SRstackware® Application Programming Interface

Developer Guide

P/N: 6806800N90F

April 2020

Trademarks

The stylized "S" and "SMART" is a registered trademark of SMART Modular Technologies, Inc. and “SMART Embedded Computing” and the SMART Embedded Computing logo are trademarks of SMART Modular Technologies, Inc. All other names and logos referred to are trade names, trademarks, or registered trademarks of their respective owners. These materials are provided by SMART Embedded Computing as a service to its customers and may be used for informational purposes only.

Disclaimer*

SMART Embedded Computing (SMART EC) assumes no responsibility for errors or omissions in these materials. **These materials are provided "AS IS" without warranty of any kind, either expressed or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.** SMART EC further does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SMART EC shall not be liable for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials. SMART EC may make changes to these materials, or to the products described therein, at any time without notice. SMART EC makes no commitment to update the information contained within these materials.

Electronic versions of this material may be read online, downloaded for personal use, or referenced in another document as a URL to a SMART EC website. The text itself may not be published commercially in print or electronic form, edited, translated, or otherwise altered without the permission of SMART EC.

It is possible that this publication may contain reference to or information about SMART EC products, programming, or services that are not available in your country. Such references or information must not be construed to mean that SMART EC intends to announce such SMART EC products, programming, or services in your country.

Limited and Restricted Rights Legend

If the documentation contained herein is supplied, directly or indirectly, to the U.S. Government, the following notice shall apply unless otherwise agreed to in writing by SMART Embedded Computing.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data clause at DFARS 252.227-7013 (Nov. 1995) and of the Rights in Noncommercial Computer Software and Documentation clause at DFARS 252.227-7014 (Jun. 1995).

SMART Embedded Computing™, Inc.

2900 S. Diablo Way, Suite 190

Tempe, Arizona 85282

USA

*For full legal terms and conditions, visit *www.smartembedded.com/ec/legal*

Table of Contents

Table of Contents

Table of Contents

List of Tables

List of Tables

About this Manual

Overview of Contents

This manual is divided into the following chapters and appendices:

Abbreviations

This document uses the following abbreviations:

Abbreviation	Definition
API	Application Programming Interface
CLI	Command Line Interface
SNMP	Simple Network Management Protocol
VLAN	Virtual LAN
VRRP	Virtual Router Redundancy Protocol

Conventions

The following table describes the conventions used throughout this manual.

Notation	Description
0x00000000	Typical notation for hexadecimal numbers (digits are 0 through F), for example used for addresses and offsets
0b0000	Same for binary numbers (digits are 0 and 1)
bold	Used to emphasize a word

Notation	Description
Screen	Used for on-screen output and code related elements or commands. `Sample of Programming used in a table (9pt)`
`Courier + Bold`	Used to characterize user input and to separate it from system output
Reference	Used for references and for table and figure descriptions
File > Exit	Notation for selecting a submenu
<text>	Notation for variables and keys
[text]	Notation for software buttons to click on the screen and parameter description
...	Repeated item for example node 1, node 2, ..., node 12
. . .	Omission of information from example/command that is not necessary at the time
..	Ranges, for example: 0..4 means one of the integers 0,1,2,3, and 4 (used in registers)
|	Logical OR
	Indicates a hazardous situation which, if not avoided, could result in death or serious injury
	Indicates a hazardous situation which, if not avoided, may result in minor or moderate injury
	Indicates a property damage message
	Indicates a hot surface that could result in moderate or serious injury

Notation	Description
	Indicates an electrical situation that could result in moderate injury or death
	Indicates that when working in an ESD environment care should be taken to use proper ESD practices
	No danger encountered, pay attention to important information

Summary of Changes

This manual has been revised and replaces all prior editions.

Part Number	Publication Date	Description
6806800N90F	March 2020	Rebranded to SMART Embedded Computing. Cleaned up abbreviations table.
6806800N90E	July 2017	Added registered trademark symbols.
6806800N90D	December 2014	Updated *Deliverables on page 13* and *Compiling the Application with SRS-API on page 17*.
6806800N90C	June 2014	Added *srs_api_exec_file_command on page 25*. Updated *srs_api_stop_listening on page 28*. Rebranded to Artesyn template.
6806800N90B	October 2012	Added notes in the document regarding the LAYER3SRS license.
6806800N90A	February 2012	EA Release

Introduction

1.1 Overview

SRstackware Application Programming Interface (API) provides the client applications an interface to work with SRstackware programmatically. This enables the client applications to configure any of the SRstackware features at run-time

The client application may be running on a remote device.

