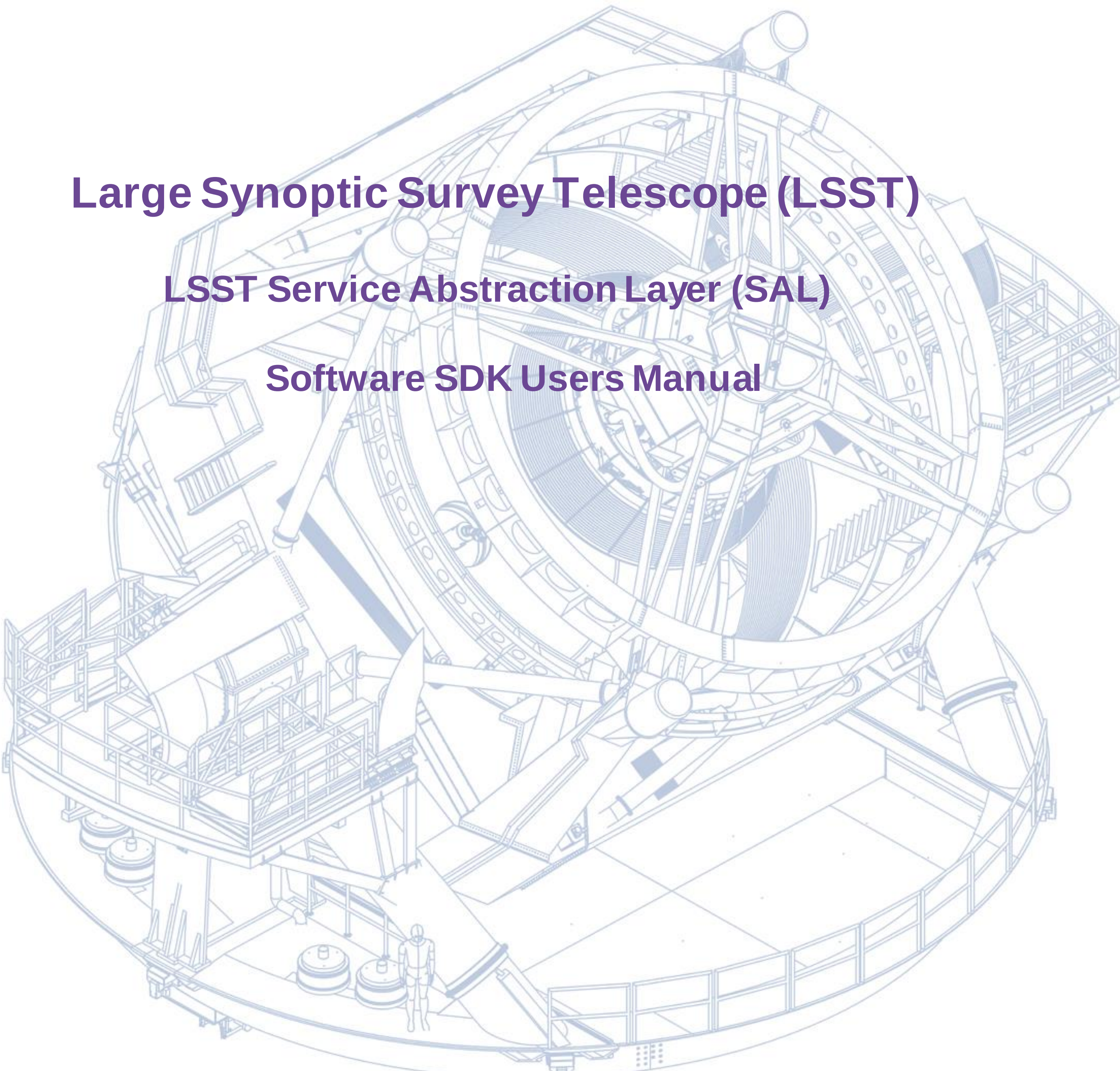
LARGE SYNOPTIC SURVEY TELESCOPE

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
V3.0	04/29/2016	Revisions for SAL V3 release	D. Mills
V3.5	02/23/2016	Revisions for SAL V3.5 release	D.Mills

Table of 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 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 7400:7413 -j ACCEPT
-A OUTPUT -p udp -m udp --dport 250:251 -j ACCEPT
-A OUTPUT -p udp -m udp --dport 7400:7413 -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=7400-7413/udp --permanent
firewall-cmd --reload

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

The location of the OpenSplice configuration file is stored in the environment variable OSPL_URI, and an extensive configuration tool exists (*osplconf)*, should customization be necessary.

2.1 Installation from a tar archive release

The tar archive format release includes a compatible version of OpenSplice as well as the SAL toolkit.

Unpack the SAL tar archive in a location of choice (/opt is recommended), e.g. in a terminal, replacing x.y.z with the appropriate version id

```
cd /opt
tar xzf [location-of-sdk-archive]/salSDK-x.y.z_x86_64.tgz
```

and then add the SDK setup command.

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

to your bash login profile.

2.2 Installation from Git repositories

Use a git client of your preference to check out the required branch of the following repositories

https://github.com/lsst-ts/ts_sal
https://github.com/lsst-ts/ts_opensplice

and then add the SDK setup command.

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

to your bash login profile.

2.3 Install location customization

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 other important environment variable is SAL_WORK_DIR. This is the directory in which you will run the SAL tools, and in which all the output files and libraries will be generated. By default this will be the "test" subdirectory in LSST_SDK_INSTALL, but you can change SAL_WORK_DIR to redefine it if required.

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 $SAL_WORK_DIR
cp $SAL_HOME/scripts/xml-templates/dome/*.xml   .
```

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

Example makefiles are provided for all the test programs. The list of libraries required to link with the middleware can be found in section 8.0

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

In all XML data definition files the IDL_Type keyword is used to specify the datatype of each field. The following datatypes are supported:

- short
- long (this is a 4 byte integer, and is represented as int on Linux 64-bit)
- long long
- unsigned short
- unsigned long (this is a 4 byte integer, and is represented as int on Linux 64-bit)
- unsigned long long
- float
- double
- char
- boolean
- octet (sequence of unsigned bytes)
- string
- string <maxlength>

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</Explan
ation>
   <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>
</SALCommand>
```

Note : The generic lifecycle commands should NOT be included , they are automatically generated during the salgenerator validation process. The current generic command set is {***start, stop, enable, disable, abort, enterControl, exit***}

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>
```

3.4 Updating the XML definitions

The XML definitions of the SAL objects for each subsystem are maintained in a github repository (https://github.com/lsst-ts/ts_xml).

When subsystem developers update the XML definitions for their interfaces, they should create a new feature branch in the github repository and put the modified version into it. Once the feature(s) have been fully tested, the corresponding changes are made made to the appropriate ICD. Once the ICD has been approved by the Change Control Board, the modified XML will be merged into the master branch and assigned an official release number. The master (release) branch is used to generate the SAL runtime libraries which can be used by other subsystems for integration testing. The master branch is also used by the Continuous Integration Unit Testing framework.

The XML definition files for the subsystem you are developing should be checked out of the github repository to ensure you are working with the latest version.

For convenience the full set of current definition files in also included in each SAL SDK Release (in lsstsal/scripts/xml-templates).

The XML definition files should be copied to the SAL_WORK_DIR directory before using the SAL tools.

The SAL tools must be run from the SAL_WORK_DIR directory.

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 https://github.com/lsst-ts/ts_xml.

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	- *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 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 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
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.

The procedure for generating test VI's for the LabVIEW interface is detailed in Appendix X. At present this is an interactive process, involving lots of LabVIEW dialogs.

5. Testing

5.1 Environment

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

idlpp

should produce

```
Usage: idlpp [-c preprocessor-path] [-b ORB-template-path]
        [-n <include-suffix] [-I path] [-D macro[=definition]] [-S] [-C]
        [-l (c | c++ \ cpp \ isocpp \ cs \ java)] [-j [old]:<new>] [-d directory] [-i]
        [-P dll_macro_name[,<h-file>]] [-o (dds-types | custom-psm | no-equality)] <filename>
```

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

salgenerator

should produce

```
SAL generator tool - Usage :

       salgenerator subsystem flag(s)

   where flag(s) may be

              validate - check the XML Telemetry/Command/LogEvent definitions
              sal      - generate SAL wrappers for cpp, java, isocpp, python
              lib      - generate shared library
              tcl      - generate tcl interface
              html     - generate web form interfaces
              labview  - generate LabVIEW low-level interface
              maven    - generate a maven repository
              db       - generate telemetry database table

