the salgenerator tool on each .idl file using the html option

e.g. salgenerator mount_TC.idl 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 on each of the .idl files using the sal option.
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_TC)

e.g. salgenerator mount_TC.idl 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	- *test program*
sacpp_mount_ctl	- *test program*
sacpp_mount_event	- *test program*
sacpp_mount_eventlog	- *test program*
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_TCDataPublisher.cpp	- Actuators data publisher
mount_TCDataSubscriber.cpp	- Actuators data subscriber

mount_TC/cpp/standalone:

Makefile	
Makefile.sacpp_mount_TC_sub	- subscriber makefile
Makefile.sacpp_mount_TC_pub	- publisher makefile
sacpp_mount_sub	- *test program*
sacpp_mount_pub	- *test program*
src	

mount_TC/cpp/standalone/src:

e.g. salgenerator mount_TC.idl sal isocpp

isocpp -

mount/isocpp:

libISO_Cxx_mount_Typesupport.so	- mount support Shared library
Makefile.ISO_Cxx_mount_Typesupport	- type support makefile
sal_mountDcps.h	- type support headers
sal_mount.h	- main include file
sal_mount_Dcps.hpp	- type support classes
sal_mount.idl	- type definition idl
sal_mount.cpp	- SAL mount object class
sal_mountDcps_impl.cpp	- type support interface classes
sal_mountSplDcps.cpp	- I/O support classes
sal_mountDcps.cpp	- type support classes
sal_mountDcps_impl.h	- type interface headers
sal_mountSplDcps.h	- I/O support headers
src	

mount_TC/isocpp: - *specific to particular telemetry stream*

implementation.cpp	- support classes
implementation.hpp	- support headers
Makefile	- makefile for test programs
publisher.cpp	- publisher source
subscriber.cpp	- subscriber source

e.g. salgenerator mount_TC.idl sal java

java -

mount/java:

classes	- compiled type classes
mount	- generated java types
Makefile.saj_mount_types	- makefile fior types
saj_mount_types.jar	- type support classes
sal_mount.idl	- validated sal idl
src	

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_TC.idl sal python

mount/cpp/src :

Makefile_sacpp_mount_python	
SALPY_mount.cpp	- Boost.python wrapper
SALPY_mount.so	- import'able python library

4.1.5 Step 5 – Verification , testing and Integration

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 restarted

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.

4.2 salgenerator Options

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

validate	- check the .idl files, , command_list and event_list
html	- generate web form interfaces and documentation
labview	- generate Labview interface
sal [lang]	- generate SAL C++, Java, or Python wrappers
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 examples

4.3.1 Example 1 – Publishing telemetry

Using C++

```
mount_TCC  myData;
long i,iseq;
SAL_mount mgr = SAL_Mount();

//create publisher
mgr.salTelemetryPub(“mount_TC”);

//set data values
for (I-0;i<18;i++) {myData.Raw[i] = i;}
for (I-0;i<18;i++) {myData.Calibrated[i] = i;}

 //publish the sample
mgr.putSample_TC(&myData);

//tidyup
mgr.salShutdown();
```

Using Java

```
// initialize
SAL_mount mgr =  SAL_mount();

// create publisher
mgr.salTelemetryPub("mount_TC");
Actuators myData = new Actuators();

//set data values
for (int i-0;i<18;i++) {myData.Raw[i] = i;}
for (int i-0;i<18;i++) {myData.Calibrated[i] = i;}

 //publish the sample
mgr.putSample(myData);

//tidyup
mgr.salShutdown();
```

Using Python

```
# initialize
from SALPY_mount import *
mgr=SAL_mount()
myData=mount_TC()

# create publisher
mgr.salTelemetryPub("mount_TC")

#set data values
for i in range (0,18)
   myData.Raw[i]=i
   myData.Calibrated[i]=i

# publish the sample
mgr.putSample_TC(myData)

# tidyup
mgr.salShutdown()
```