The SRstackware API enables the application to:

- Read current configuration of the system
- Configure SRstackware and make it persistent
- Receive asynchronous events from SRstackware for the registered events

The SRstackware API provides all the above mentioned features through few API functions. These API functions take the CLI commands as their arguments to simplify the integration of client application with SRstackware.

If the client application is to be run remotely, remote-client support of SRS-API should be enabled on the blade. For this, `REMOTE_SRS_API` should be set to `YES in /etc/opt/srstackware/config/srsinit.conf.` Then the blade is to be rebooted for the change to take effect. By default, SRS-API does not support remote clients i.e., `REMOTE_SRS_API is set to NO`.

1.2 Deliverables

The deliverables are as follows:

- SRstackware API is packaged in `srs-enhanced-BUILDx.ppc.rpm, where BUILDx` is the SRS build. Libraries:
  - `srslib.a, libsrsfp.so, and libsrsl3.so` for PPC
  - `srslib_x86.a` for x86 32bit and `srslib_64bit_x86.a` for x86 64-bit architecture
  - `srs_api.h` header file

  These are packaged in this RPM and installed on the blade at `/opt/srstackware/lib/` and `/usr/share/srstackware/include/` directories respectively. The application can include this library and header file to use SRS-API. The

later sections of this document explain the procedure to integrate client application with SRS-API.

- A configuration file `srsinit.conf`, to enable or disable the support for remote clients in SRS-API. The file will be installed on the blade at `/etc/opt/srstackware/config` directory.
- A demo application using various features of SRstackware API is packaged insrs-enhanced RPM . The source code and binary will be installed on the blade at `/usr/share/srstackware/demo_apps/api_app/` and `/opt/srstackware/bin/` directories respectively.

Features

2.1 Overview

SRstackware provides the following features:

- For configuration, the SRS provides an API to configure the functionality. The existing CLI commands can be given as a parameter to the config API function. The SRstackware API function returns a `SRS_SUCCESS` or `SRS_FAILURE`. The SRstackware ensures that this configuration is persistent once the application saves it by issuing write file command through API.
- To read the configuration, the SRstackware provides an API to pass the `show` commands of CLI as a parameter to the API function. This API function returns a buffer which contains the output of the show command.
- The SRstackware provides an API function to application to register a callback function for desired events. The SRS library calls the callback function with the event as parameter along with necessary information when the event occurs.
- The SRstackware APIs can be invoked by a client application running on a remote device, if the support for remote client is enabled on the board.

Integration

3.1 Overview

This chapter explains the procedure to integrate a client application with SRstackware Application Programming Interface (API). The section *Compiling the Application with SRS-API* describes the compilation procedure after integrating the application with SRS-API. The sections *Configuring through SRS-API*, *Getting Configuration/State of the System through SRS-API*, *Registering for Asynchronous Events through SRS-API*, and *Deinitializing with SRS-API* creates the routines to be invoked while using the features of SRstackware API.

3.2 Compiling the Application with SRS-API

The application has to link few libraries in order to compile. Apart from SRS-API library (srslib.a), the following libraries have to be linked:

- srsfp (required for compiling locally)
- srsl3 (required for compiling locally)
- curses
- crypt

The application has to include SRS-API header file (srs_api.h) in their Source files to access all the variables and routines mentioned in the later sections.

The command to compile the application for PPC architecture can be as shown:

```
powerpc-wrs-linux-gnu-ppc_e500v2-glibc_cgl-gcc <application-
source-file> -lcurses -lcrypt -lsrsfp -lsrsl3 -L. srslib.a -o
<application_output_file>
```

If compilation fails, add /opt/srstackware/lib folder to standard library path.

The command to compile the x86 application can be as given as shown:

```
i586-wrs-linux-gnu-i686-glibc_std-gcc <applicationsource-file> -
lcurses -lcrypt -lpthread -L. srslib_x86.a -o
<application_output_file>
```

3.3 Configuring through SRS-API

1. Allocate SRS global structure using `srs_api_globals_alloc()`. This returns a pointer to structure `srs_api_globals` which should be used while invoking various SRS library routines below.
2. If the client application is running remotely, then the IP address of the target device need to be set in the `srs_ip_address` member of the `srs_api_globals` structure. If `srs_ip_address` is not mentioned, then the target device is considered as a local device. This IP address is used by SRS-API routines to connect to the target machine .