                   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

              sim      - generate simulation configuration
              icd      - generate ICD document
              link     - link a SAL program
              verbose  - be more verbose ;-)
```

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

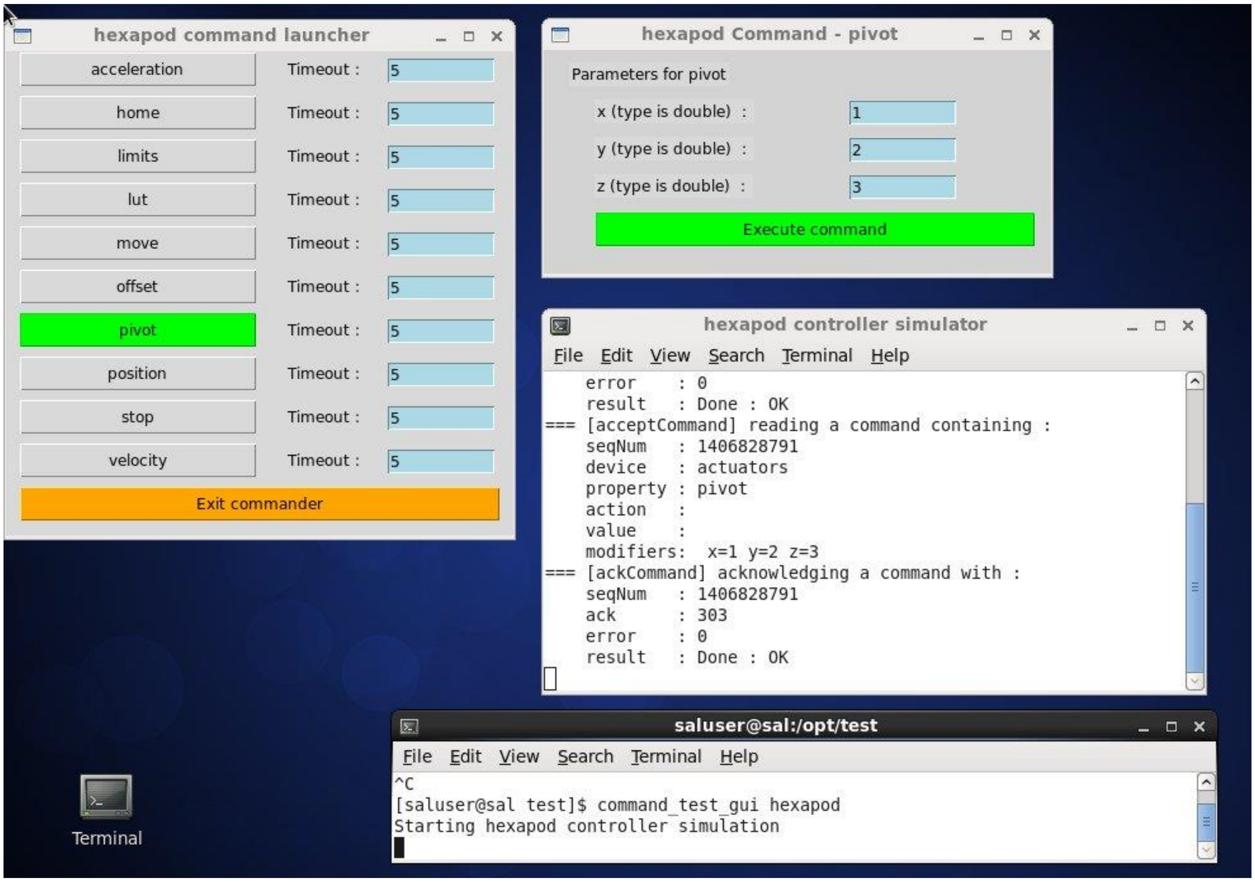
mount/java/src/mount_cmdctl.run - starts command processor

In addition a gui can be used to send all supported subsystem commands (with 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.


hexapod command launcher
acceleration
Timeout : 5
home
Timeout : 5
limits
Timeout : 5
lut
Timeout : 5
move
Timeout : 5
offset
Timeout : 5
pivot
Timeout : 5
position
Timeout : 5
stop
Timeout : 5
velocity
Timeout : 5
Exit commander
hexapod Command - pivot
Parameters for pivot
x (type is double) : 1
y (type is double) : 2
z (type is double) : 3
Execute command
hexapod controller simulator
File Edit View Search Terminal Help
error : 0
result : Done : OK
=== [acceptCommand] reading a command containing :
seqNum : 1406828791
device : actuators
property : pivot
action :
value :
modifiers: x=1 y=2 z=3
=== [ackCommand] acknowledging a command with :
seqNum : 1406828791
ack : 303
error : 0
result : Done : OK
saluser@sal:/opt/test
File Edit View Search Terminal Help
^C
[saluser@sal test]$ command_test_gui hexapod
Starting hexapod controller simulation
Terminal

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

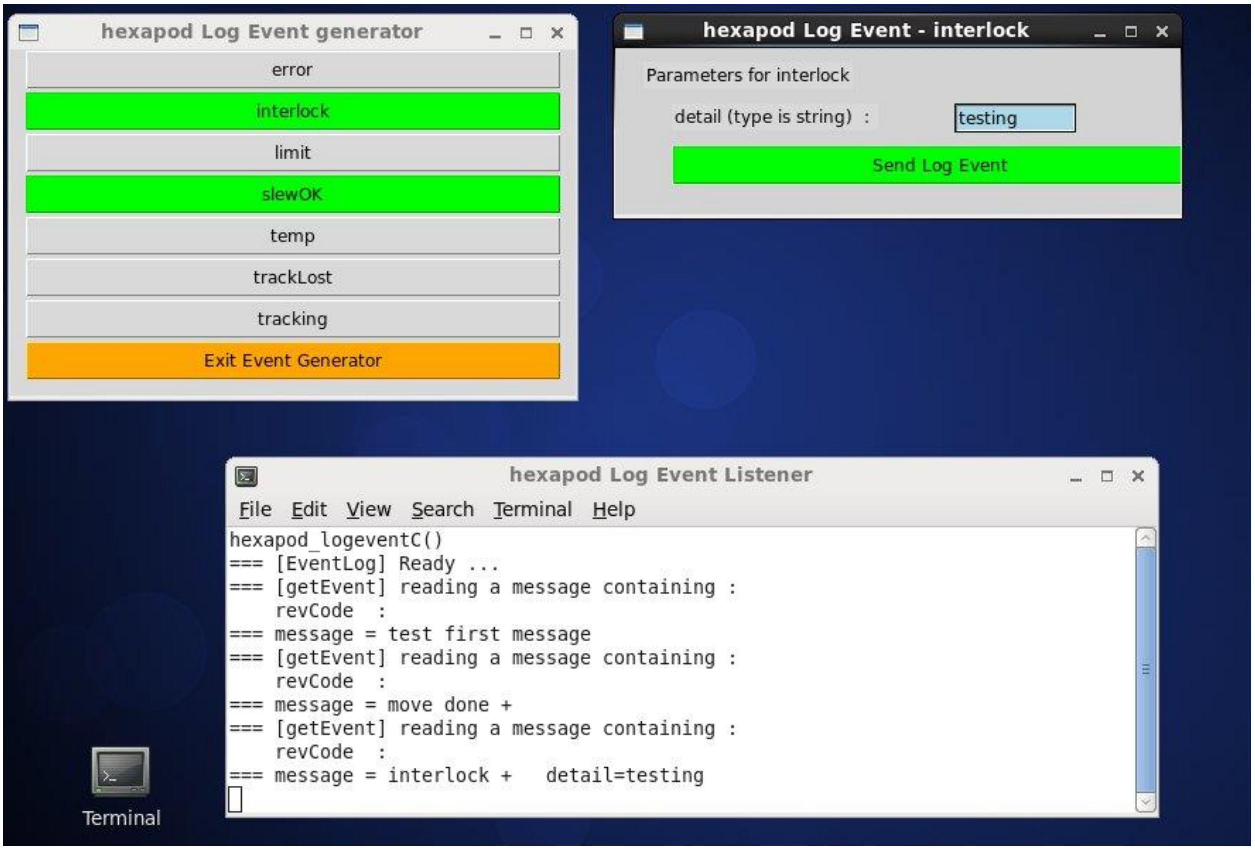
For java

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

$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

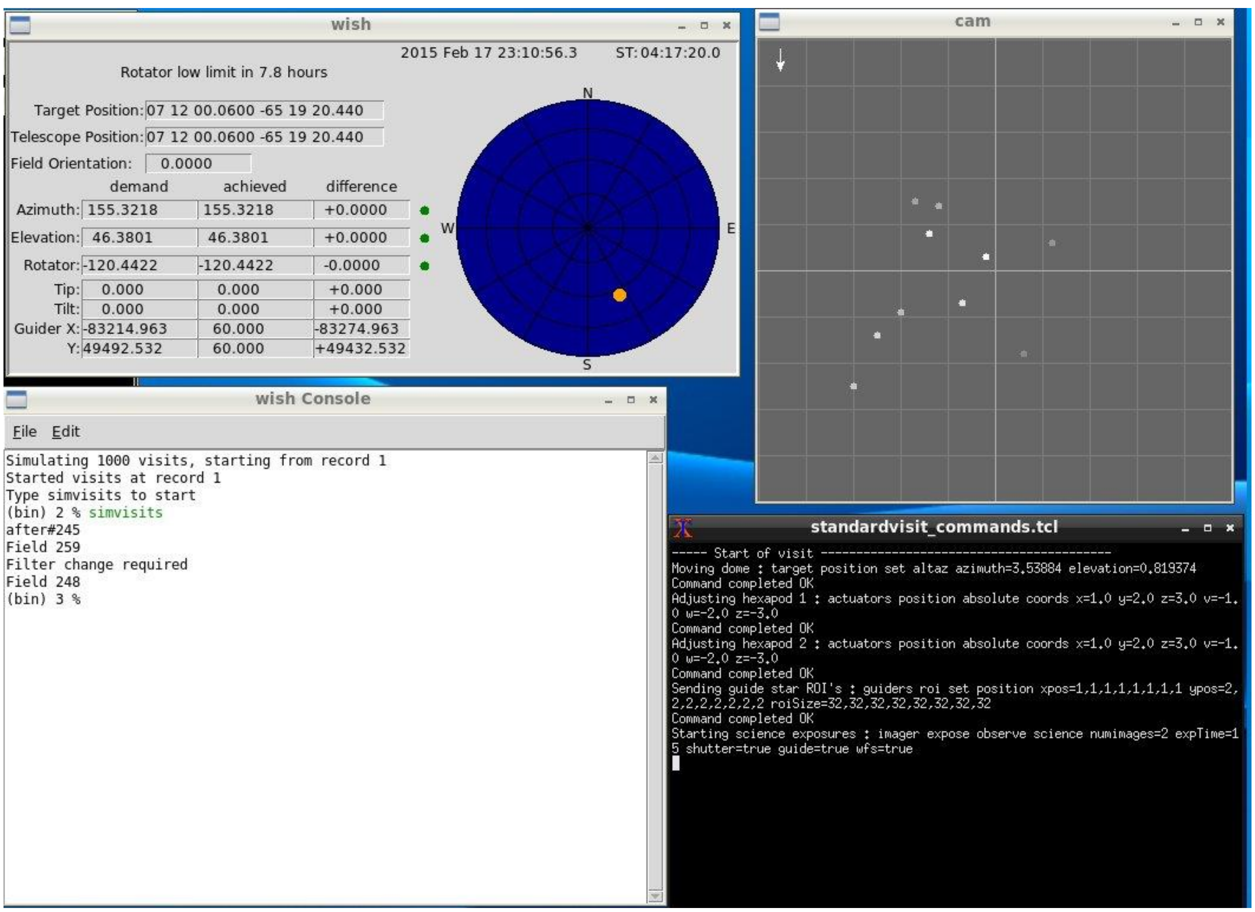
Once all the windows have deployed, the tcs simulator will automatically slew to the default target. Once it arrives (watch the GUI to follow it's progress), locate the command line interface window and type

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 <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 publisher object
void salTelemetrySub(char *topicName);                               - create telemetry subscriber object
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 subsystemname here
import org.lsst.sal.<SAL_subsystem>;  //substitute actual subsystemname 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")  ) )
```

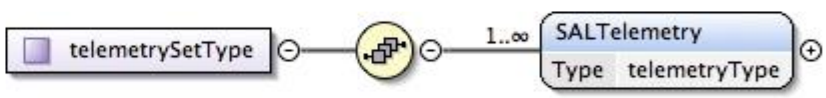
```
  .def(
        "setDebugLevel"
        , (::int  ( ::SAL_subsystem::* )( int ) )( &::SAL_subsystem::setDebugLevel )
        , ( bp::arg("level") ) )
  .def(
        "setProperty"
        , (::int  ( ::SAL_subsystem::* )( char *,char * ) )( &::SAL_subsystem::setProperty )
        , ( bp::arg("property"), 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 )
    ;
```

