European Organisation for Astronomical Research in the Southern Hemisphere

Programme: ELT

Project/WP: Instrumentation Framework

ELT ICS Framework - System Supervisor - User Manual

Document Number: ESO-398434

Document Version: 3

Document Type: Manual (MAN)

Released on: 2024-12-11

Document Classification: Public

Owner:	Kiekebusch, Mario
Validated by PM:	Kornweibel, Nick
Validated by SE:	González Herrera, Juan Carlos
Validated by PE:	Biancat Marchet, Fabio
Approved by PGM:	Tamai, Roberto
	Name

Release

This document corresponds to ifw-sup[1] v4.0.0.

Authors

Name	Affiliation
Kiekebusch, Mario	ESO/DOE/CSE

Change Record from Previous Version

Affected Section(s)	Changes / Reason / Remarks
	See CRE ET-1517
All	All sections updated
2.1,2.2,3.1,3.2	New sections added

[1] https://gitlab.eso.org/ifw/ifw-sup

Contents

1 Introduction

The System Supervisor (SysSup) is one of the top level supervision components in the control software architecture, aimed to provide state control and monitoring capabilities for all instrument subsystems. The System Supervisor shall facilitate a quick assessment of the overall health of the ICS. It is assumed that the SysSup will write the estimated status of the system into the OLDB so that GUIs can read and display it for the users.

1.1 Scope

This document is the user manual for the ELT ICS System Supervisor. The intended audience are ELT users, consortia developers or software quality assurance engineers. This release is to be used by the Consortia developers in trying out the control of instrument hardware functions using the provided libraries and applications, as well as getting acquainted with the design choices and their implementations.

1.2 Acronyms

DB	Database
CCS	Central Control System
ELT	Extremely Large Telescopen
OCF	Observation Coordination Framework
GUI	Graphical User Interface
ICS	Instrument Control System
RAD	Rapid Application Development
SCXML	State Chart XML

1.3 Main Components

The present version of the SysSup covers the following main components:

- The *Supervisor Server* implementation that can control and monitor a configurable number of subsystems from a standard ELT WS.
- The *Subsystem Simulator*, A dummy server implementing the standard interface of a subsystem.
- A Supervisor CLI that simplifies the interface with with Supervisor for engineering purposes.

1.4 Top Directory Structure

The first level of the sup directory contains the following:

```
<root>          # Supervisor component root
├── subsim      # directory containing the modules for the subsystem simulator
├── syssup      # directory containing the modules for the supervisor
├── doc         # directory containing the sphinx user manual
├── test        # directory containing the integration tests
└── wscript     # WAF build script
```

1.5 System Supervisor (syssup)

The server implementation is based on the ICS application framework (rad). Following the ELT and ICS development standards, the client and server are implemented in C++.

1.5.1 Directory Structure

In the present version, the System Supervisor contains:

```
<root>          # syssup root directory
├── client      # C++ client (deprecated)
├── common      # Common C++ library for syssup server
├── supif       # Syssup interface module
├── clib        # Python client library
├── server      # Syssup server module
├── cli         # Syssup Command Line Interface (shell)
└── wscript
```

Where:

- client is a C++ client application that can be used to send commands to the server from the command line.
- common is a library implementing core server classes like actions and activities.
- supif is the CII XML interface module with the payload definition for commands and topics.
- clib is a python library that simplifies the interaction with the server from Python scripts.
- server is the server application (supSupervisor). This is a reference implementation that can be configured to control instrument subsystems.
- cli is a command line interface (CLI) that simplifies the interaction with the Supervisor. It uses the supclib.

1.6 Subsystem Simulators (subsim)

The Supervisor includes a Subsystem Simulator with the purpose of allowing the testing of the Supervisor functionality. The Subsystem Simulator is a dummy process implementing the standard interface and allowing to configure the response of the standard requests. Users can define the time taking to process a particular request and the type of the reply, e.g. success or with a given error.

1.6.1 Directory Structure

In the present version, the Subsystem Simulator contains:

```
<root>          # subsim root directory
├── client      # C++ client (deprecated)
├── common      # Common C++ library for subsim server
├── subsimf     # Subsim interface module
├── clib        # Python client library
├── server      # Subsim server module
├── cli         # Subsim Command Line Interface (shell)
└── wscript
```

Where:

- client is a C++ client application that can be used to send commands to the server from the command line.
- common is a library implementing core server classes like actions and activities.
- subsimif is the CII XML interface module with the payload definition for commands and topics.
- clib is a python library that simplifies the interaction with the server from Python scripts.
- server is the server application (supSubSimulator).
- cli is a command line interface (CLI) that simplifies the interaction with the Simulator. It uses the subsimclib.

2 System Supervisor(supSupervisor)

The System Supervisor provides the functionality for supervision and monitoring of a configurable set of subsystems.

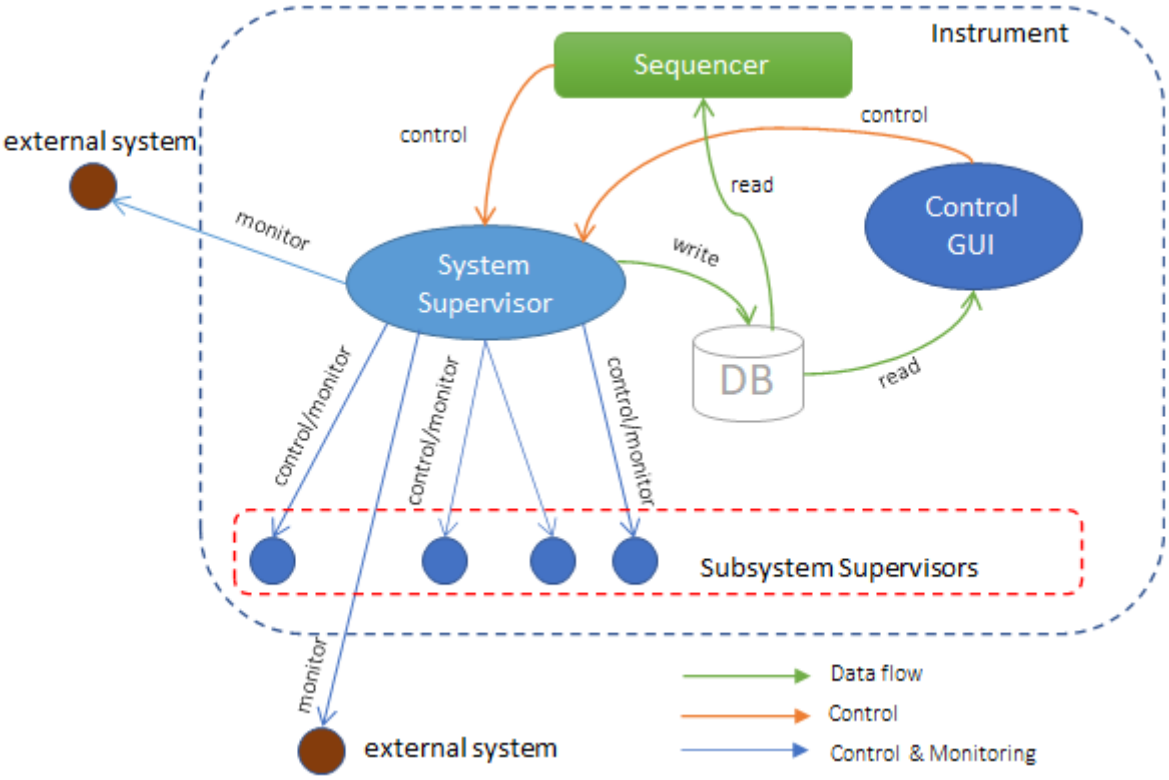

Fig. 1: System Supervisor.

The main components of the *System Supervisor* server are:

- State Machine engine based on SCXML and implemented in RAD. It contains a set of action and activity classes.
- A Subsystem Factory class that creates the instances of all subsystem classes at start-up and based on the server configuration.
- A Facade class that manages the interface between the state machine engine and the subsystem classes.