If the support for remote client is not enabled on the device and a target srs_ip_address is mentioned, then the SRS-API routines return failure as the connection between SRS and the remote client application is not established.

3. Initialize and lock the configuration mode using the function `srs_api_config_init()`. Once the application calls this function, the SRS `no..` commands will not work for all the other users including CLI configuration mode.
4. Pass the desired CLI command and CLI mode using the API function `srs_api_exec_command()`.

   This function returns `SRS_API_SUCCESS` or `SRS_API_FAILURE` depending on whether the given CLI command execution is a success or failure. In case of failure, the failure string is passed back to the application as additional information. It is not recommended to parse or take action based on this additional information.

The CLI command that is being passed should be licensed. Please check if command is licensed before issuing the command. If the command is not licensed, an SRS_API_FAILURE is returned. And the failure string will be "not licensed".

5. After the desired configuration is done, deinitialise and unlock the configuration mode to other users using the function `srs_api_config_deinit()`. If the configuration mode need not be unlocked for any other users, this step is optional. This must be invoked if the application is exiting.

3.4 Getting Configuration/State of the System through SRS-API

1. Allocate SRS global structure using `srs_api_globals_alloc()`. This returns a pointer to structure `srs_api_globals` which should be used while invoking various SRS library routines below.

2. If the client application is running remotely, then the IP address of the target device need to be set in the srs_ip_address member of the srs_api_globals structure. If srs_ip_address is not mentioned, then the target device is considered as a local device. This IP address is used by SRS-API routines to connect to the target machine.

If the support for remote client is not enabled on the device and a target srs_ip_address is mentioned, then the SRS-API routines return failure as the connection between SRS and the remote client application is not established.

3. The application can execute show commands only after initializing SRS-API, by invoking `srs_api_show_init()`.
4. Pass the desired CLI show commands and the CLI mode using the API function `srs_api_exec_command()`. This function returns `SRS_API_SUCCESS` or `SRS_API_FAILURE` depending on whether the CLI command is success or failure.

   On success, the output of the show command is returned as part of this function. We do not recommend to parse or take action based on the output of these commands except for `show running-config` <> set of commands. In case of failure, the failure string is passed back to the application as additional information. SRstackware do not recommend parsing or taking action based on this additional information.

The CLI command that is being passed should be licensed. Please check if command is licensed before issuing the command.

5. After the desired show commands are issued, deinitialise using the function `srs_api_show_deinit()`. This must be invoked if the application is exiting if it has initialized.

This is separated from configuration steps to enable you to lock/unlock the configuration of the system for other users.

3.5 Registering for Asynchronous Events through SRS-API

1. Allocate SRS global structure using `srs_api_globals_alloc()`. This returns a pointer to structure `srs_api_globals` which should be used while invoking various SRS library routines below.

2. If the client application is running remotely, then the IP address of the target device need to be set in the srs_ip_address member of the srs_api_globals structure. If srs_ip_address is not mentioned, then the target device is considered as a local device. This IP address is used by SRS-API routines to connect to the target machine.

If the support for remote client is not enabled on the device and a target srs_ip_address is mentioned, then the SRS-API routines return failure as the connection between SRS and the remote client application is not established.

3. The application can register for asynchronous events only after initializing SRS-API, by invoking `srs_api_show_init()`.
4. For the applications to receive the asynchronous events, it must register itself with SRstackware by providing the callback functions and the events for which they need to be called. This is done be calling the library function `srs_api_set_callback()`. The prototype of the call back function is as follows:

   ```
   int (*f)(int event, void *srs_data);
   ```

5. Create a thread (recommended) and call the function `srs_api_start_listening()`. This is a blocking function and the callback functions will be executed under this context. This may return `SRS_API_FAILURE` in case it was invoked without proper initialization.
6. SRS-API invokes the registered callback with the event and the data. The data should be type-casted depending on the event and the application can take decision based on this data.

The callback function is called when the event occurs. Currently the following events are supported.

- Interface DOWN
- Interface UP
- Port error notificationPaired-link status notification

For the Interface UP and Interface DOWN events, the application has to typecast `void*` to `structure srs_interface*`.

For port error notification, typecast `void*` to `structure srs_port_error_info*`.

For Paired-link status notification, typecast `void*` to `structure srs_paired_link_event_info*`.