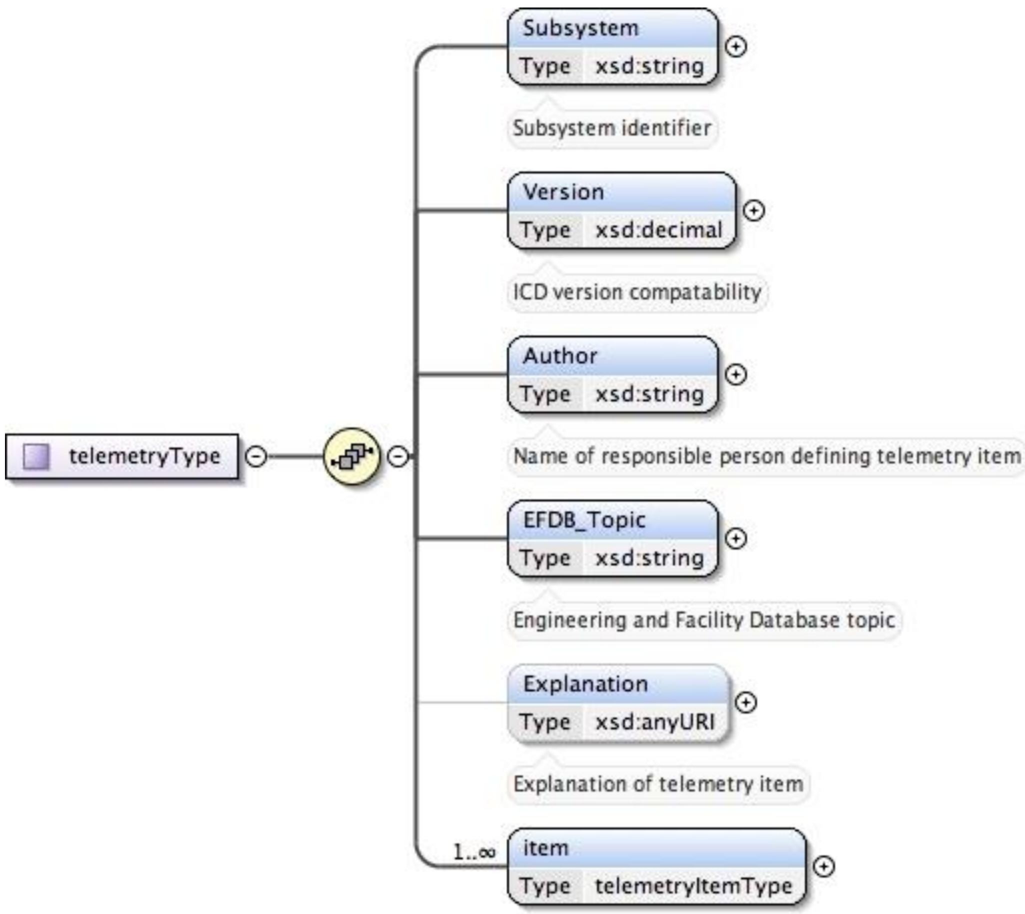
7.0 SAL XML Schema

7.1 Telemetry

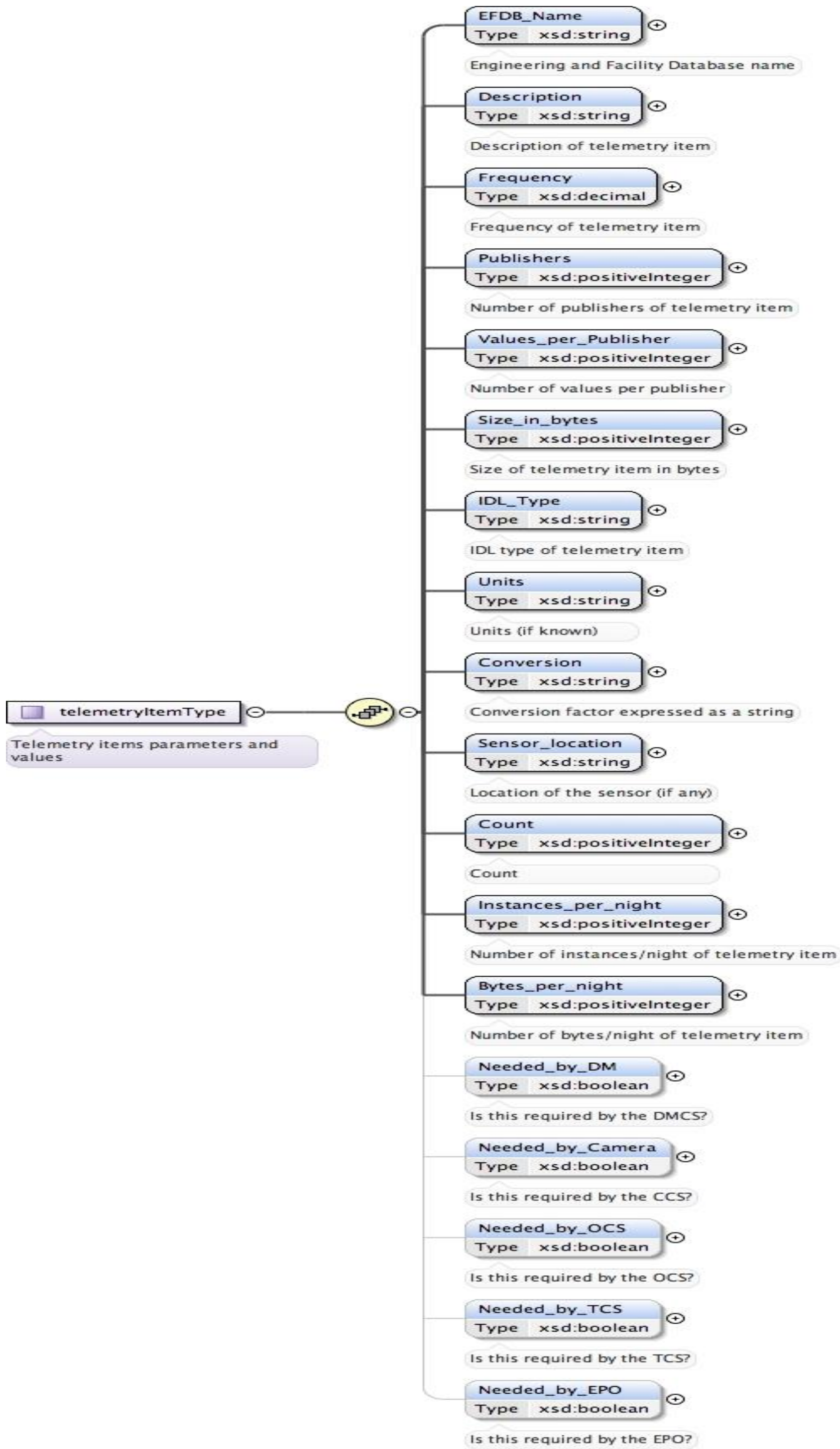
7.1.1 telemetrySetType



7.1.2 telemetryType

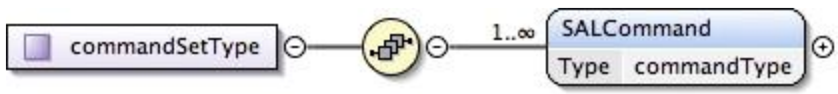


7.1.3 telemetryItemType

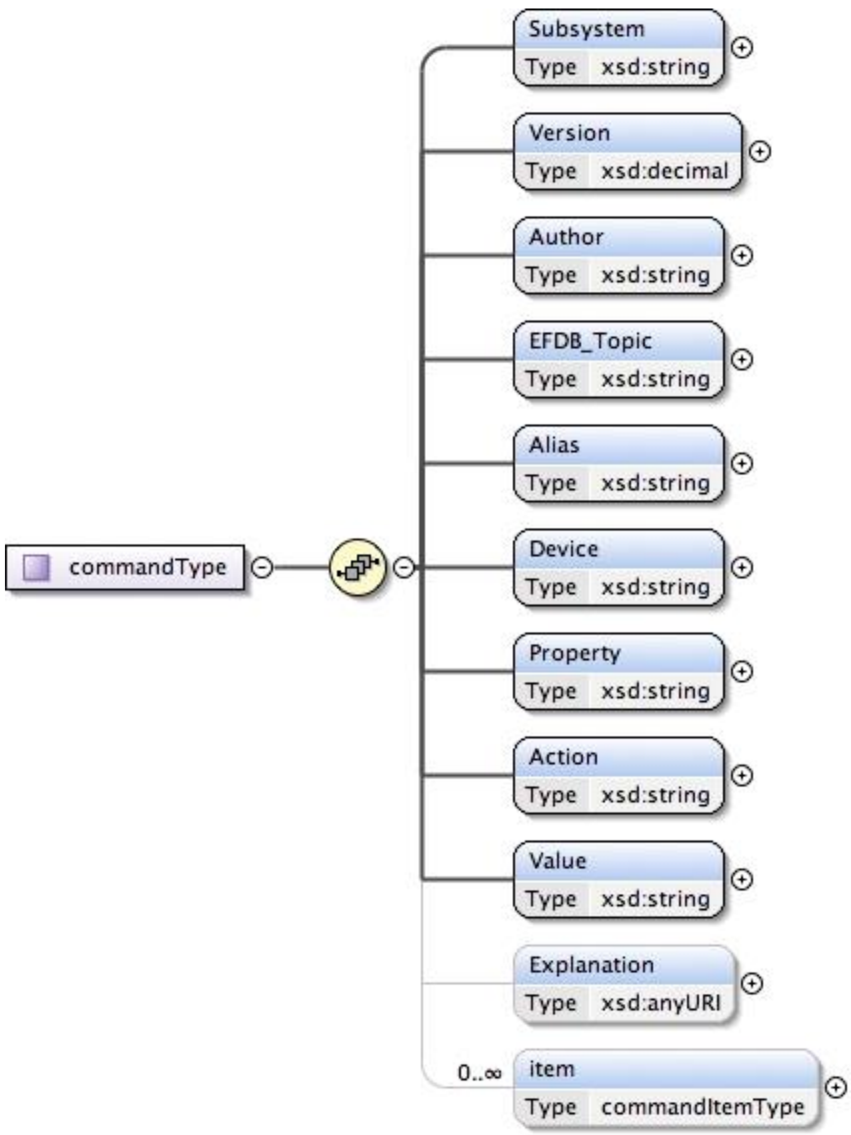


7.2 Commands

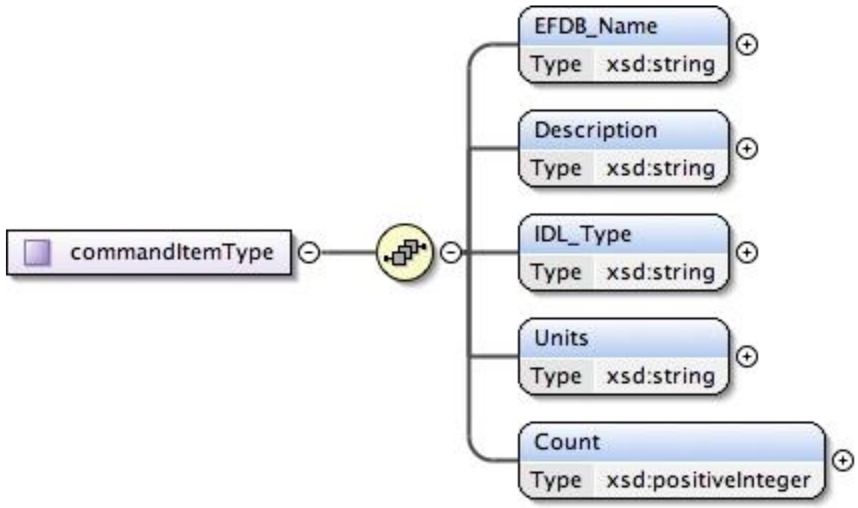
7.2.1 commandSetType



7.2.2 commandType

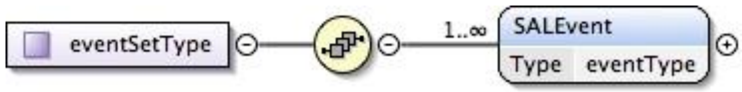


7.2.3 commandItemType

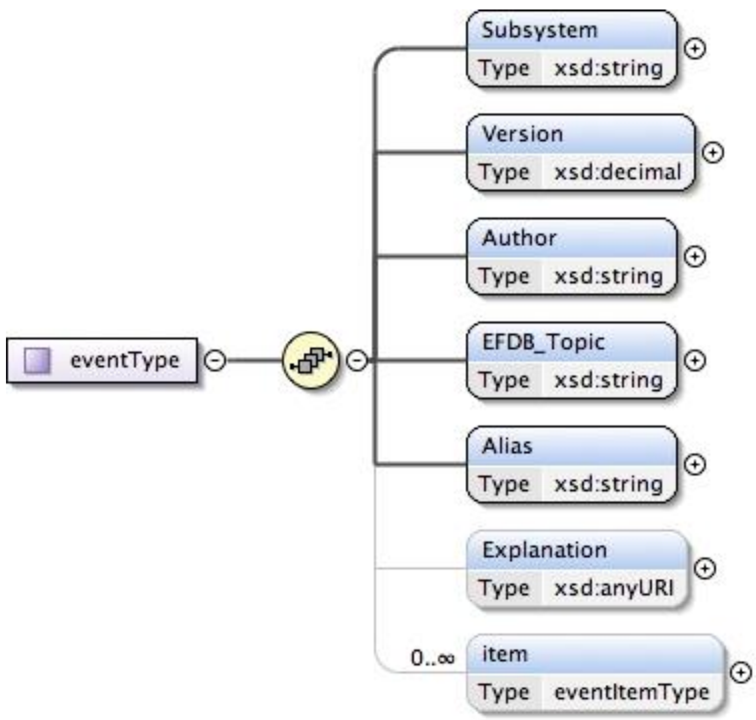


7.3 Events

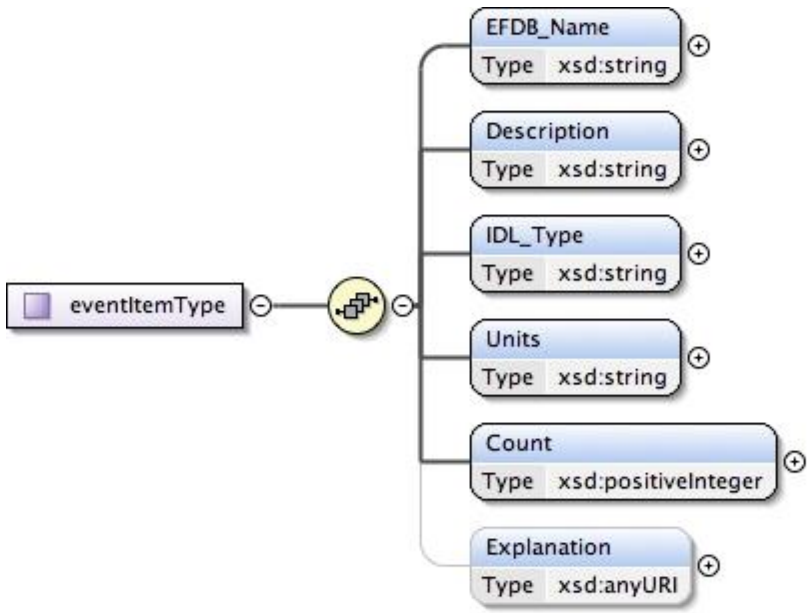
7.3.1 eventSetType



7.3.2 eventType



7.3.3 eventItemType



8.0 Compiler Options and Link Libraries

The following compiler options are required when compiling application code For subsequent linking with the SAL and DDS middleware.

CFLAGS/CXXFLAGS : -m64 -D_REENTRANT –fPIC –Wno-write-strings

Subsystems with duplicate instantiations (e.g. Hexapods) also require

-DSAL_SUBSYSTEM_IS_KEYED

and the following include paths will be required

```
-I$(OSPL_HOME)/include
-I$(OSPL_HOME)/include/sys
-I$(OSPL_HOME)/include/dcps/C++/SACPP
-I$(SAL_HOME)/include
-I$(SAL_WORK_DIR)/include
-I../../-subsys-/cppsrc
```

Where -subsys- is the subsystem name e.g. hexapod

The following libraries are required when linking an application to use the SAL and DDS middleware. For an application that communicates with multiple subsystems, the SAL libraries for each must be included.