4.3.2 Example 2 – Sending a command

Using C++

```
// initialize
 SAL_mount cmd = SAL_mount();

 // create command object
  cmd.salCommand();
  mount::command command; /* Example on Stack */
  command.device = DDS::string_dup(device);
  command.property = DDS::string_dup(property);
  command.action = DDS::string_dup(action);
  command.value = DDS::string_dup(value);
  command.modifiers = DDS::string_dup(modifiers);

  // send the command
  cmdId = cmd.issueCommand(command);

  // wait for ack/completion
   os_nanoSleep(delay_1s);
  status = cmd.waitForCompletion(cmdId, timeout);
```

```
// tidyup
cmd.salShutdown();
```

Using Java

```
// initialize
SAL_mount mgr = new SAL_mount();

// Issue command
int cmdId=1;
int timeout=5; //seconds
int status=0;

// create command object
mgr.salCommand();
static command  = new mount.command();

command.device =   “rotator”;
command.property =  “angle”;
command.action =    “move”;
command.value =     “23.0”
command.modifiers = “”;

cmdId = mgr.issueCommand(command);

Thread.sleep(1000);
status = mgr.waitForCompletion(cmdId, timeout);

// tidyup
mgr.salShutdown();
```

Using Python

```
# intialize
from SALPY_mount import *
mgr=SAL_mount()

# create command object
mgr.salCommand()
command=mount_CommandC()
command.device=”rotater”
command.property=”angle”
command.action=”move”
command.value=”23.0”
command.modifiers=””


# send the command
cmdId=mgr.issueCommand(command)

# wait for ack/completion
status=cmd.waitForCompletion(cmdId)

# tidyup
mgr.salShutdown()
```

4.3.3 Example 3 – Receiving commands

Using C++

```
// initialize
int timeout=5;
SAL_mount cmd = SAL_mount();

// create command object
cmd.salProcessor();
mount::commandSeq command; /* Example on Stack */

 // wait for a command to arrive
 cmdId = cmd.acceptCommand(command);
 if (cmdId > 0) {
    if (timeout > 0) {
      // take some time to complete
      cmd.ackCommand(cmdId, SAL__CMD_INPROGRESS, timeout, "Ack : OK");
      os_nanoSleep(delay);
    }
    // pass back command completion ack
    cmd.ackCommand(cmdId, SAL__CMD_COMPLETE, 0, "Done : OK");
 }

// tiduyp
cmd.salShutdown();
```

Using Java

```
// initialize
SAL_mount cmd = new SAL_mount();
int status = SAL__OK;
int cmdId       = 0;
int timeout     = 0;

// Initialize
cmd.salProcessor();
command = new mount::commandSeq();

// wait for command to arrive
cmdId = cmd.acceptCommand(command);
if (cmdId > 0) {
  if (timeout > 0) {
    // take some time to complete
     cmd.ackCommand(cmdId, SAL__CMD_INPROGRESS, timeout, "Ack : OK");
     Thread.sleep(timeout);
   }
    //  pass back command completion ack
    cmd.ackCommand(cmdId, SAL__CMD_COMPLETE, 0, "Done : OK");
}

// tidyup
cmd.salShutdown();
```

Using Python

```
# initialize
from SALPY_mount import *
cmd=SAL_mount()
cmd.salProcessor()
command=mount_CommandC()

// wait for a command to arrive
cmdId=cmd.acceptCommand(command)

// pass back command completion ack
cmd.ackCommand(cmdId,SAL__OK , 0 , ”OK”)

// tiduyp
cmd.salShutdown()
```

4.3.4 Example 4 – Generating an Event

Using C++

```
int priority = SAL__EVENT_INFO;
SAL_mount mgr = SAL_mount();
string message=”Testing the Event mechanism”;

// generate event
mgr.logEvent(message.c_str(), priority);
cout << "=== Event " << alias << " generated = " << message << endl;

// tidyup
mgr.salShutdown();
```

Using Java

```
// Initialize
int status=0;
SAL_mount mgr = new SAL_mount();

String msg=”Testing the Event mechanism”;
int priority=1;
status = mgr.logEvent(msg,priority);

// tidyup
```

```
mgr.salShutdown();
```

Using Python

```
from SALPY_mount import *
h=SAL_mount(1)
h.logEvent("Testing the Event mechanism",1)
```

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)

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 am 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 show 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

/opt/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
h). M2 controller simulator