Refer to `srs_api.h` for the structure definitions.

3.6 Deinitializing with SRS-API

If the client application is exiting, then it has to perform all of the following based on the initializations. Otherwise, if the application does not want to use a specific feature of an SRS API, then it has to deinitialize appropriately as mentioned below.

- If the application does not want to receive any of the asynchronous events, then it can stop the listening thread by calling the function `srs_api_stop_listening()`.
- If the application does not need to execute any configuration commands, then it can call the function `srs_api_config_deinit()`.
- If the application does not need to execute any show commands, then it can call the function `srs_api_show_deinit()`.
- Call the function `srs_api_globals_free()` to free the globals structure allocated.

NOTE: If `srs_api_show_deinit()` is not called and if the application wants to listen for asynchronous events again, then it can start from *Step 5.* in *Registering for Asynchronous Events through SRS-API on page 19*.

Application Programming Interfaces

4.1 srs_api_globals_alloc

This API allocates memory for `srs_api_globals` structure and returns the pointer for it.

4.1.1 API Call

```
struct srs_api_globals * srs_api_globals_alloc();
```

4.1.2 Input Parameters

None

4.1.3 Output Parameters

None

4.1.4 Return Values

Pointer to structure `srs_api_globals` or NULL upon failure.

4.2 srs_api_config_init

This API initializes the variables of `srs_api_globals` that are required for executing various configuration commands. The pointer returned by `srs_api_globals_alloc` should be passed to this routine. Once the application calls this function, the SRS no.. commands will not work for all the other users including CLI configuration mode.

4.2.1 API Call

```
int srs_api_config_init(struct srs_api_globals *app_data)
```

4.2.2 Input Parameters

`app_data` - Pointer to `srs_api_globals` returned by `srs_api_globals_alloc`

4.2.3 Output Parameters

None

4.2.4 Return Values

Returns SRS_API_FAILURE upon failure, while initializing srs_api_globals or SRS_API_SUCCESS upon success.

4.3 srs_api_show_init

This API initializes the variables of srs_api_globals which are useful for executing show commands with SRstackware. The pointer returned by srs_api_globals_alloc should be passed to this routine.

4.3.1 API Call

```
int srs_api_show_init(struct srs_api_globals *app_data)
```

4.3.2 Input Parameters

app_data - Pointer to srs_api_globals returned by srs_api_globals_alloc.

4.3.3 Output Parameters

None

4.3.4 Return Values

Returns SRS_API_FAILURE upon failure while initializing srs_api_globals or SRS_API_SUCCESS upon success.

4.4 srs_api_exec_command

This API is used to execute any SRS CLI command programmatically through SRS-API. This CLI command could be any configuration command or show command.

4.4.1 API Call

```
int srs_api_exec_command(struct srs_api_globals *app_data, char *cmd_str,
int mode);
```

4.4.2 Input Parameters

app_data - Pointer to srs_api_globals returned by srs_api_globals_alloc.

cmd_str - CLI command to be executed.

mode - The mode in which the CLI command should be executed (Refer *Appendix A, Supported Command Modes* for supported modes and their values).

4.4.3 Output Parameters

The output of the command is stored in a character variable output_string of srs_api_globals (app_data). output_string will be filled in the following two scenarios:

Upon failure of configuration command

Output of show command if show command is passed in cmd_str

4.4.4 Return Values

SRS_API_FAILURE

SRS_API_SUCCESS

4.5 srs_api_exec_file_command

This API is used to programmatically load new configuration file or add/modify system configuration using configuration file through SRS-API. The CLI commands in the file should be only configuration command and no show commands.

4.5.1 API Call

```
int srs_api_exec_file_command(struct srs_api_globals *app_data, char
*file);
```

4.5.2 Input Parameters

app_data - Pointer to srs_api_globals returned by srs_api_globals_alloc.

file - Configuration file with complete path reference.

4.5.3 Output Parameters

0: Success

SRS_API_FAILURE: In case of failure in system related operations. For example, if the input file is invalid.

SRS_API_FILE_ERROR: In case of error, while applying configuration mentioned in a file.

Further configuration will not be applied.

In case of failure, the output of the failed command can be fetched from output_string.

srs_api_globals (app_data). output_string will be filled with failed CLI string and line number.