The System Supervisor uses the OLDB to store run-time information about itself and about the subsystems it monitors. The System Supervisor subscribes to the status information published by the subsystems. The System Supervisor publishes its own status as well like any other subsystem.

2.1 Command Line Arguments

Command line argument help is available under the option --help.

--server-id ARG| -i ARG **(string)**
Server id. If not specified uses the one included in the configuration file.

--config ARG| -c ARG **(string)**
Application configuration file.

--log-level ARG| -l ARG **(enum) [default: *ERROR*]**
Log level to use. One of ERROR, INFO, DEBUG, TRACE.

--log-prop-file ARG| -l ARG **(string)**
Log property file.

--req-endpoint ARG| -l ARG **(string)**
Server MAL Req/Rep endpoint (zpb.rr://<ipaddr>:<port>/).

2.2 Environment Variables

$CFGPATH
Used to resolve configuration file paths.

$DATAROOT
Specifies the default root path used as output directory for FITS metadata. Metadata files are stored under $DATAROOT/fcf/<fcs instance>.

2.3 System Supervisor State Machine

The System Supervisor uses a state machine described in a SCXML format that is interpreted by the state machine engine provided by the rad application framework. (SCXML specification[1]).

[1] https://www.w3.org/TR/scxml/

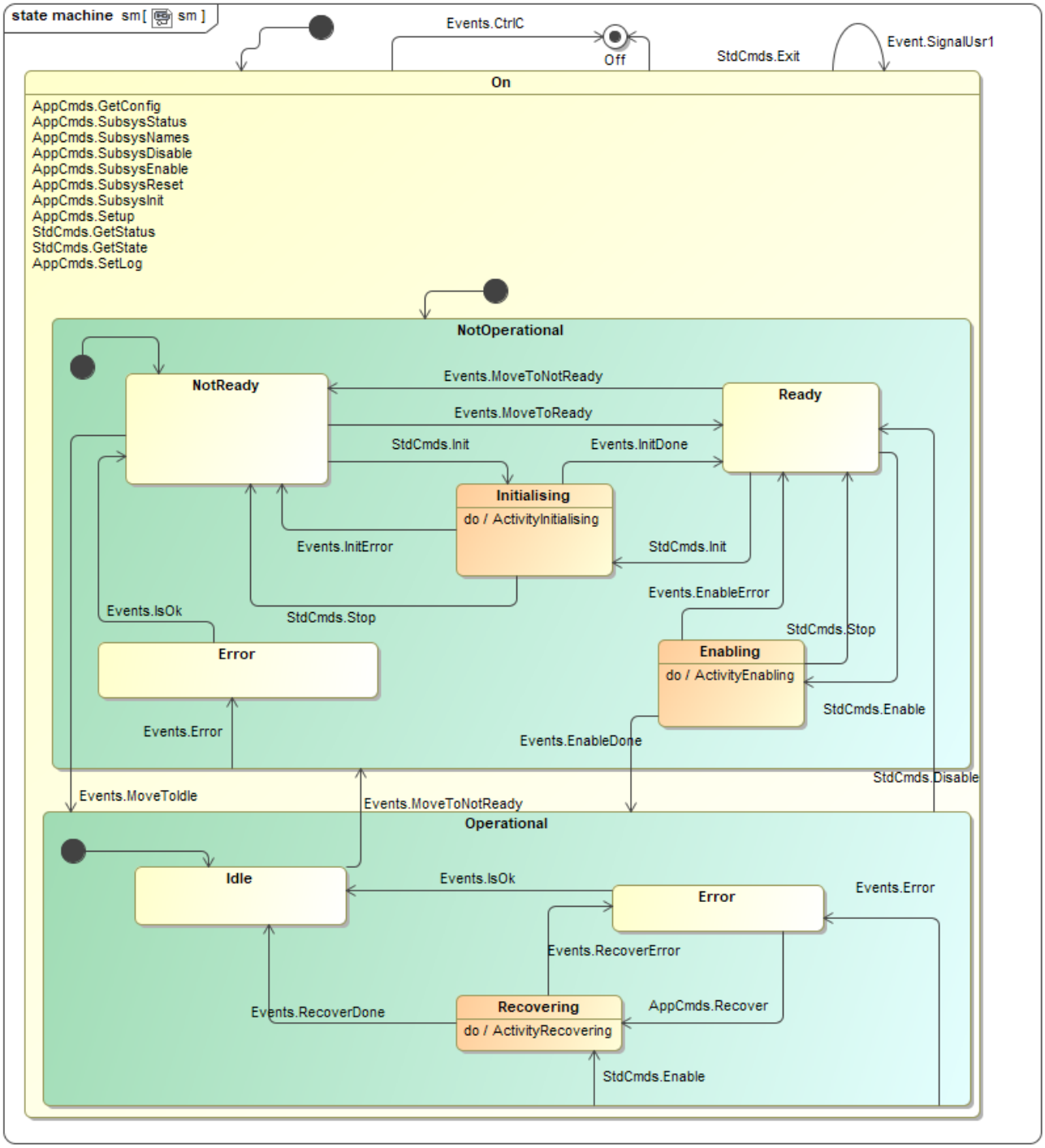

Fig. 2: System Supervisor State Machine Diagram.

Off* –> *NotReady*, event: *Startup

The System Supervisor starts up and goes automatically to *NotOperational/NotReady*. Main server objects are instantiated including the basic application that uses the State Machine engine. The System Supervisor reads its own configuration and completes its initialisation. The system supervisor connects to the configured subsystems and request its current status. With this information it computes the estimated state of the system. The System Supervisor will adapt its own state according the overall state of the subsystems. The above means, that if all the subsystems are Operational/Idle, the System Supervisor will trigger an internal event to go to Operational/Idle. The same happens in any state, which means the System Supervisor could go from Operational/Idle back to NotOperational/NotReady.

***NotReady* –> *Ready*, event: Init**

The Supervisor dispatch the Init command to all the configured subsystems whose access is enabled. Depending of the replied received from the subsystems it will go to *Ready* substate or remain in substate *NotReady*. If at least one command returns with an error or timeout this will prevent the Supervisor reaching *Ready* substate.

NotOperational/Ready* –> *Operational/Idle*, event: *Enable

The System Supervisor goes through the Enabling transient state. If configured subsystems are already Operational, the Supervisor does not affect their state and goes immediately to the Operational state. If subsystems are not operational, System Supervisor will dispatch the Enable request to each of the configured subsystems whose access is enabled . If at least one request fails, it will reply with a failure remaining in *NotOperational/Ready* state.

Operational* –> *NotOperational/Ready*, event: *Disable

The System Supervisor is moved back to NotOperational/Ready substate. However if all subsystems are Operational, the System Supervisor will remain Operational since its state reflects the state of the subsystems it coordinates.

NotOperational/Ready* –> *NotOperational/NotReady*, event *Reset

The System Supervisor is moved back to NotOperational/NotReady substate. The subsystems are not affected by this transition and keep its actual state/substate.

2.4 Configuration

2.4.1 System Supervisor Configuration

The SysSup in version 4.0.0 has been ported to the CII config-ng library. Unlike yaml-cpp, this library allows to define type information for the configuration parameters. The System Supervisor includes a predefined set of configuration definitions. These files can be found in the syssup/server/resources/config directory.

You can find more information about CII config-ng in the following link. (Config-ng manual[2]).

Note: The entry point for the *System Supervisor* configuration is the file that contains the server configuration.

---

server::server_id

This is the id associated with the specific server. This id is used to associate all server configuration parameters as well as the prefix for the DB keys.

server::req_endpoint

This is the endpoint for CII MAL request/reply. The server will listen to incoming commands using this endpoint.

server::pub_endpoint

This is the endpoint for CII MAL pub/sub. The server will publish device topics using this endpoint.

server::db_timeout

This is the server timeout for connecting to the OLDB.

server::scxml

This is the state machine specification file used by the server.