The simulation is started by

cd /opt/test/tcs/tcs/bin
./startdemo

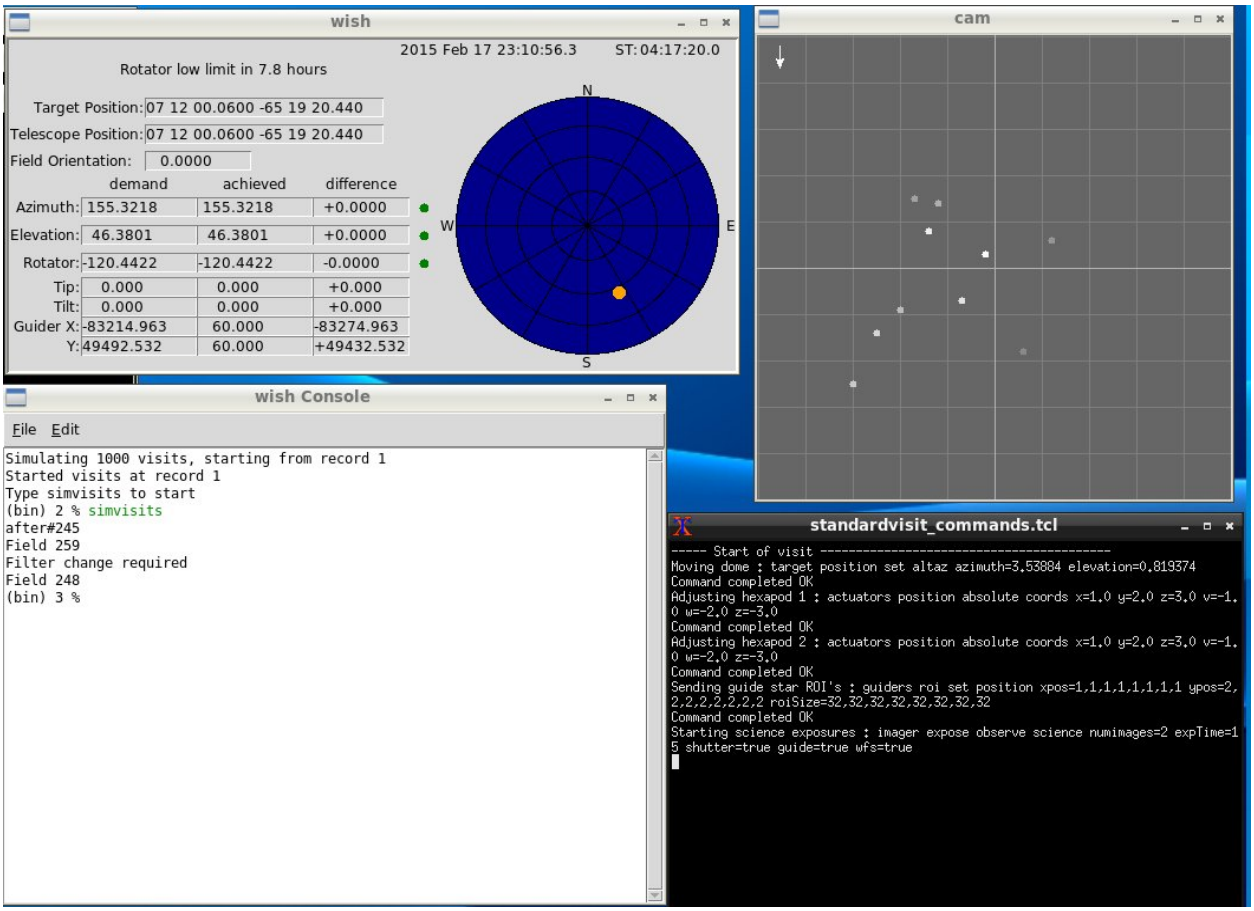
Once all the windows have deployed, the tcs simulator will automatically slew to the default target. Once it arrives (watch the GUI to follow it's progress), locate the command line interface window and type

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

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 <subssytem>;      // 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 publishber
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,                                - acknowledge a command
                long error, char *result );
```

```
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::* )( ::subssytem_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", &usbsystem_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 >( "subssytem_logeventC" )
      .def_readwrite( "message", &subsystem_logeventC::message )
```

;