SAL : libSAL_[subsystem-name].so , libsacpp_[subsystem-name]_types.so

DDS : libdcpssacpp.so , libdcpsgapi.so , libddsuser.so , libddskernel.so , libddsserialization.so , libddsconfparser.so , libddsdatabase.so , libddsutil.so, libddsos.so, libddsconf.so

Other : libdl.so , libpthread.so

Appropriate linker path directives are

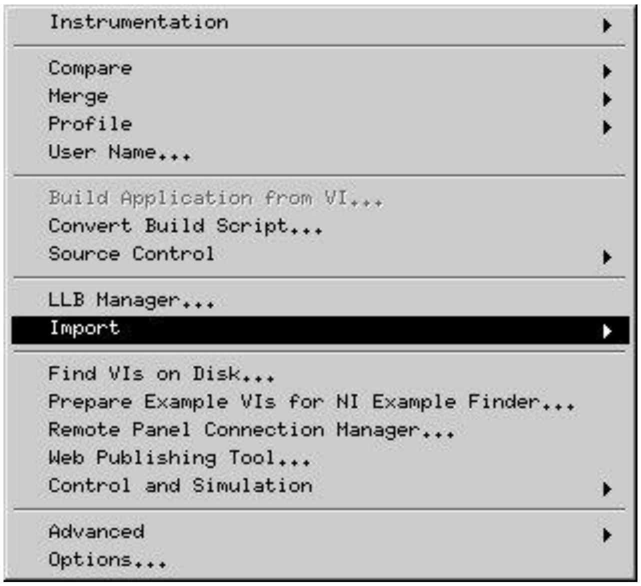
```
-L$(OSPL_HOME)/lib   -L$(SAL_HOME)/lib
```

9.0 LabVIEW test VI generation

The generation of the LabVIEW test VI's is an interactive process. The LabVIEW Shared library import is used to automatically generate VI's to interact with the Salgenerator produced SALLV_[subsystem].so library.

> **NOTE : It is vital to COMPLETELY DELETE the entire destination directory and it's Contents so that wizard can create it's output directory afresh. For example if you choose To place the results *in /home/me/sal/test/tcs/labview/lib*, then you should run the following command BEFORE starting the LabVIEW tools.**
>
> ***rm –fr /home/me/sal/test/tcs/labview/lib***

1. Start LabVIEW and select the Tools->Import->Shared Library (.so) option

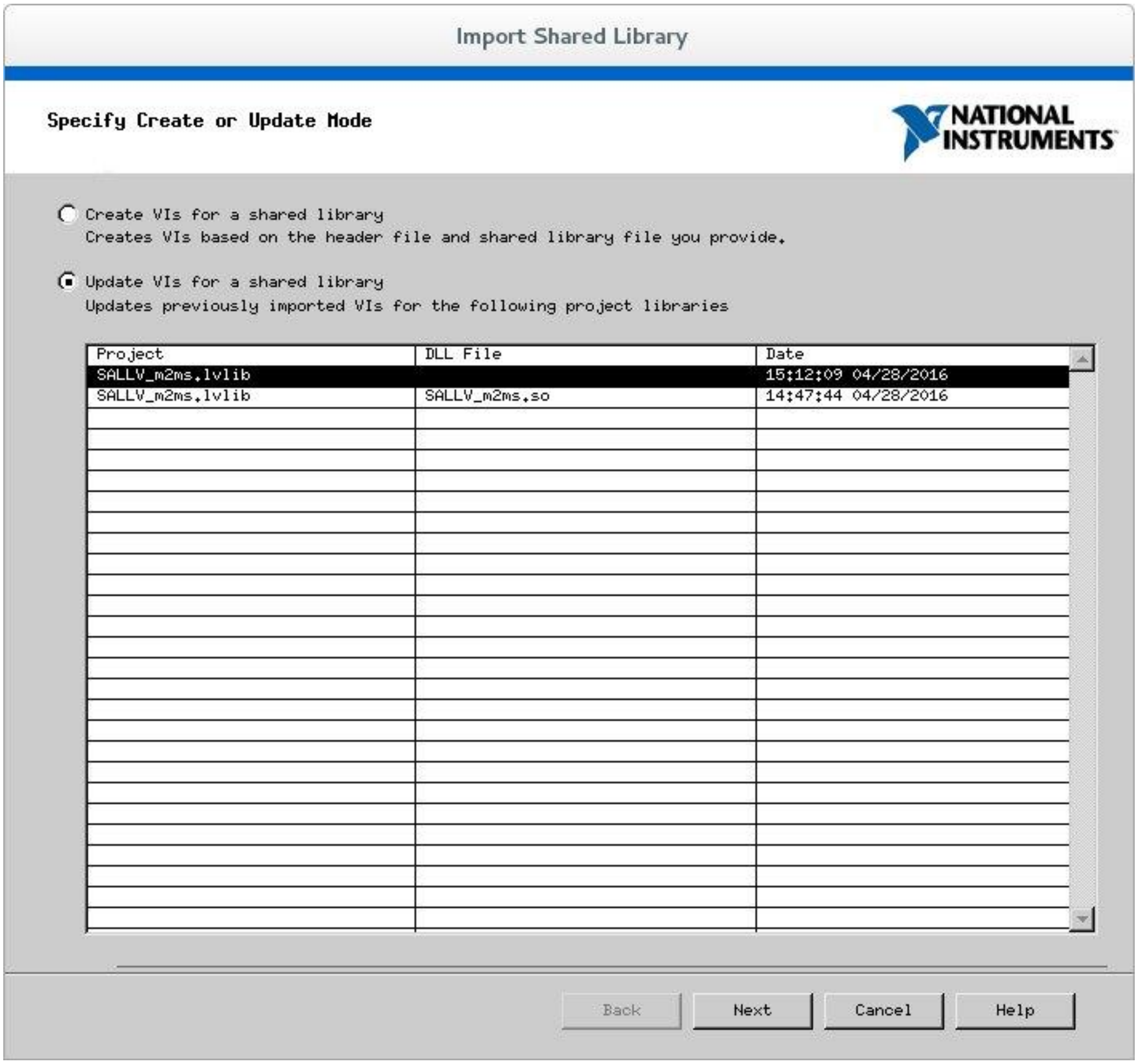


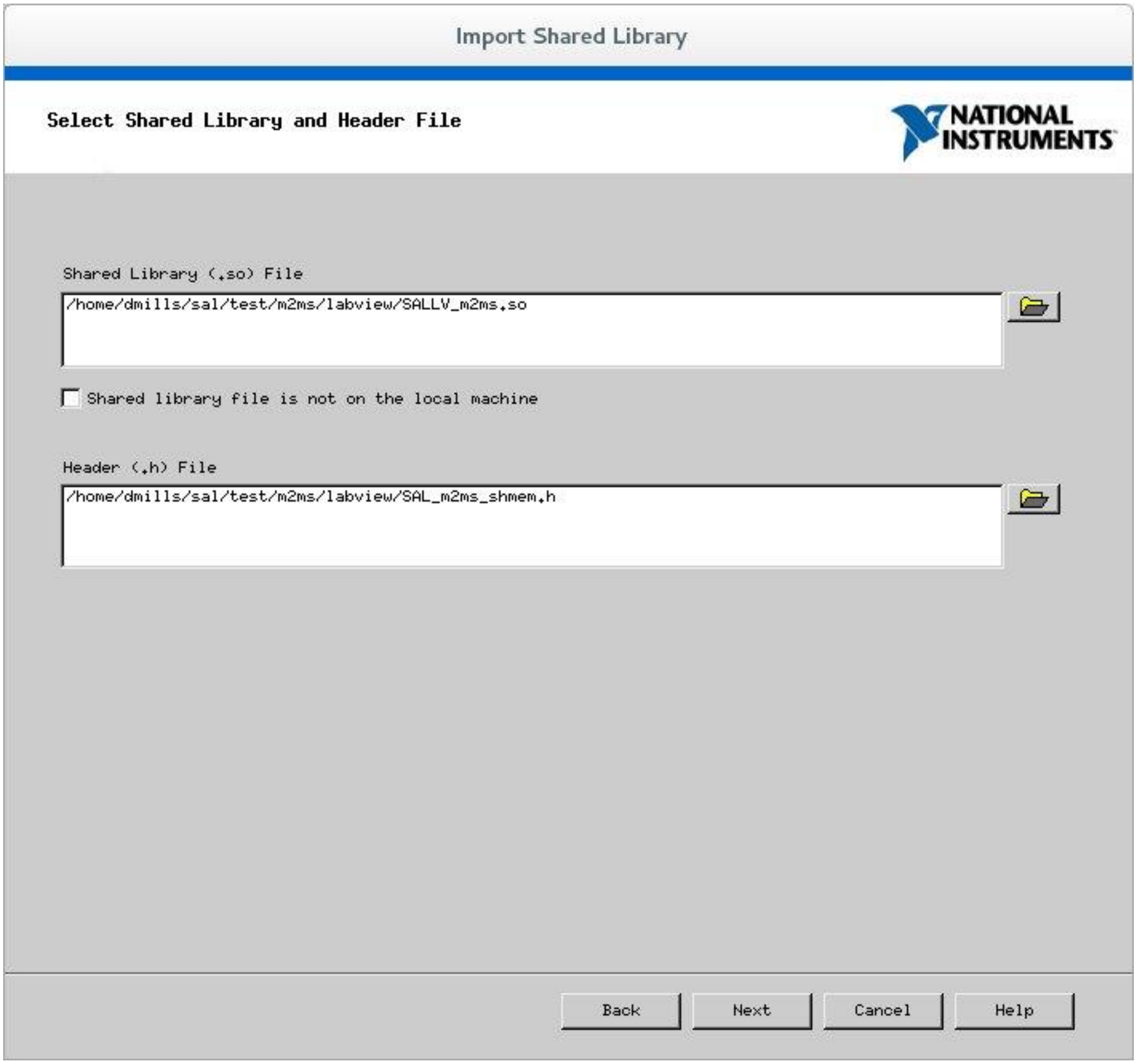
2. Choose either New or Update option and specify the path to the library and then click Next. Proceed through the rest of the dialogs as illustrated below. Generally selecting the default and clicking Next is appropriate.

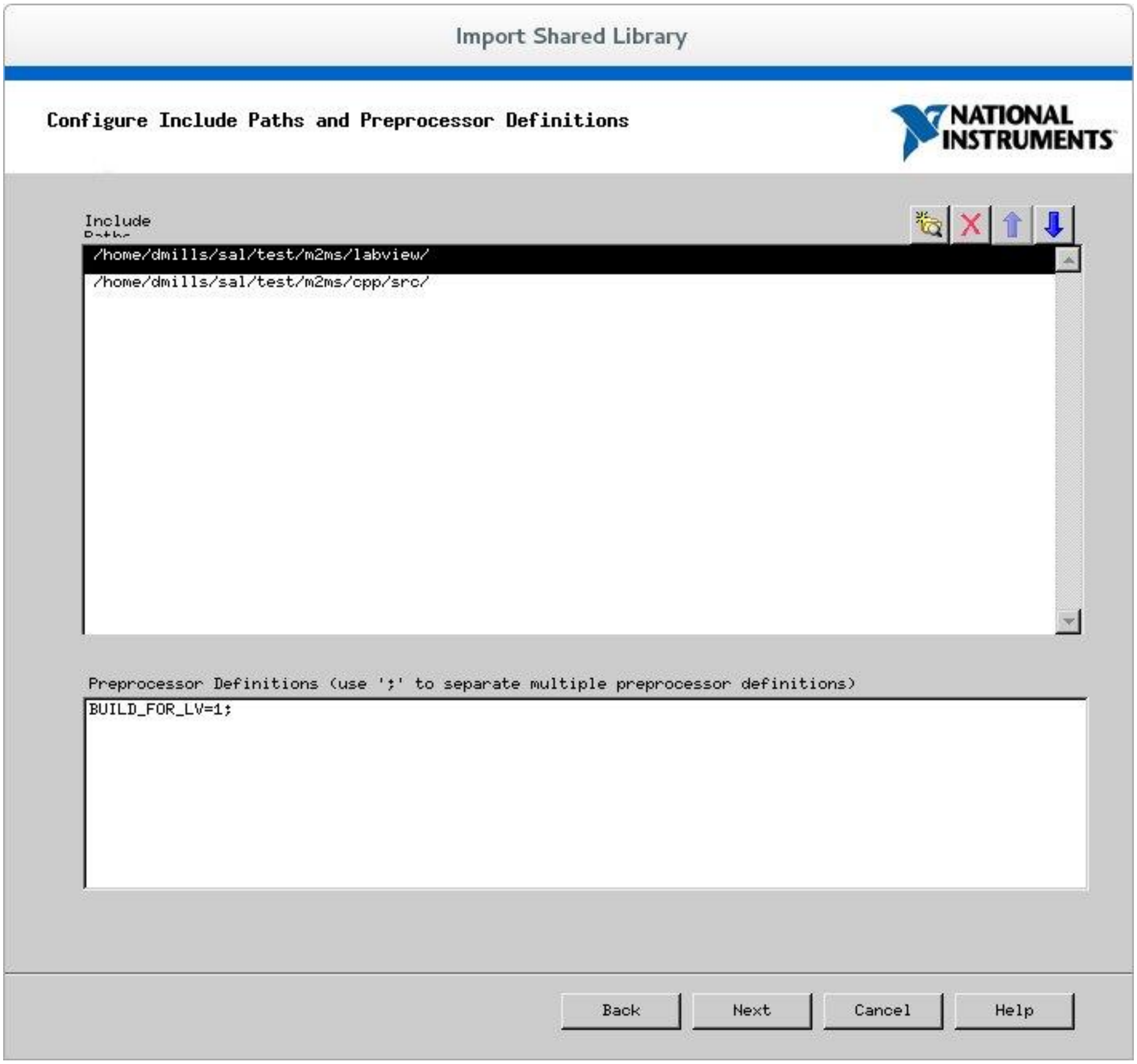
   The only non-standard option is in the “Configure Include Paths…” dialog where you must enter the