[2] https://www.eso.org/~eltmgr/CII/v2024-03/4.3.0/manuals/html/docs/config-ng.html

server::dictionaries

This is the vector of dictionaries to be used by the server.

server::oldb_prefix

This is the prefix to be used for the DB. This prefix is meant to identify uniquely a given system, e.g. micado.

server::log_properties

log4cplus property file to be used by the server.

server::mon_timeout

Monitor timeout for waiting to establish connections to the subsystems. This value should be rarely bigger than few seconds. Default is 1000 [ms].

server::req_timeout

General command timeout for sending commands to subsystems.

server::ob_modes

Vector of observation modes supported by the instrument. Each observation mode defines a list of associated subsystems.

server::subsystems

This is the vector of subsystems active in the supervisor configuration. Only subsystems listed here will be managed by the supervisor.

Each Subsystem has its own set of configuration parameters

<subsystem id>::scope

This is the scope of each subsystems. It can be internal or external. Requests are not forwarded to external subsystems and the System Supervisor only monitors them.

<subsystem id>::type

This is the subsystem type class. Normally subsystem will use the provided class: sup::syssup::common::Generic

<subsystem id>::rr_endpoint

This is the endpoint for the subsystem CII MAL request/reply. The subsystem listen to incoming commands using this endpoint.

<subsystem id>::ps_endpoint

This is the endpoint for the subsystem CII MAL pub/sub. The subsystem publish its status using this endpoint.

<subsystem id>::access

This is a flag to enable/disable accessibility of a subsystem.

An example of a server configuration is provided below.

```
!cfg.include config/sup/syssup/server/definitions.yaml:
server: !cfg.type:SysSup
    server_id       : 'sup'
    req_endpoint    : "zpb.rr://127.0.0.1:13082/"
    pub_endpoint    : "zpb.ps://127.0.0.1:13345/"
    db_timeout      : 2000
    scxml           : "sup/syssup/server/sm.xml"
    log_properties  : "config/sup/syssup/server/log_properties.cfg"
    oldb_prefix     : "ins1"
    req_timeout     : 60000
    ob_modes        : [
    {
    name: Engineering,
```

(continues on next page)

(continued from previous page)

```
subsystems: ['fcs1','dummy1']
},
{
name: Imaging,
subsystems: ['dummy2']
}
]
subsystems      : [
{
name: 'fcs1',
scope: internal,
type: sup::syssup::common::Generic,
rr_endpoint: "zpb.rr://127.0.0.1:15085/StdCmds",
ps_endpoint: "zpb.ps://127.0.0.1:15045/",
access: true
},
{
name: 'dummy1',
scope: internal,
type: sup::syssup::common::Generic,
rr_endpoint: "zpb.rr://127.0.0.1:15086/StdCmds",
ps_endpoint: "zpb.ps://127.0.0.1:15046/",
access: false
},
{
name: 'dummy2',
scope: internal,
type: sup::syssup::common::Generic,
rr_endpoint: "zpb.rr://127.0.0.1:15087/StdCmds",
ps_endpoint: "zpb.ps://127.0.0.1:15047/",
access: true
}
]
```

2.4.2 Supervisor OLDB

The supervisor stores the actual values of the server configuration parameters into the OLDB. This helps to verify whether the configuration has been loaded correctly. For details of the server configuration parameters, see :ref: *sup_config_ref_*.

Table 1: Supervisor configuration OLDB keys

OLDB Key
<instrument id>/<server id>/cfg/db_timeout
<instrument id>/<server id>/cfg/db_task_period
<instrument id>/<server id>/cfg/dictionaries
<instrument id>/<server id>/cfg/req_timeout
<instrument id>/<server id>/cfg/mon_timeout
<instrument id>/<server id>/cfg/filename
<instrument id>/<server id>/cfg/fits_prefix
<instrument id>/<server id>/cfg/pub_endpoint
<instrument id>/<server id>/cfg/req_endpoint
<instrument id>/<server id>/cfg/scxml
<instrument id>/<server id>/cfg/oldb_prefix
<instrument id>/<server id>/cfg/log_properties
<instrument id>/<server id>/cfg/server_id
<instrument id>/<server id>/cfg/subsystems/<subsystem id>/scope
<instrument id>/<server id>/cfg/subsystems/<subsystem id>/type
<instrument id>/<server id>/cfg/subsystems/<subsystem id>/rr_endpoint
<instrument id>/<server id>/cfg/subsystems/<subsystem id>/ps_endpoint
<instrument id>/<server id>/cfg/subsystems/<subsystem id>/access

2.4.3 Server Status

The server stores the string representation of its state and substate into the OLDB DB.

Table 2: Server status DB keys

OLDB Key
<instrument id>/<server id>/stat/states/state
<instrument id>/<server id>/stat/states/substate
<instrument id>/<server id>/stat/subsystems/<subsystem id>/states/state
<instrument id>/<server id>/stat/subsystems/<subsystem id>/states/substate
<instrument id>/<server id>/stat/subsystems/<subsystem id>/ob_mode

Status Estimation

The estimated state/substate of the overall system is based on the individual subsystem states/substates and according to the following criteria:

Each of the known state/substate strings have associated a coding system to simplify the estimation. In the case of the state, the estimation is just the minimum state withing all managed subsystems. Here we have normally only three possible cases: Undetermined, NotOperational and Operational.

In the case of the substate, the estimation it is similar. The overall substate is the minimum substate with the following exception: * if at least one of the substate of the subsystems is any of the transient substates like SettingUp or Recording. The estimated substate will reflect the minimum transient state. The above helps to report the ongoing activities of the managed subsystems.

Note: The estimation is done by a virtual method of the Supervisor Facade and it could be replaced by the applications if needed.

Warning: The estimation relies on the fact that subsystem publish their status according to the defined format.

2.5 Commands

The commands currently supported by the server are listed here: *List of Commands*.

2.5.1 Error Handling

Supervisor commands throw an exception in case of errors or timeouts. Client applications can catch the exceptions and obtain the error message associated with the function **getDesc()**. This error does not contain neither the history nor the error stack but it normally indicates precisely where the error occurred. Since CII Error service is not yet available, Supervisor cannot use it.

Note: The specific exceptions depends of the given command used.

```
try {
    auto reply = client->GetState();
 } catch (const stdif::ExceptionErr& e) {
    RAD_LOG_ERROR() << "Error reply " << e.getDesc()  << ").";
}
```

2.5.2 Serialization

The *System Supervisor* uses the CII MAL ZPB (ZeroMQ + Google Proto buffers) for serialising commands.

Note: Each command has two parts: a payload and its corresponding reply, see the details in the *supif* module. The normal replies are plain strings.

Setup Command

The *Setup* command is intended to produce a change in the run-time configuration.

Since there is a not long operations associated with the Setup command, this operation is blocking. The Supervisor executes the action and then it send the reply back to the originator.

The interface definition of the *Setup* command can be found in module *supif*.

Warning: The array does not have a fixed size but it has a limit of 100 elements. A limit is needed by the CII XML ICD.

```
<method name="Setup" returnType="string" throws="ExceptionErr">
   <argument name="payload" type="nonBasic" nonBasicTypeName="SetupElem"␣
↪arrayDimensions="(100)"/>
</method>
```

SubsysNames Command

The SubsysNames command reports in a comma separated list, the subsystems managed by the *System Supervisor*. An example of the output generated by the SubsysNames command is shown below. The URI shall be adapted to the correct values.

```
$ supClient zpb.rr://134.171.3.48:30519 SubsysNames ""
subsim2, subsim3
```

SubsysStatus Command

The SubsysStatus command provides information about each subsystem managed by the *System Supervisor*. An example of the output generated by the SubsysStatus command is shown below.

```
$ supClient zpb.rr://134.171.3.48:30519 SubsysStatus ""
subsim2.access = true
subsim2.scope = internal
subsim2.connection_status = Connected
subsim2.state = Operational
subsim2.substate = Idle
subsim3.access = true
subsim3.scope = internal
subsim3.connection_status = Connected
subsim3.state = Operational
subsim3.substate = Idle
```

2.6 Subscriptions

Each subsystem instance created by the factory subscribes to the status of the subsystem. The subscription follows the following naming convention. The *System Supervisor* relies on this convention to monitor the status of the subsystems.

Subsystem	Parameter	end point
<subsystem>	status	<ps endpoint>/std/status

2.7 Publishing

The *System Supervisor* publishes as any other subsystem its estimated state/substate. This can be used to build a hierarchy of subsystems.

Parameter	end point
status	<ps endpoint>/std/status

2.8 Signal Handling

The supervisor handles the SIGUSR1 emitted by Nomad to notify when changes in the template configuration file at run-time. When the Supervisor receives this signal, it reloads the configuration and reconnect to the given subsystem if needed.

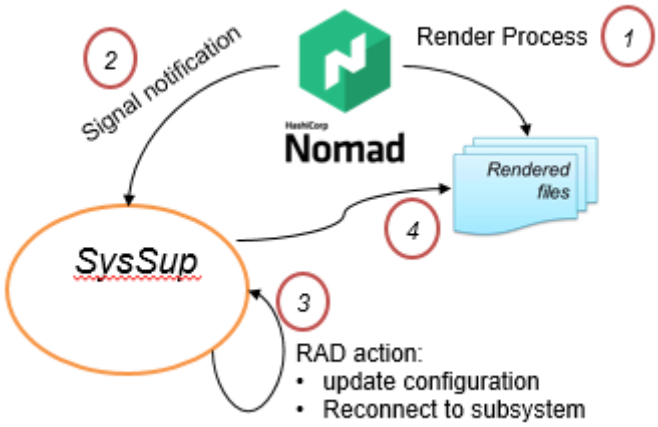

Fig. 3: Supervisor Handling of Nomad Signals.

2.9 Troubleshooting

2.9.1 Logging

The *System Supervisor* implements logging levels according to the log4cplus package where the concept is:

ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF

The basic log levels supported by the SysSup for troubleshooting are listed in the table below.

Name	Verbosity	Description
ERROR	very low	Provide logging only in case of errors.
INFO	low	Provide information for the most important actions.
DEBUG	medium	Provide additional information for the developer.
TRACE	very high	Includes all the function tracing.

To activate a new logging, the command SetLogLevel shall be used. See the example below.

```
$ supClient zpb.rr://134.171.3.48:30519 SetLogLevel "TRACE"
```

2.9.2 Loggers

The *System Supervisor* provides a default configuration (log_cii_properties.cfg) for the logging with the CII logging service. This configuration defines one general logger (app).

Logger	Description
app	General logger for common server classes.

2.9.3 Log File

The default log configuration provides two appenders. One for the console and another one for a file. The file is stored in the CII Logging directory (CII_LOGS). The name of the file is supSupervisor.log.

2.9.4 Logging Viewer

Since version 5.0.0, the logs can be visualised using the CII Logging Viewer.

3 Subsystem Simulator(supSimulator)

The Subsystem Simulator mimics the respond of a generic subsystem process. It is used to have an independent validation of the standard interface used between the System Supervisor and subsystems. The Subsystem Simulator is a dummy RAD based application which implements the standard state machine and the standard interface together with few specific features.

Note: From now on we will use the term Simulator to refer to the Subsystem Simulator.

The Simulator uses the OLDB to store run-time information. The Simulator publishes its own status following the requirements for interacting with the System Supervisor.

3.1 Command Line Arguments

Command line argument help is available under the option --help.

--server-id ARG| -i ARG **(string)**
: Server id. If not specified uses the one included in the configuration file.

--config ARG| -c ARG **(string)**
: Application configuration file.

--log-level ARG| -l ARG **(enum) [default: *ERROR*]**
: Log level to use. One of ERROR, INFO, DEBUG, TRACE.

--log-prop-file ARG| -l ARG **(string)**
: Log property file.

--req-endpoint ARG| -l ARG **(string)**
: Server MAL Req/Rep endpoint (zpb.rr://<ipaddr>:<port>/).

3.2 Environment Variables

$CFGPATH
: Used to resolve configuration file paths.

3.3 Simulator State Machine

The Simulator uses a state machine described in a SCXML format that is interpreted by the state machine engine provided by the rad application framework. (SCXML specification[3]).

[3] https://www.w3.org/TR/scxml/

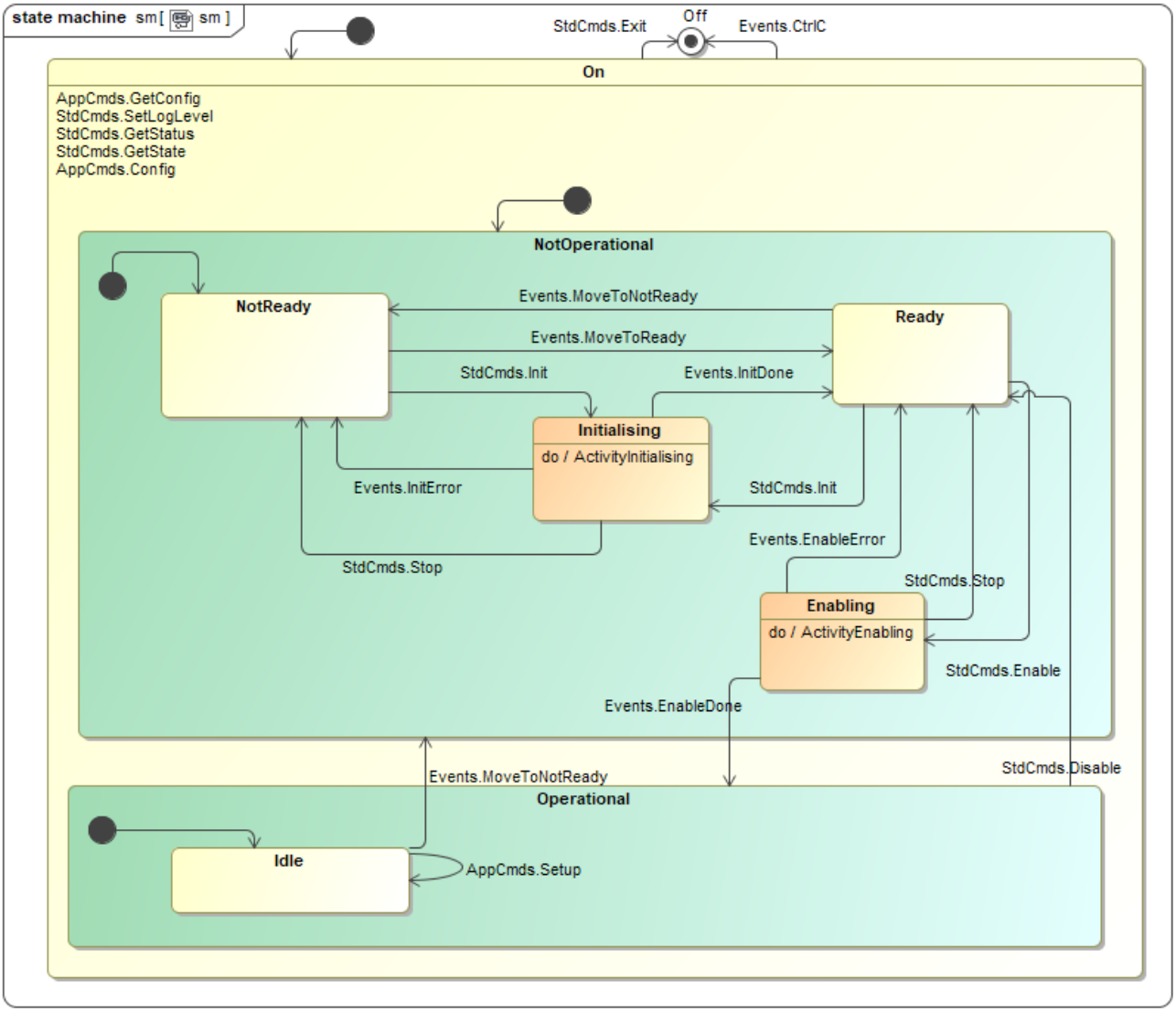

Fig. 1: Simulator State Machine Diagram.