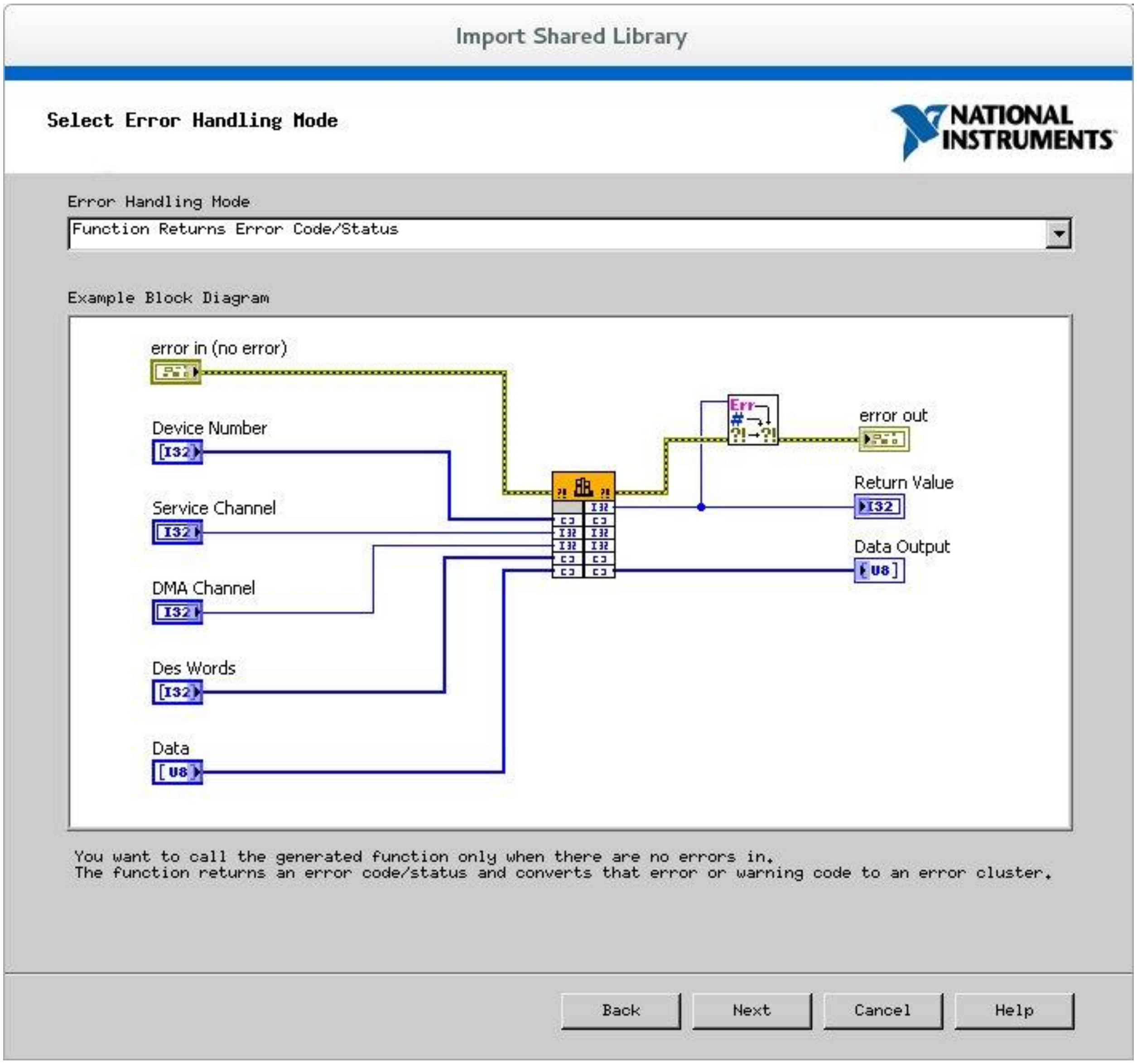
   BUILD_FOR_LV=1

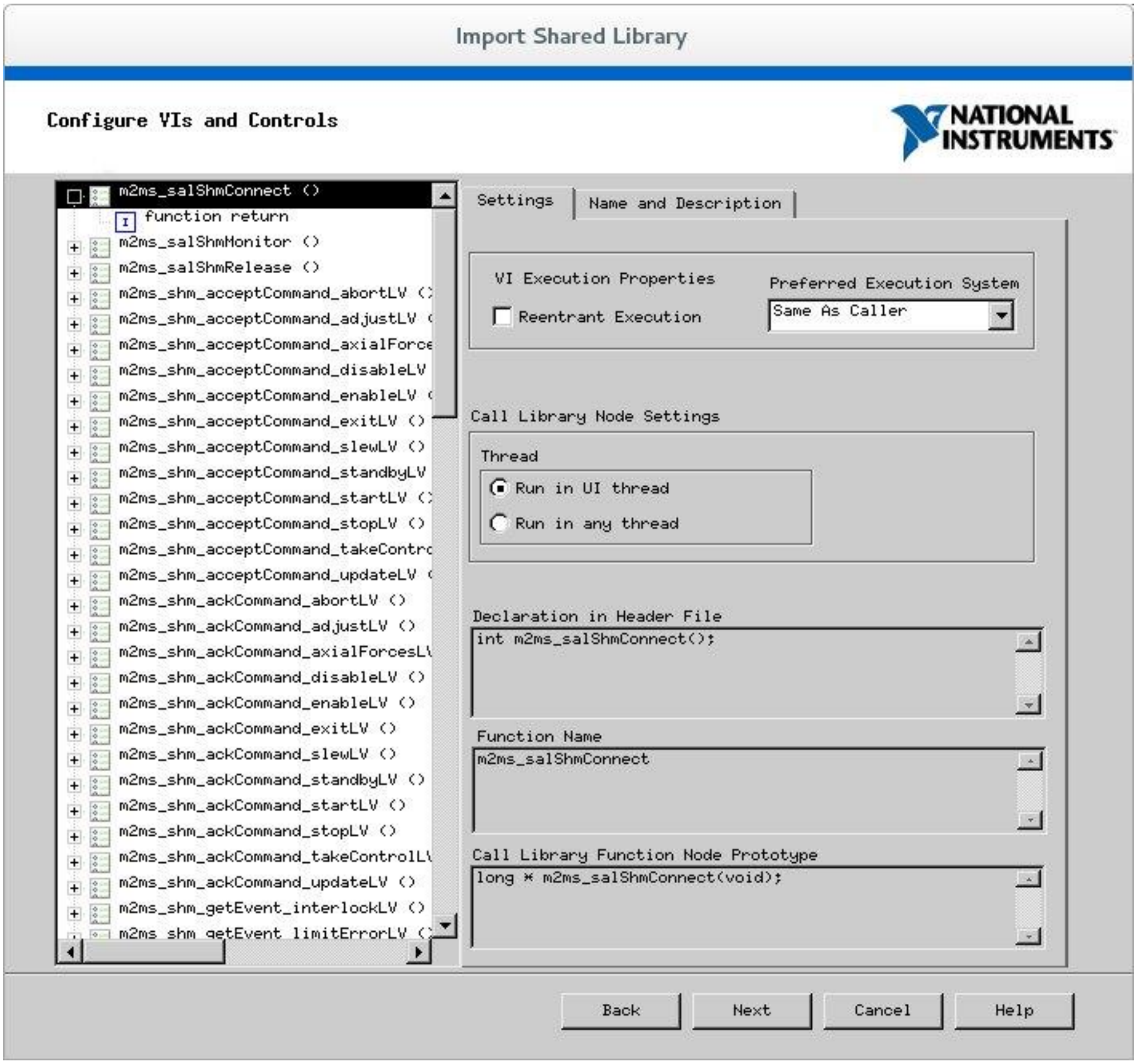
   Option in the Preprocessor options section.

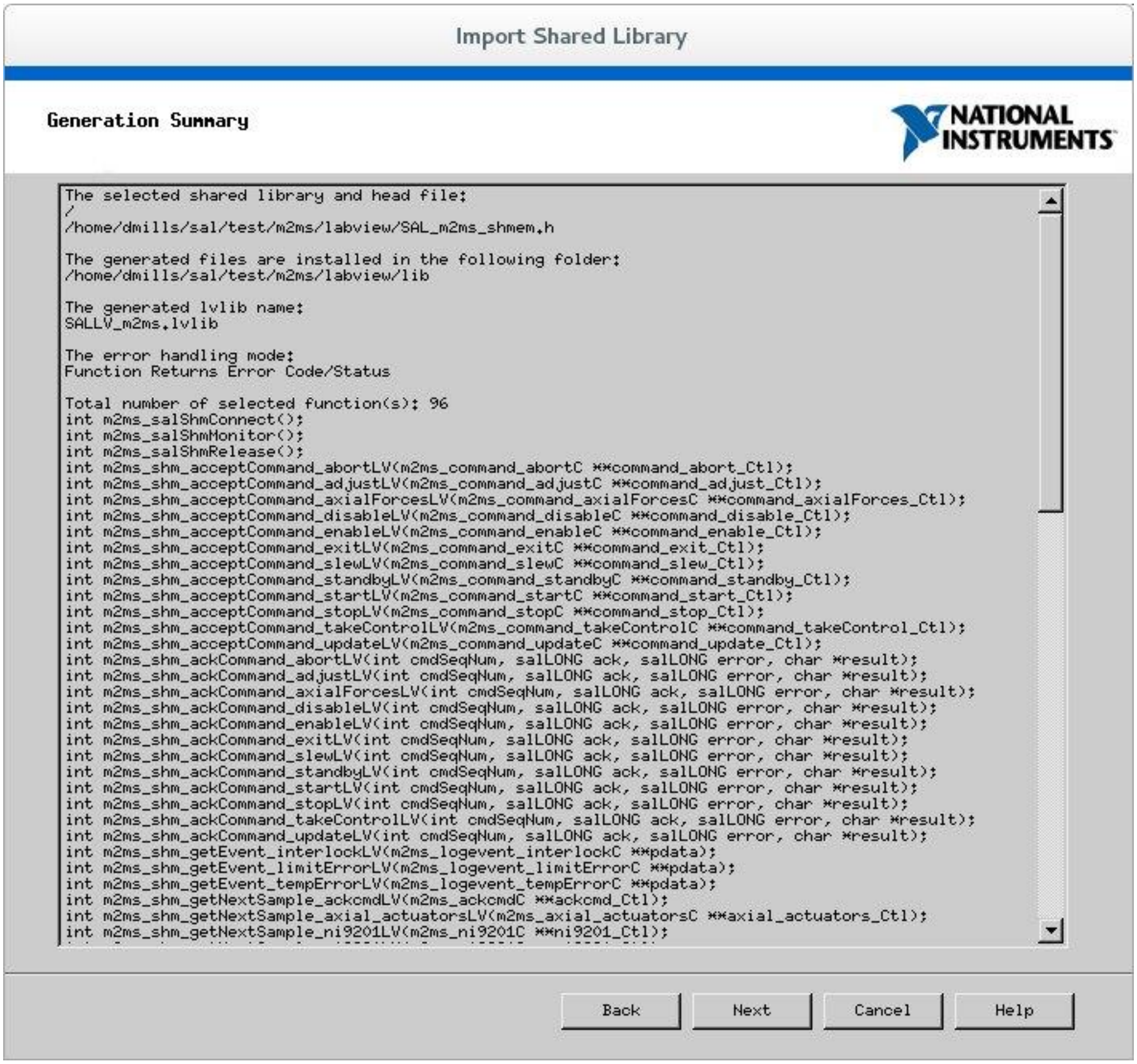

Import Shared Library
Specify Create or Update Mode
NATIONAL INSTRUMENTS
Create VIs for a shared library
Creates VIs based on the header file and shared library file you provide.
Update VIs for a shared library
Updates previously imported VIs for the following project libraries
Project
DLL File
Date
SALLV_m2ms.lvlib
15:12:09 04/28/2016
SALLV_m2ms.lvlib
SALLV_m2ms.so
14:47:44 04/28/2016
Back
Next
Cancel
Help