Off* –> *NotReady*, event: *Startup

The Simulator starts up and goes automatically to *NotOperational/NotReady*. Main server objects are instantiated including the basic application that uses the State Machine engine. The Simulator reads its own configuration and completes its initialisation.

***NotReady* –> *Ready*, event: Init**

The Simulator moves from *NotReady* to *Ready* state by going through the *Initialising* transient

state. During this transient state, the simulator will wait according to the configured delay time before to reply to the originator.

NotOperational/Ready –> Operational/Idle*, event: *Enable

The Simulator moves from *NotOperational/Ready* to *Operational/Idle* by going through the *Enabling* transient state. During this transient state, the simulator will wait according to the configured delay time before to reply to the originator.

Operational –> NotOperational/Ready*, event: *Disable

The Simulator is moved back to *NotOperational/Ready* substate.

NotOperational/Ready –> NotOperational/NotReady*, event *Reset

The Simulator is moved back to *NotOperational/NotReady* substate.

3.4 Configuration

3.4.1 SubSystem Simulator Configuration

The server configuration in version 4.0.0 has been ported to the CII config-ng library. Unlike yaml-cpp, this library allows to define type information for the configuration parameters. The Simulator includes a predefined set of configuration definitions. These files can be found in the sub-sim/server/resources/config directory.

You can find more information about CII config-ng in the following link. (Config-ng manual[4]).

server::server_id

This is the id associated with the specific server. This id is used to associate all server configuration parameters as well as the prefix for the DB keys.

server::req_endpoint

This is the endpoint for CII MAL request/reply. The server will listen to incoming commands using

[4] https://www.eso.org/~eltmgr/CII/v2024-03/4.3.0/manuals/html/docs/config-ng.html

this endpoint.

server::pub_endpoint

This is the endpoint for CII MAL pub/sub. The server will publish device topics using this endpoint.

server::db_timeout

This is the server timeout for connecting to the OLDB.

server::scxml

This is the state machine specification file used by the server.

server::req_timeout

General command timeout.

server::log_properties

Log4cplus property configuration file.

server::commands

This is the vector of commands active in the Simulator configuration. Only commands listed here will be managed by the Simulator.

Each command has its own set of configuration parameters

<command id>::reply_ok

This is a flag to indicate if command replies successfully or not. When it is true, the command replies ok otherwise it returns with an error.

<command id>::reply_delay

This is the delay in milliseconds to be used to reply to the originator.

<command id>::reply_ok_message

This is the message to be used when replying successfully.

<command id>::reply_error_msg

This is the message to be used when replying with an error.

<command id>::reply_error_code

This is a flag to enable/disable accessibility of a subsystem.

An example of a server configuration is provided below.

```
!cfg.include config/sup/subsim/server/definitions.yaml:
server: !cfg.type:SubSim
    server_id       : 'subsim1'
    req_endpoint    : "zpb.rr://127.0.0.1:15086/"
    pub_endpoint    : "zpb.ps://127.0.0.1:15046/"
    db_timeout      : 2000
    log_properties  : "config/utils/bat/log_properties.cfg"
    scxml           : "config/sup/subsim/server/sm.xml"
    oldb_prefix     : "ins1"
    commands: [
    {
    name: 'Init',
    reply_ok: true,
    reply_delay: 3000,
    reply_error_msg: "Init failed - subsystem could not initialize"
    },
    {
    name: 'Enable',
    reply_ok: true,
    reply_delay: 3000,
    reply_error_msg: "ERROR: Enable failed"
    },
```

(continues on next page)

(continued from previous page)

```
{
name: 'Disable',
reply_ok: true,
reply_delay: 3000,
reply_error_msg: "ERROR: Enable failed"
},
]
```

3.4.2 Simulator OLDB

The simulator stores the actual values of the server configuration parameters into the OLDB . This helps to verify whether the configuration has been loaded correctly. For details of the server configuration parameters, see :ref: *sup_sim_config_ref_*.

Table 1: Simulator configuration OLDB keys

OLDB Key
<server id>/cfg/db_timeout
<server id>/cfg/dictionaries
<server id>/cfg/req_timeout
<server id>/cfg/mon_timeout
<server id>/cfg/filename
<server id>/cfg/pub_endpoint
<server id>/cfg/req_endpoint
<server id>/cfg/scxml
<server id>/cfg/server_id
<server id>/cfg/<command id>/reply_delay
<server id>/cfg/<command id>/reply_error_msg
<server id>/cfg/<command id>/reply_ok

3.4.3 Server Status

The server stores the string representation of its state and substate into the DB.

Table 2: Server status OLDB keys

OLDB Key
<instrument id>/<server id>/stat/states/state
<instrument id>/<server id>/stat/states/substate

3.5 Commands

3.5.1 Error Handling

Simulator commands throw an exception (subsimif::ExceptionErr) in case of errors or timeouts from Simulator specific commands. For standard commands, the server throws a stdif::Exception. Client applications can catch the exceptions and obtain the error message associated with the function **getDesc()**. This error does not contain neither the history nor the error stack but it normally indicates precisely where the error occurred.

```
try {
    auto reply = client->GetState();
 } catch (const stdif::ExceptionErr& e) {
    RAD_LOG_ERROR() << "Error reply " << e.getDesc()  << ").";
}
```

3.5.2 Serialization

The *Subsystem Simulator* uses the CII MAL ZPB (ZeroMQ + Google Proto buffers) for serialising commands.

Note: Each command has two parts: a payload and its corresponding reply, see the details in the *supif* module. The normal replies are plain strings.

3.5.3 Sever Commands

The commands (events) currently supported by the *Simulator* are:

Table 3: Simulator commands

Command	Parameters	Interface
Init	“”	stdif
Enable	“”	stdif
Disable	“”	stdif
Reset	“”	stdif
GetState	“”	stdif
GetStatus	“”	stdif
Exit	“”	stdif
SetLogLevel	“<ERROR|INFO|DEBUG|TRACE>”	subsimif
Setup	“”	subsimif
Config	“<config buffer>”	subsimif
GetConfig	“”	subsimif

3.5.4 GetConfig Command

The *GetConfig* command returns the actual configuration used by the Simulator. This command is useful when starting the process with Nomad where configuration might be rendered.

3.6 Publishing

The *Subsystem Simulator* publishes as any other subsystem its state/substate. This is a requirement for enable visibility from the System Supervisor.

Parameter	**end point**
status	<ps endpoint>/std/status

3.7 Troubleshooting

3.7.1 Logging

The *Subsystem Simulator* implements logging levels according to the log4cplus package where the concept is:

ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF

The basic log levels supported by the Simulator for troubleshooting are listed in the table below.

Name	**Verbosity**	**Description**
ERROR	very low	Provide logging only in case of errors.
INFO	low	Provide information for the most important actions.
DEBUG	medium	Provide additional information for the developer.
TRACE	very high	Includes all the function tracing.

3.8 C++ Simulator Client

The client application (*supsimClient*) is a simple utility allowing to send messages to the *Simulator* from the command line. In this context we use the words messages and events as synonyms. The supDummyClient uses two interface module (stdif and supsimif) to compose the payload of the messages. For simplicity purposes, the associated interface for each command is hidden to the user. The *supsimClient* sends the messages using CII MAL request/reply.

Note: This client uses the synchronous version of the stdif and supsimif interfaces.

```
$ supsimClient <serviceURI> <command> ["<parameters>"]

Where
    <serviceURI> destination of the command (e.g. zpb.rr://127.0.0.1:12081)
    <command>    command to be sent to the server (e.g. Init)
    <parameters> optional parameters of the command.
```

3.8.1 Examples

```
$ supsimClient zpb.rr://134.171.3.48:30519 Init
```

Note: The Config command is not supported by the C++ client.

4 Python Client Library

It is possible to communicate with the *Subsystem Simulator* through clients developed in Python. The Simulator provides a library that simplifies the interaction with the System Supervisor (clib).

Users might want to interact directly with Supervisor ICD binding methods. This is, of course possible, but it is outside the scope of this library.

Note: The Supervisor python library uses the synchronous mal interface.

4.1 Error Handling

The clib reports as a RuntimeError exceptions that may be delivered by the Simulator.

4.2 Classes

The clib library provides one class that encapsulates the interface with the Supervisor. This class is the SubsimCommands class.

4.2.1 SubsimCommands

The constructor of the SubsimCommands class support two parameters: uri and timeout. The timeout is optional and has a default of one minute, expressed in milliseconds.

The class handles two MAL client interfaces to deal with standard commands and Simulator specific commands. The correct interface will be selected according to the method used so this is hidden to the user.

4.2.2 Methods for Command Interface

Method	parameters	interface
setup	None	subsimif
config	None	subsimif
get_config	None	subsimif
state	None	stdif
status	None	stdif
init	None	stdif
enable	None	stdif
disable	None	stdif
reset	None	stdif
stop	None	stdif

4.2.3 Additional Methods

The SubsimCommands class provides additional methods which uses the Config command to facilitate the update of the configuration at run-time.

Method	parameters	interface
cfgreply	<cmd>, <flag>	subsimif
cfgdelay	<cmd>, <delay>	subsimif
cfgerrmsg	<cmd>, <msg>	subsimif

4.3 Examples

4.3.1 Retrieving the Status

```
import ifw.sup.subsim.clib.subsim_commands as subsim

uri = "zpb.rr://134.171.3.48:28206"
subsimif = subsim.SubsimCommands(uri)
print(subsimif.status())
```

5 Simulator CLI

The Simulator CLI (*supsimcli*) provides a experimental command shell with simple commands aiming to simplify the interaction with the Simulator. The Simulator shell can be invoked issuing the command supsimcli.

5.1 Command Line Parameters

The supsimcli offers few command line parameters. If no parameters are specified. The supsimcli shell commands are not necessary using the same names as the MAL interfaces with the purpose to shorten the commands names.

Parameter	Description
–uri	if the URI is specified, the subsimcli will use it to connect to the server
–name	When using nomad, one could specify the name of the service instead of the URI
–module	Custom interface library
–class_name	Custom command class name
–timeout	Timeout for CII MAL requests in ms
–log_level	log level (ERROR, INFO, DEBUG)
–help	Show the usage message

Warning: The supsimcli shell assumes NOMAD/CONSUL services are up and running. If this is not the case then only –uri parameter can be used.

Note: The supsimcli shell was created for the Simulator but since it uses the standard interface, it can be used for any server implementing this interface, although only for the standard events like init, enable, disable, etc.

```
supsimcli --uri zpb.rr://134.171.3.48:30269
supsimSh>?
Available command list:
 - cfgdelay
 - cfgreply
 - cfgerrmsg
 - disable
 - enable
 - help
 - init
 - reset
```

(continues on next page)

(continued from previous page)

```
- setloglevel
- setup
- get_config
- state
- status
```

Command	Parameters	Description
disable		sends the disable (stdif) event to the connected server
enable		sends the enable (stdif) event to the connected server
help		print the list of supported commands
init		sends the init (stdif) event to the connected server
reset		sends the reset (stdif) event to the connected server
state		sends the GetState (stdif) event to the connected server
status		sends the GetStatus (stdif) event to the connected server
setup		sends the setup (subsimif) event to the connected server
get_config		sends the GetConfig (subsimif) event to the connected server
cfgreply	<cmd>,<true|false>	Configure reply flag for a given command. If flag is true, the command will return successfully otherwise with an error.
cfgdelay	<cmd>,<delay>	Configure reply delay for a given command in milliseconds.
cfgerrmsg	<cmd>,<msg>	Configure reply error message for a given command. The reply flag has to be false to take any effect.
setlogloevel	<INFO|DEBUG|TRACE>	sends the SetLogLevel (stdif) event to the connected server
ctrl-d		Stop the shell

5.1.1 Setting Configuration Parameters

Forcing Init command to finalize with an error:

```
supsimSh> cfgreply Init,false
reply> = OK
supsimSh>
```

Changing delay time for Init command:

```
supsimSh> cfgdelay Init,3000
reply> = OK
supsimSh>
```

6 Client Application

The client application (*supClient*) is a simple utility allowing to send commands to the *System Supervisor* from the command line. In this context we use the words commands and events as synonyms. The supClient uses two interface module (stdif and supif) to compose the payload of the messages. For simplicity purposes, the associated interface for each command is hidden to the user. The *supClient* sends the messages using CII MAL request/reply.

Note: This client uses the synchronous version of the stdif and supif interfaces.

```
$ supClient <serviceURI> <command> ["<parameters>"]

Where
    <serviceURI> destination of the command (e.g. zpb.rr://127.0.0.1:12081)
    <command>    command to be sent to the server (e.g. Init)
    <parameters> optional parameters of the command.
```

Warning: The URI shall not contain the '/' at the end otherwise the client will hang trying to connect to a non existing server.

6.1 List of Commands

The commands (events) currently supported by the *supClient* utility are:

Table 1: Client commands

Command	Parameters	Description
Init	“”	Moves the server from NotReady to Ready state.
Enable	“”	Moves the server from Ready to Operational state.
Disable	“”	Moves the server from Operational to Ready state.
Reset	“”	Moves the server to NotReady state.
GetState	“”	Get the state of the server.
GetStatus	“”	Get the status of the server.
Exit	“”	Stop the server.
SetLogLevel	“<ERROR|INFO|DEBUG|TRACE>”	Change the logging level of the server.
Recover	“”	Recover the server in case it entered in error state.
GetConfig	“”	Get the actual configuration of the server.
SubsysNames	“”	Get the list of subsystems coordinated by the Supervisor.
SubsysStatus	“[<subsystem id>]”	Get the status of a subsystem.
SubsysInit	“<subsystem id>”	Move a given subsystem to Ready state.
SubsysEnable	“<subsystem id>”	Move a given subsystem to Operational state.
SubsysDisable	“<subsystem id>”	Move a given subsystem from Operational back to Ready state.
SubsysReset	“<subsystem id>”	Move a given subsystem to NotReady state.

Note: The subsystem commands like SubsysInit or SubsysEnable control the individual state of a subsystem.

6.1.1 Examples

Note: The following examples assume the server is listening for incoming events under the URI zpb.rr://127.0.0.1:12081 in the local host.

Initialising the server

```
$ supClient zpb.rr://127.0.0.1:12081  Init ""
```

Moving the server to Operational state

```
$ supClient zpb.rr://127.0.0.1:12081  Enable ""
```

7 Python Client Library

It is possible to communicate with the *System Supervisor* through clients developed in Python. The SysSup provides a library that simplifies the interaction with the System Supervisor (clib).

Users might want to interact directly with Supervisor ICD binding methods. This is, of course possible, but it is outside the scope of this library.

Note: The Supervisor python library provides both versions of the MAL communication: synchronous and asynchronous.

7.1 Error Handling

The clib reports as a RuntimeError exceptions that may be delivered by the System Supervisor.

7.2 Classes

The clib library provides two classes that encapsulates the interface with the Supervisor. The class SysSupCommands (synchronous) and the class SysSupAsyncCommands (asynchronous). Both classes provide the same functionality.

7.2.1 SysSupCommands

The constructor of the SysSupCommands class support two parameters: uri and timeout. The timeout is optional and has a default of one minute, expressed in milliseconds.

The class handles two MAL client interfaces to deal with standard commands and Supervisor specific commands. The correct interface will be selected according to the method used so this is hidden to the user.