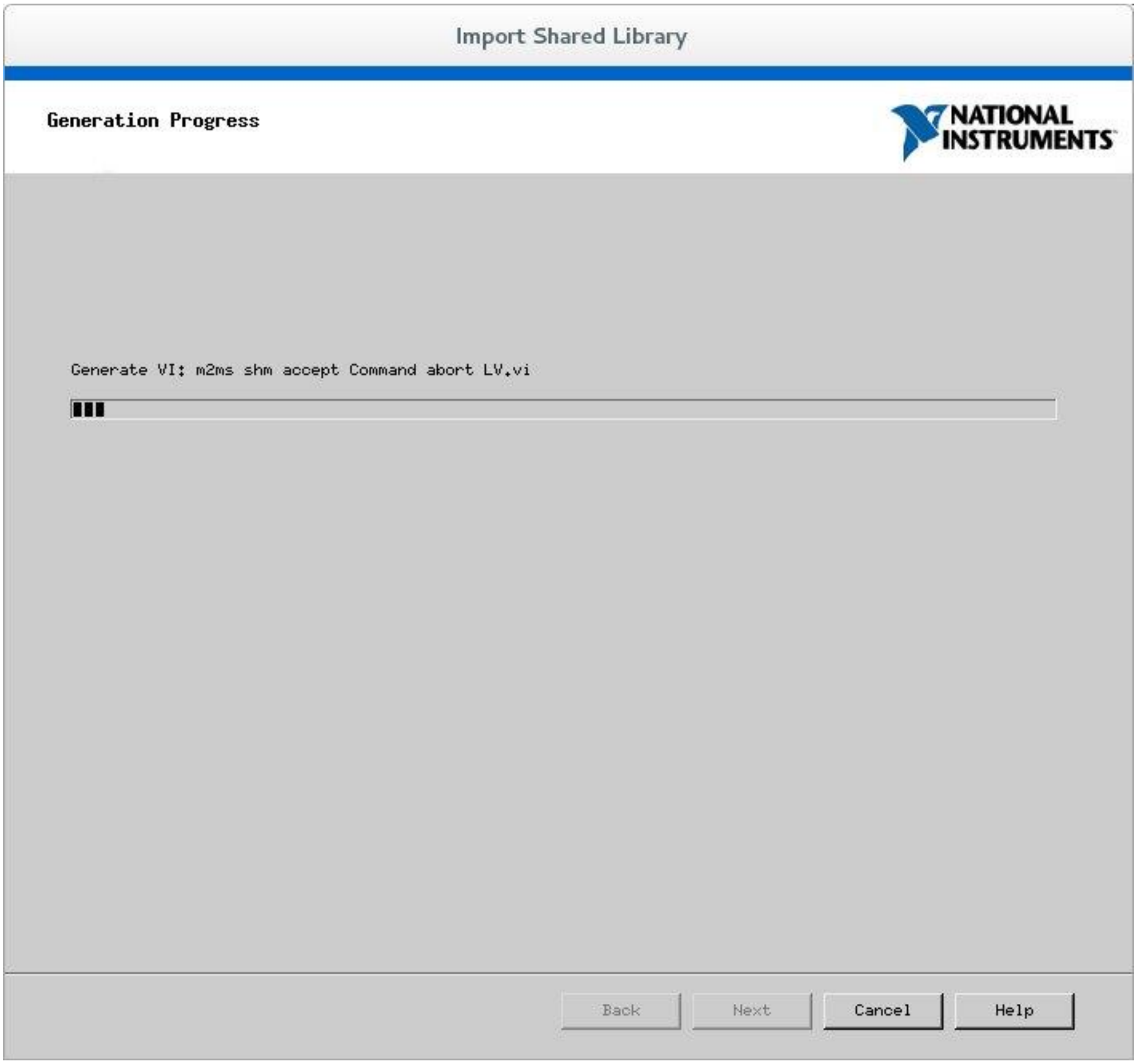

Import Shared Library
Select Shared Library and Header File
NATIONAL INSTRUMENTS
Shared Library (.so) File
/home/dmills/sal/test/m2ms/labview/SALLV_m2ms.so
Shared library file is not on the local machine
Header (.h) File
/home/dmills/sal/test/m2ms/labview/SAL_m2ms_shmem.h
Back
Next
Cancel
Help


Import Shared Library
Configure Include Paths and Preprocessor Definitions
Include Paths
/home/dmills/sal/test/m2ms/labview/
/home/dmills/sal/test/m2ms/cpp/src/
Preprocessor Definitions (use ';' to separate multiple preprocessor definitions)
BUILD_FOR_LV=1;
Back
Next
Cancel
Help

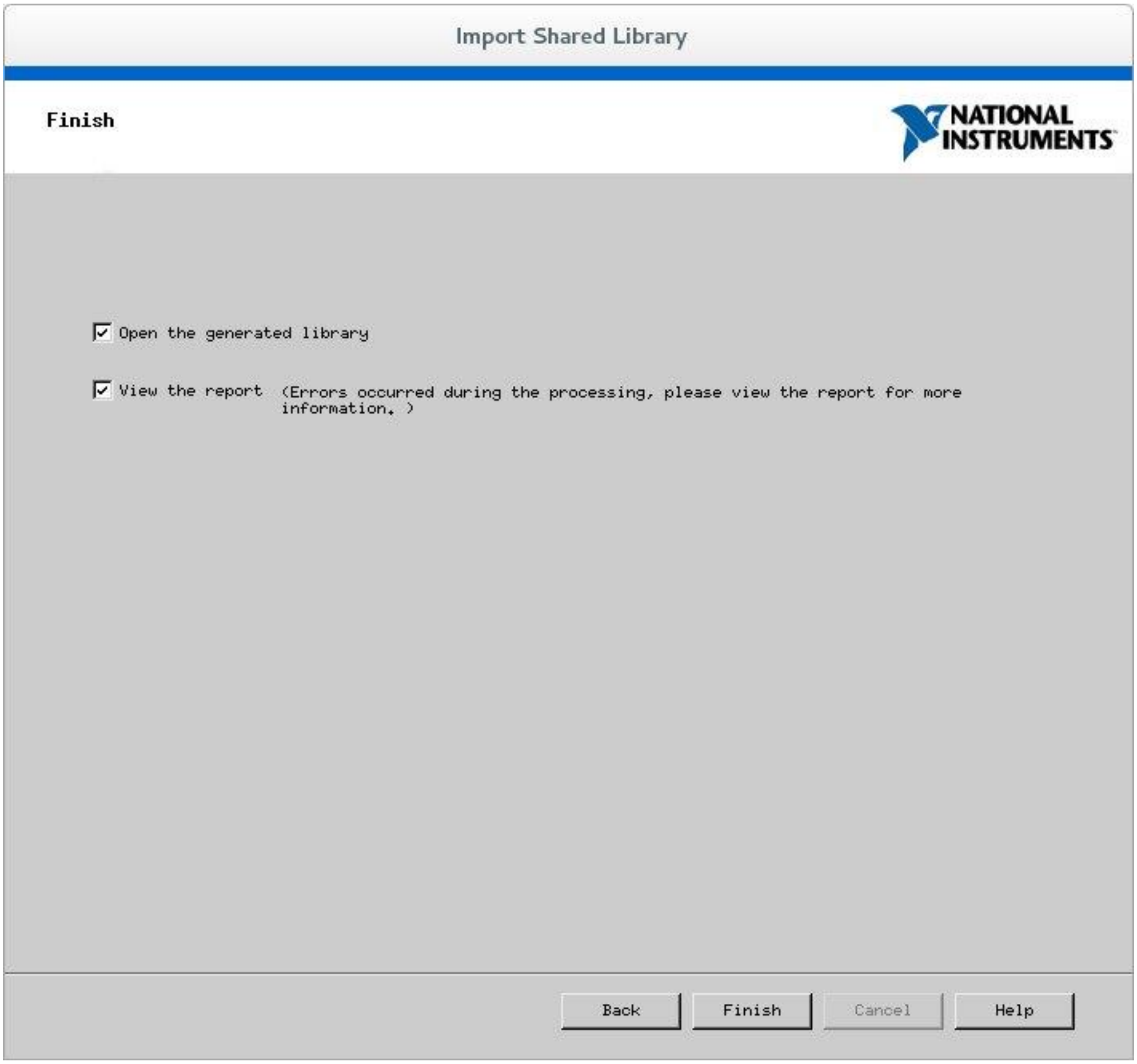

Import Shared Library
Select Error Handling Mode
NATIONAL INSTRUMENTS
Error Handling Mode
Function Returns Error Code/Status
Example Block Diagram
error in (no error)
Device Number
Service Channel
DMA Channel
Des Words
Data
error out
Return Value
Data Output
You want to call the generated function only when there are no errors in.
The function returns an error code/status and converts that error or warning code to an error cluster.
Back
Next
Cancel
Help

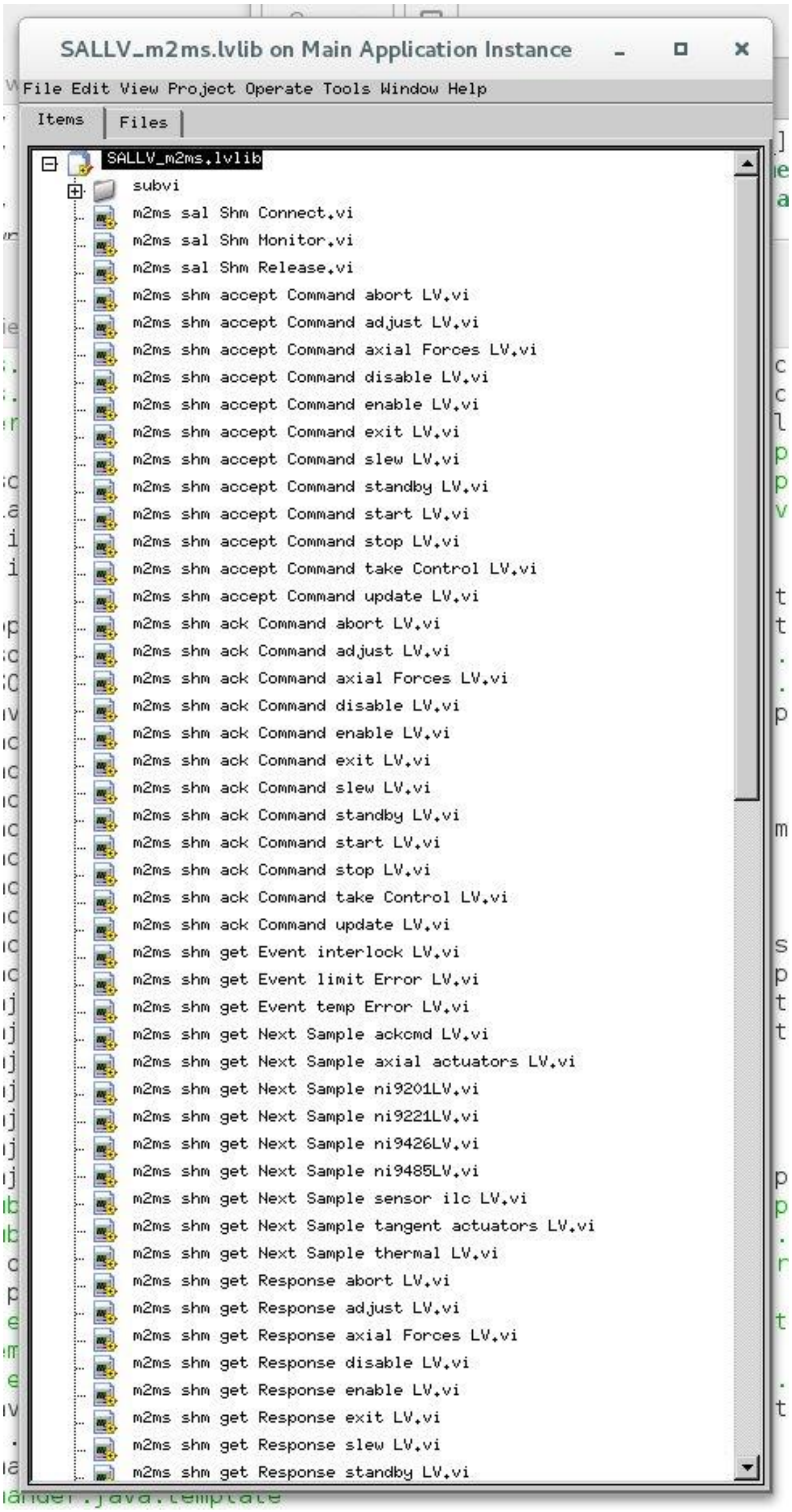

Import Shared Library
Configure VIs and Controls
NATIONAL INSTRUMENTS
m2ms_salShmConnect ()
function return
m2ms_salShmMonitor ()
m2ms_salShmRelease ()
m2ms_shm_acceptCommand_abortLV
m2ms_shm_acceptCommand_adjustLV
m2ms_shm_acceptCommand_axialForce
m2ms_shm_acceptCommand_disableLV
m2ms_shm_acceptCommand_enableLV
m2ms_shm_acceptCommand_exitLV ()
m2ms_shm_acceptCommand_slewLV ()
m2ms_shm_acceptCommand_standbyLV
m2ms_shm_acceptCommand_startLV
m2ms_shm_acceptCommand_stopLV ()
m2ms_shm_acceptCommand_takeContro
m2ms_shm_acceptCommand_updateLV
m2ms_shm_ackCommand_abortLV ()
m2ms_shm_ackCommand_adjustLV ()
m2ms_shm_ackCommand_axialForcesL
m2ms_shm_ackCommand_disableLV ()
m2ms_shm_ackCommand_enableLV ()
m2ms_shm_ackCommand_exitLV ()
m2ms_shm_ackCommand_slewLV ()
m2ms_shm_ackCommand_standbyLV ()
m2ms_shm_ackCommand_startLV ()
m2ms_shm_ackCommand_stopLV ()
m2ms_shm_ackCommand_takeControlL
m2ms_shm_ackCommand_updateLV ()
m2ms_shm_getEvent_interlockLV ()
m2ms_shm_getEvent_limitErrorLV
Settings
Name and Description
VI Execution Properties
Reentrant Execution
Preferred Execution System
Same As Caller
Call Library Node Settings
Thread
Run in UI thread
Run in any thread
Declaration in Header File
int m2ms_salShmConnect();
Function Name
m2ms_salShmConnect
Call Library Function Node Prototype
long * m2ms_salShmConnect(void);
Back
Next
Cancel
Help