7.2.2 Methods for Command Interface

Method	parameters	interface
setup	<setup buffer>	supif
get_config		supif
set_config		supif
subsystem_status		supif
subsystem_init	<subsystem name>	supif
subsystem_enable	<subsystem name>	supif
subsystem_disable	<subsystem name>	supif
subsystem_reset	<subsystem name>	supif
subsystem_names	None	supif
state	None	stdif
status		stdif
init	None	stdif
enable	None	stdif
disable	None	stdif
reset	None	stdif
stop	None	stdif

7.3 Examples

7.3.1 Retrieving the Status

```
import ifw.sup.syssup.clib.syssup_commands as sup

uri = "zpb.rr://134.171.3.48:30269"
supif = sup.SysSupCommands(uri)
print(supif.status())
NotOperational;Ready
```

7.4 Supervisor CLI

The SysSup python library provides already an easy way to interact with the Supervisor. However it might not be the best choice when more interactivity is needed. The SysSup CLI provides a experimental command shell with simple commands aiming to simplify the interaction with the Supervisor. The SysSup shell can be invoked issuing the command supcli.

7.4.1 Command Line Parameters

The supcli offers some few command line parameters. If no parameters are specified, the supcli will use the default name service and use nomad/consul to obtain the correct IP and port numbers of the Supervisor. The syssup shell commands are not necessary using the same names as the MAL interfaces with the purpose to shorten the commands names. Commands that could take long time, are executed in a dedicated thread to allow the shell to continue being responsive while waiting for the answer from the previous command. Asynchronous CII cannot be used here because unlike in C++, it is not yet available in python (see ECII-365).

Parameter	Description
–uri	if the URI is specified, the supcli will use it to connect to the server
–name	When using nomad, one could specify the name of the service instead of the URI
–module	Custom interface library
–class_name	Custom command class name
–timeout	Timeout for CII MAL requests in ms
–log_level	log level (ERROR, INFO, DEBUG)
–help	Show the usage message

Warning: The supcli shell assumes NOMAD/CONSUL services are up and running. If this is not the case then only –uri parameter can be used.

Note: The supcli shell was created for the Supervisor but since it uses the standard interface, it can be used for any server implementing this interface, although only for the standard events like init, enable, disable, etc.

```
supcli --uri zpb.rr://134.171.3.48:30269
supSh>?
Available command list:
- disable
- enable
- get_config
- help
- init
- is_operational
- reset
- set_config
- set_obmode
- setaccess
- setloglevel
```

(continues on next page)

(continued from previous page)

- state
- status
- stop
- subsystem_disable
- subsystem_enable
- subsystem_init
- subsystem_names
- subsystem_reset
- subsystem_status

Command	Parameters	Description
disable		sends the disable (stdif) event to the connected server.
enable		sends the enable (stdif) event to the connected server.
help		print the list of supported commands
init		sends the init (stdif) event to the connected server.
reset		sends the reset (stdif) event to the connected server.
state		sends the GetState (stdif) event to the connected server
status		sends the GetStatus (stdif) event to the connected server
set_obmode	<mode>	Set observation mode.
setaccess	<subsystem>, <access level>	Set the access level for a particular subsystem. existing configuration. Example: setaccess fcs1,true
get_config		Get the actual configuration used by the server
set_config	<yaml string>	Update the actual configuration. The parameter shall be a valid yaml string. The configuration will be merged to the existing configuration. Example: set_config {server: {req_timeout: 15000}}
setloglevel	<level>,<logger>	Update logging level of the server.
subsystem_disable	<subsystem>	Forwarded the disable command to the subsystem.
subsystem_enable	<subsystem>	Forwarded the enable command to the subsystem.
subsystem_init	<subsystem>	Forwarded the init command to the subsystem.
subsystem_names		Get the list of the subsystems managed by the server.
subsystem_reset	<subsystem>	Forwarded the reset command to the subsystem.
subsystem_status	[<subsystem>]	Get the detailed status of the subsystems managed by the server
ctrl-d		Stop the shell

7.4.2 JSON Schema

The schema used by the supervisor to validate the syntax of the setup buffer is described below.

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "type": "object",
   "title": "Supervisor schema",
   "type": "array",
   "items": {
      "type": "object",
      "properties": {
         "id": {
            "type": "string",
            "description": "Subsystem identifier."
         },
      "param": {
         "$ref": "#/definitions/param"
         }
      }
   },
   "definitions": {
      "param": {
         "type": "object",
         "properties": {
            "subsystem": {
               "$ref": "#/definitions/subsystem"
               }
            },
         "required": [
            "subsystem"
            ]
      },
      "subsystem": {
         "type": "object",
            "properties": {
               "access": {
                  "type": "boolean",
                  "description": "Subsystem access flag."
               }
            },
            "required": [
               "access"
            ]
         }
```

(continues on next page)

(continued from previous page)

```
        }
}
```

8 Getting Started

8.1 Log-in

Login to a standard ELT machine.

8.2 IFW Software

If not yet done, retrieve and install the complete ICS Framework from the ESO RPMs repository. For more details, please have a look to the installation procedure here[5]

You should create your software based on the provided template by following the procedure in the Getting Started guide here[6]

This guide assumes you have followed all the steps in the above guide.

8.3 SysSup Configuration

The project template includes a sample of a System Supervisor configuration. After executing the *cookiecutter* command with the provided template, you can build, install and deploy the provided example.

Please follow the instructions in the general Getting Started guide here[7]

The System Supervisor requires a configuration file including all the definition of all subsystems managed by the server. A pre-defined configuration has been created as part of the generation process. This configuration includes five subsystems. They are all considered as internal subsystems.

Generated server configuration: <ins id>/resource/nomad/syssup.yml.tpl. This files contains the Nomad tags so it should not be used directly but through a nomad job command.

```
!cfg.include config/sup/syssup/server/definitions.yaml:

server: !cfg.type:SysSup

    server_id       : 'sup'
    req_endpoint    : "zpb.rr://{{ range service "syssup-req" }}{{ .Address }}:{{ .Port }}{{ end }}/"
    pub_endpoint    : "zpb.ps://{{ range service "syssup-pub" }}{{ .Address }}:{{ .Port }}{{ end }}
↪/"
    db_timeout      : 2000
    scxml           : "config/sup/syssup/server/sm.xml"
    log_properties  : "config/sup/syssup/server/log_properties.cfg"
```

(continues on next page)

[5] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/installation.html
[6] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/guide.html
[7] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/guide.html

(continued from previous page)

```
    req_timeout     : 120000
    dictionaries    : []
    mon_timeout     : 2000
    oldb_prefix     : 'tins'
    ob_modes        : [
    {
    name: Imaging,
    subsystems: ['subsim2', 'subsim3']
    }
    ]
    subsystems      : [
    {
    name: fcs,
    scope: internal,
    type: sup::syssup::common::Generic,
    rr_endpoint: "zpb.rr://{{ range service "fcs-req" }}{{ .Address }}:{{ .Port }}{{ end }}/StdCmds
↪",
    ps_endpoint: "zpb.ps://{{ range service "fcs-pub" }}{{ .Address }}:{{ .Port }}{{ end }}/",
    access: true
    },
    {
    name: subsim2,
    scope: internal,
    type: sup::syssup::common::Generic,
    rr_endpoint: "zpb.rr://{{ range service "subsim2-req" }}{{ .Address }}:{{ .Port }}{{ end }}/
↪StdCmds",
    ps_endpoint: "zpb.ps://{{ range service "subsim2-pub" }}{{ .Address }}:{{ .Port }}{{ end }}/",
    access: true
    },
    {
    name: subsim3,
    scope: internal,
    type: sup::syssup::common::Generic,
    rr_endpoint: "zpb.rr://{{ range service "subsim3-req" }}{{ .Address }}:{{ .Port }}{{ end }}/
↪StdCmds",
    ps_endpoint: "zpb.ps://{{ range service "subsim3-pub" }}{{ .Address }}:{{ .Port }}{{ end }}/",
    access: true
    },
    {
    name: TestSim1,
    scope: internal,
    type: sup::syssup::common::Generic,
    rr_endpoint: "zpb.rr://{{ range service "tinsccf-req" }}{{ .Address }}:{{ .Port }}{{ end }}/
```

(continues on next page)

(continued from previous page)