Import Shared Library
Generation Summary
NATIONAL INSTRUMENTS
The selected shared library and head file:
/
/home/dmills/sal/test/m2ms/labview/SAL_m2ms_shmem.h
The generated files are installed in the following folder:
/home/dmills/sal/test/m2ms/labview/lib
The generated lvlib name:
SALLV_m2ms.lvlib
The error handling mode:
Function Returns Error Code/Status
Total number of selected function(s): 96
int m2ms_salShmConnect();
int m2ms_salShmMonitor();
int m2ms_salShmRelease();
int m2ms_shm_acceptCommand_abortLV(m2ms_command_abortC **command_abort_Ctl);
int m2ms_shm_acceptCommand_adjustLV(m2ms_command_adjustC **command_adjust_Ctl);
int m2ms_shm_acceptCommand_axialForcesLV(m2ms_command_axialForcesC **command_axialForces_Ctl);
int m2ms_shm_acceptCommand_disableLV(m2ms_command_disableC **command_disable_Ctl);
int m2ms_shm_acceptCommand_enableLV(m2ms_command_enableC **command_enable_Ctl);
int m2ms_shm_acceptCommand_exitLV(m2ms_command_exitC **command_exit_Ctl);
int m2ms_shm_acceptCommand_slewLV(m2ms_command_slewC **command_slew_Ctl);
int m2ms_shm_acceptCommand_standbyLV(m2ms_command_standbyC **command_standby_Ctl);
int m2ms_shm_acceptCommand_startLV(m2ms_command_startC **command_start_Ctl);
int m2ms_shm_acceptCommand_stopLV(m2ms_command_stopC **command_stop_Ctl);
int m2ms_shm_acceptCommand_takeControlLV(m2ms_command_takeControlC **command_takeControl_Ctl);
int m2ms_shm_acceptCommand_updateLV(m2ms_command_updateC **command_update_Ctl);
int m2ms_shm_ackCommand_abortLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_adjustLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_axialForcesLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_disableLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_enableLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_exitLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_slewLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_standbyLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_startLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_stopLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_takeControlLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_ackCommand_updateLV(int cmdSeqNum, salLONG ack, salLONG error, char *result);
int m2ms_shm_getEvent_interlockLV(m2ms_logevent_interlockC **pdata);
int m2ms_shm_getEvent_limitErrorLV(m2ms_logevent_limitErrorC **pdata);
int m2ms_shm_getEvent_tempErrorLV(m2ms_logevent_tempErrorC **pdata);
int m2ms_shm_getNextSample_ackcmdLV(m2ms_ackcmdC **ackcmd_Ctl);
int m2ms_shm_getNextSample_axial_actuatorsLV(m2ms_axial_actuatorsC **axial_actuators_Ctl);
int m2ms_shm_getNextSample_ni9201LV(m2ms_ni9201C **ni9201_Ctl);
Back
Next
Cancel
Help


Import Shared Library
Generation Progress
NATIONAL INSTRUMENTS
Generate VI: m2ms shm accept Command abort LV.vi
Back
Next
Cancel
Help

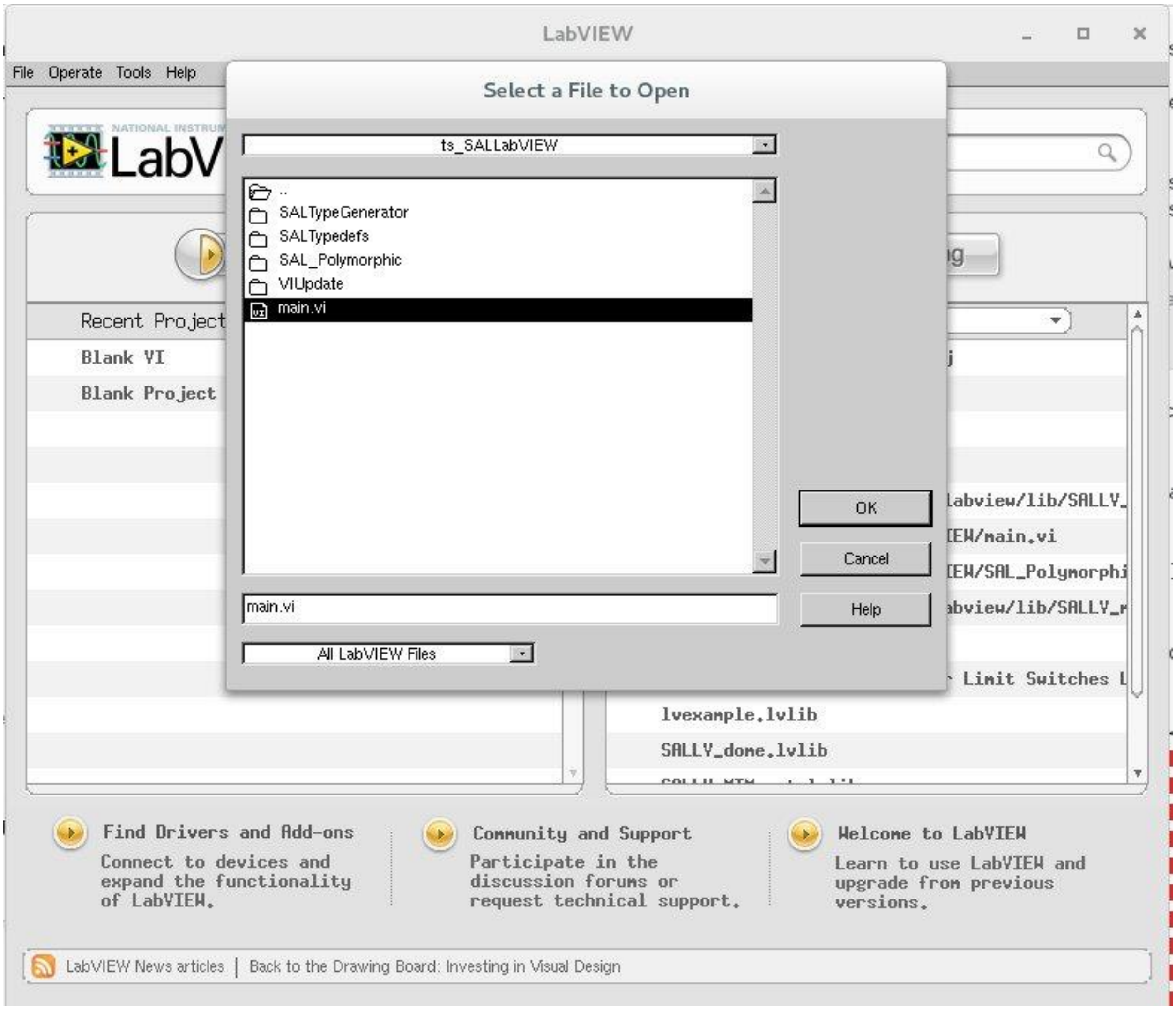
Click Finish on the dialog.



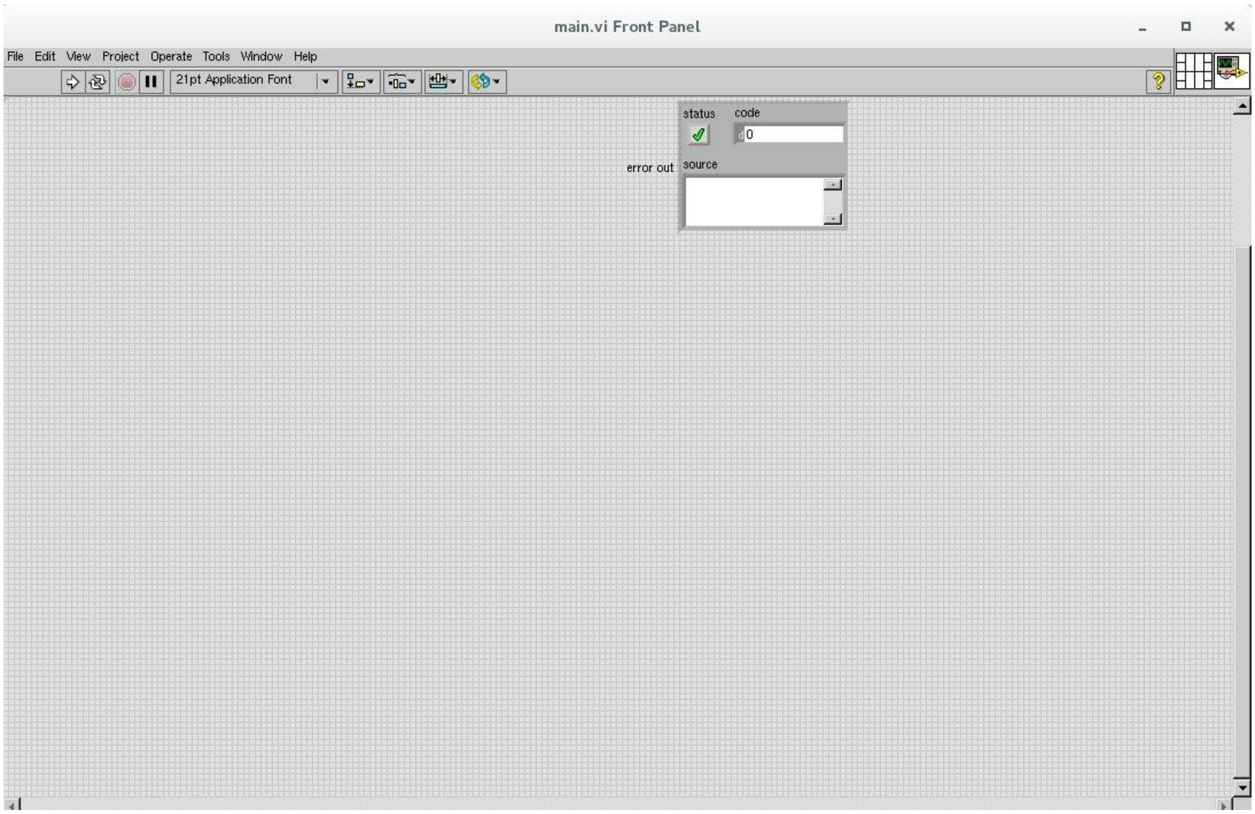

SALLV_m2ms.lvlib on Main Application Instance
File Edit View Project Operate Tools Window Help
Items
Files
SALLV_m2ms.lvlib
subvi
m2ms sal Shm Connect.vi
m2ms sal Shm Monitor.vi
m2ms sal Shm Release.vi
m2ms shm accept Command abort LV.vi
m2ms shm accept Command adjust LV.vi
m2ms shm accept Command axial Forces LV.vi
m2ms shm accept Command disable LV.vi
m2ms shm accept Command enable LV.vi
m2ms shm accept Command exit LV.vi
m2ms shm accept Command slew LV.vi
m2ms shm accept Command standby LV.vi
m2ms shm accept Command start LV.vi
m2ms shm accept Command stop LV.vi
m2ms shm accept Command take Control LV.vi
m2ms shm accept Command update LV.vi
m2ms shm ack Command abort LV.vi
m2ms shm ack Command adjust LV.vi
m2ms shm ack Command axial Forces LV.vi
m2ms shm ack Command disable LV.vi
m2ms shm ack Command enable LV.vi
m2ms shm ack Command exit LV.vi
m2ms shm ack Command slew LV.vi
m2ms shm ack Command standby LV.vi
m2ms shm ack Command start LV.vi
m2ms shm ack Command stop LV.vi
m2ms shm ack Command take Control LV.vi
m2ms shm ack Command update LV.vi
m2ms shm get Event interlock LV.vi
m2ms shm get Event limit Error LV.vi
m2ms shm get Event temp Error LV.vi
m2ms shm get Next Sample ackcmd LV.vi
m2ms shm get Next Sample axial actuators LV.vi
m2ms shm get Next Sample ni9201LV.vi
m2ms shm get Next Sample ni9221LV.vi
m2ms shm get Next Sample ni9426LV.vi
m2ms shm get Next Sample ni9485LV.vi
m2ms shm get Next Sample sensor ilc LV.vi
m2ms shm get Next Sample tangent actuators LV.vi
m2ms shm get Next Sample thermal LV.vi
m2ms shm get Response abort LV.vi
m2ms shm get Response adjust LV.vi
m2ms shm get Response axial Forces LV.vi
m2ms shm get Response disable LV.vi
m2ms shm get Response enable LV.vi
m2ms shm get Response exit LV.vi
m2ms shm get Response slew LV.vi
m2ms shm get Response standby LV.vi

When the LabVIEW import library wizard has completed it is necessary to run another LSST provided VI to finish the generation process.

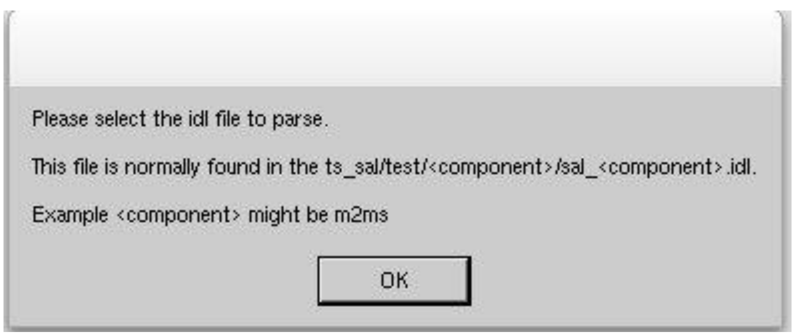
Use the LabVIEW File->Open dialog to locate ts_SALLabVIEW/main.vi



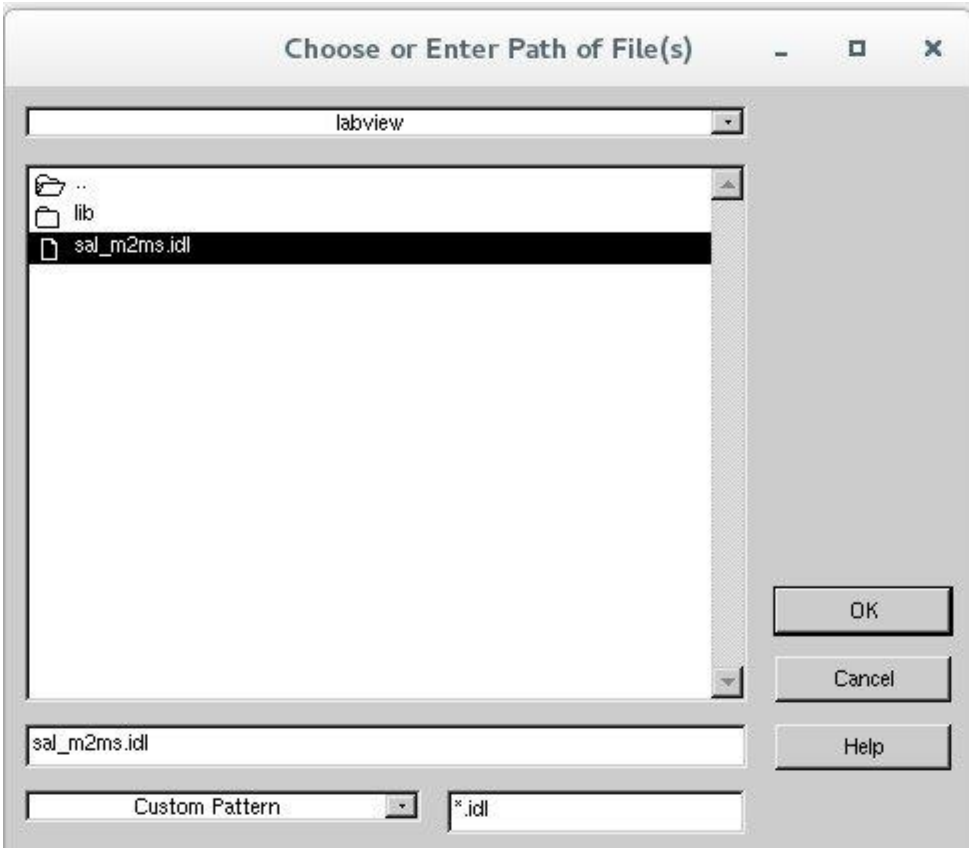
Click OK to run the main.vi VI. It will open a mostly empty interface.



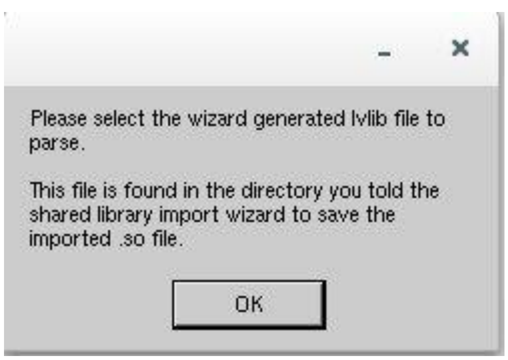
Click the Run icon.



Click OK and select the subsystem IDL file. The correct file should be found in the [subsystem]/labview directory of the SAL_WORK_DIR tree.

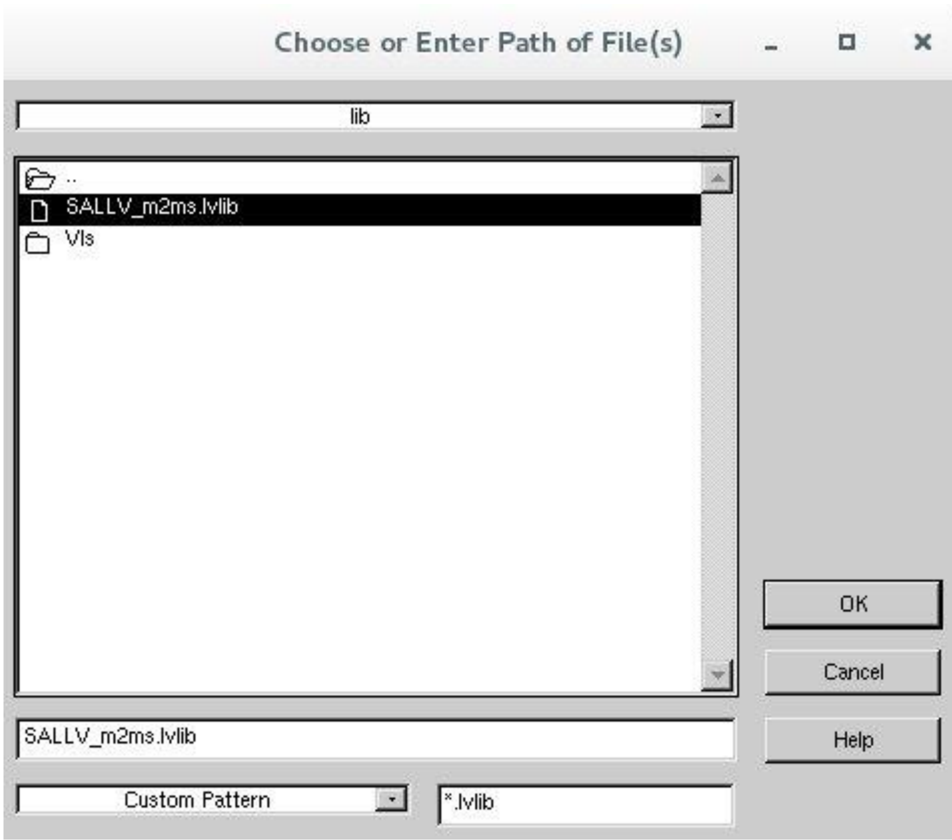


Click OK to select it.



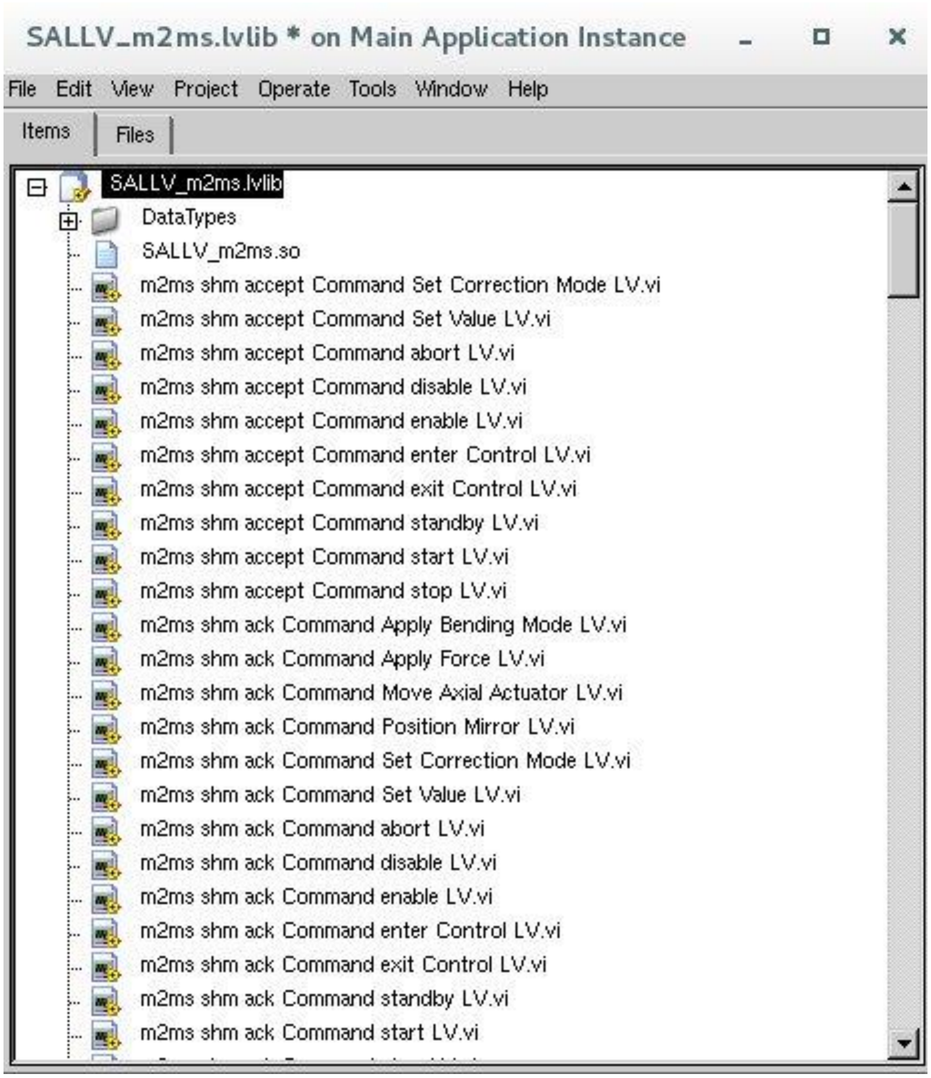
Click OK

Another file dialog then appears for you to select the .lvlib containing the VI's.
This should be located in the [subsystem]/labview/lib directory of the SAL_WORK_DIR tree.



Click OK.

There will then be an extensive period where multiple windows flash on the screen as each VI is individually processed. Finally a library contents window will appear.



Another extensive period will follow where each VI is processed again (you will see them being removed and re-added to the list one-by-one.
Finally the process completes and the main LabVIEW window will reappear.

Once the VI's has been built, you can manually test them by running them against either each other , or against the C++/Java/Python test programs.

Regardless of which option you choose, the LabVIEW environment must be set up first by

1. Running the SALLV_[subsystem]_Monitor daemon in a terminal (this executable manages the shared memory used to mediate the transfer of data to and from LabVIEW).

2. Run the [subsystem]_shm_connect VI and leave it open