```
↪StdCmds",
  ps_endpoint: "zpb.ps://{{ range service "tinsccf-pub" }}{{ .Address }}:{{ .Port }}{{ end }}/",
  access: true
  },
  {
  name: ocm,
  scope: internal,
  type: sup::syssup::common::Generic,
  rr_endpoint: "zpb.rr://{{ range service "ocm-req" }}{{ .Address }}:{{ .Port }}{{ end }}/std",
  ps_endpoint: "zpb.ps://{{ range service "ocm-pub" }}{{ .Address }}:{{ .Port }}{{ end }}/",
  access: true
  }
  ]
```

Note: This configuration shall be adapted to the instrument specific needs.

8.4 System Supervisor Logs

```
$ nomad logs -job syssup
```

The output of the server shall be something like the following:

```
2021-04-26T09:33:44.931933  INFO Application ocfSupervisor started.
2021-04-26T09:33:44.941777  INFO PS Endpoint: zpb.ps://134.171.3.48:29958/std/status
2021-04-26T09:33:45.047840  INFO Updating DB
2021-04-26T09:33:45.048304  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:31156/
↪StdCmds>
2021-04-26T09:33:45.048465  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:26592/
↪StdCmds>
2021-04-26T09:33:45.048607  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:25925/
↪StdCmds>
2021-04-26T09:33:45.048749  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22497/
↪StdCmds>
2021-04-26T09:33:45.048892  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22123/std>
2021-04-26T09:33:45.049533  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:31156/
↪StdCmds>
2021-04-26T09:33:45.050409  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:26592/
↪StdCmds>
2021-04-26T09:33:45.050865  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:25925/
↪StdCmds>
2021-04-26T09:33:45.051282  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22497/
```

(continues on next page)

(continued from previous page)

```
↪StdCmds>
2021-04-26T09:33:45.051692  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22123/std>
2021-04-26T09:33:45.086988  INFO min state: -7
2021-04-26T09:33:45.087545  INFO min state: -7
2021-04-26T09:33:45.087728  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:31156/
↪StdCmds>
2021-04-26T09:33:45.087748  INFO [fcs] Generic Connect:  zpb.rr://134.171.3.48:31156/StdCmds
2021-04-26T09:33:45.087835  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:26592/
↪StdCmds>
2021-04-26T09:33:45.087848  INFO [subsim2] Generic Connect:  zpb.rr://134.171.3.48:26592/
↪StdCmds
2021-04-26T09:33:45.087900  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:25925/
↪StdCmds>
2021-04-26T09:33:45.087913  INFO [subsim3] Generic Connect:  zpb.rr://134.171.3.48:25925/
↪StdCmds
2021-04-26T09:33:45.087954  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22497/
↪StdCmds>
2021-04-26T09:33:45.087967  INFO [TestSim1] Generic Connect:  zpb.rr://134.171.3.48:22497/
↪StdCmds
2021-04-26T09:33:45.088001  INFO Subsys RR endpoint -  = <zpb.rr://134.171.3.48:22123/std>
2021-04-26T09:33:45.088012  INFO [ocm] Generic Connect:  zpb.rr://134.171.3.48:22123/std
2021-04-26T09:33:45.089118  INFO Name: fcs
2021-04-26T09:33:45.089146  INFO Name: subsim2
2021-04-26T09:33:45.089162  INFO Name: subsim3
2021-04-26T09:33:45.089178  INFO Name: TestSim1
2021-04-26T09:33:45.089193  INFO Name: ocm
```

By default the System Supervisor saves its logs into a file $HOME/supSupervisor.log. A much easier way to see the logs is to monitor this file.

```
$ tail -f $HOME/supSupervisor
```

8.5 Interacting with the Server

The best way to interact with the server is using the SysSup CLI.

```
$ supcli
zpb.rr://134.171.3.48:27943
supSh>
```

Note: The shell will connect automatically to the running server.

8.5.1 Getting list of registered subsystems

```
supSh> subsystem_names
reply> = fcs, subsim2, subsim3, TestSim1, ocm
```

8.5.2 Getting status of subsystems

```
supSh> subsystem_status
reply> = fcs.access = true
fcs.scope = internal
fcs.connection_status = Connected
fcs.state = Operational
fcs.substate = Idle
subsim2.access = true
subsim2.scope = internal
subsim2.connection_status = Connected
subsim2.state = NotOperational
subsim2.substate = NotReady
subsim3.access = true
subsim3.scope = internal
subsim3.connection_status = Connected
subsim3.state = Operational
subsim3.substate = Idle
TestSim1.access = true
TestSim1.scope = internal
TestSim1.connection_status = Connected
TestSim1.state = Operational
TestSim1.substate = Idle
ocm.access = true
ocm.scope = internal
ocm.connection_status = Connected
ocm.state = Operational
ocm.substate = Idle
```

8.5.3 Restarting an individual subsystem

```
supSh> subsystem_reset subsim3
reply> = OK

supSh> subsystem_init subsim3
reply> = OK init completed.
```

(continues on next page)

(continued from previous page)

```
supSh> subsystem_enable subsim3
reply> = OK enable completed.
```

8.5.4 Getting global state/substate

```
supSh> status
reply> = Operational;Idle
```

8.5.5 Changing the access level for a subsystem

A subsystem can be ignored by System Supervisor. This can be done by changing its access level to false. When the subsystem is ignored, its state is not considered in the overall state of the system.

```
supSh> setaccess fcs,false
reply> = OK
supSh> subsystem_status fcs
reply> = fcs.access = false
```

8.5.6 Getting the actual configuration used by the System Supervisor

```
supSh> get_config
reply> = ...
server: !cfg.type:SysSup
    server_id: sup
    req_endpoint: zpb.rr://134.171.2.248:28459/
    pub_endpoint: zpb.ps://134.171.2.248:20319/
    db_endpoint: 134.171.2.248:26498
    db_timeout: 2000
    scxml: config/sup/syssup/server/sm.xml
    log_properties: config/sup/syssup/server/log_properties.cfg
    mon_timeout: 2000
    req_timeout: 15000
    dictionaries: []
    oldb_prefix: tins
    fits_prefix: ""
    ob_modes:
        - name: Imaging
          subsystems: [subsim2, subsim3]
    subsystems:
```

(continues on next page)

(continued from previous page)

```
- name: fcs
  scope: internal
  type: sup::syssup::common::Generic
  rr_endpoint: zpb.rr://134.171.2.248:25124/StdCmds
  ps_endpoint: zpb.ps://134.171.2.248:27455/
  access: True

- name: subsim2
  scope: internal
  type: sup::syssup::common::Generic
  rr_endpoint: zpb.rr://134.171.2.248:24273/StdCmds
  ps_endpoint: zpb.ps://134.171.2.248:25119/
  access: true

- name: subsim3
  scope: internal
  type: sup::syssup::common::Generic
  rr_endpoint: zpb.rr://134.171.2.248:27557/StdCmds
  ps_endpoint: zpb.ps://134.171.2.248:29982/
  access: true

- name: TestSim1
  scope: internal
  type: sup::syssup::common::Generic
  rr_endpoint: zpb.rr://134.171.2.248:21019/StdCmds
  ps_endpoint: zpb.ps://134.171.2.248:23543/
  access: true

- name: ocm
  scope: internal
  type: sup::syssup::common::Generic
  rr_endpoint: zpb.rr://134.171.2.248:29388/std
  ps_endpoint: zpb.ps://134.171.2.248:31381/
  access: true
```

8.5.7 Closing SysSup Shell

Type Ctrl-d.

```
supSh>
bye!
```

8.6 Stopping the Software

You can use the startup/shutdown script provide to stop the whole software. For more details please check the general Getting Started guide here[8]

For stopping only the System Supervisor, the nomad CLI can be used.

```
$ nomad stop syssup
```

[8] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/guide.html