4.5.4 Limitations

While Applying the configuration through file:

1. Follow steps below while calling this API:
   - Stop listening on the SRS asynchronous events using srs_api_stop_listening( ) API.
   - Call srs_api_exec_file_command ( ) API
   - Start listening on the SRS asynchronous events using srs_api_start_listening( ) API.
2. There should not be other instance in config-mode.
3. On configuration failure, error message will be printed to terminal and /var/log/srslib.log file in the format below and further configuration will not be applied.

   Tue Feb 18 10:55:47 2014:410808000:Failed to configure:static-channel-group 1:LINE 4
4. In case of failure return from this API, user has to clean the partial applied configuration and reload the configuration with appropriate correction in the configuration file.

4.6 srs_api_set_callback

This API is used to register for asynchronous events with SRS-API.

4.6.1 API Call

```
int srs_api_set_callback(int event, int (*cb_func)(int, void*));
```

4.6.2 Input Parameters

event - The event-id for which the application is registering for.

cb_func - The callback function that SRS-API has to invoke when the event occurs.

4.6.3 Output Parameters

None

4.6.4 Return Values

SRS_API_FAILURE - If srs_api_globals is not initialized properly or if the arguments passed are invalid.

SRS_API_SUCCESS

4.7 srs_api_start_listening

This enables the application to listen for the registered asynchronous events. As this is a blocking function and it is recommended to create a thread so that the callback functions will be executed under this context.

4.7.1 API Call

```
int srs_api_start_listening()
```

4.7.2 Input Parameters

None

4.7.3 Output Parameters

None

4.7.4 Return Values

SRS_API_FAILURE - If srs_api_globals is not initialized properly.

SRS_API_SUCCESS

4.8 srs_api_stop_listening

This API stops the listening thread, which is waiting for asynchronous events.

4.8.1 API Call

```
int srs_api_stop_listening()
```

4.8.2 Input Parameters

None

4.8.3 Output Parameters

None

4.8.4 Return values

SRS_API_SUCCESS

SRS_API_FAILURE - If the thread is not stopped successfully

4.9 srs_api_config_deinit

This API de-initializes the variables initialized in `srs_lib_globals` as part of `srs_api_config_init()`.

4.9.1 API Call

```
int srs_api_config_deinit(struct srs_api_globals * app_data)
```

4.9.2 Input Parameters

`app_data` - Pointer to `srs_api_globals` returned by `srs_api_globals_alloc`.

4.9.3 Output Parameters

None

4.9.4 Return Values

SRS_API_FAILURE - if the pointer to `srs_api_globals` is not proper.

SRS_API_SUCCESS

4.10 srs_api_show_deinit

This API deinitializes the variables initialized in `srs_lib_globals` as part of `srs_api_show_init()`.

4.10.1 API Call

```
int srs_api_show_deinit(struct srs_api_globals * app_data)
```

4.10.2 Input Parameters

`app_data` - Pointer to `srs_api_globals` returned by `srs_api_globals_alloc`.

4.10.3 Output Parameters

None

4.10.4 Return Values

SRS_API_FAILURE - if the pointer to `srs_api_globals` is not proper.

SRS_API_SUCCESS

4.11 srs_api_globals_free

This API is used to free `srs_lib_globals` that was allocated when `srs_api_globals_alloc()` was called.

4.11.1 API Call

```
void srs_api_globals_free(struct srs_api_globals * app_data)
```

4.11.2 Input Parameters

`app_data` - Pointer to `srs_api_globals` returned by `srs_api_globals_alloc`

4.11.3 Output Parameters

None

4.11.4 Return Values

None

Demo Application

5.1 Overview

Demo application is packaged with SRS-Enhanced RPM. Refer *Deliverables on page 13*, for the location details.

The executable of demo application (api_demo) is at `/opt/srstackware/bin`.

You may execute it as any other executable. The command syntax would be as mentioned below:

```
./api_demo <Target ip_address>
```

Target ip_address: The IP address of the device where SRstackware is running.

If no IP address is mentioned as a parameter, then the application considers that the SRstackware is running on the local device.

The output of the packaged demo application is as given below:

```
Current Command: vlan database
Command Success
Current Command: vlan 15 bridge 1 state enable
Command Success
Current Command: interface ge10
Command Success
Current Command: no shutdown
Command Success
Current Command: bridge 3 protocol mstp
Command Success
Current Command: spanning-tree mst configuration
Command Success
Current Command: bridge 3 region Region1
Command Success
Current Command: router rip
Command Success
Current Command: version 1
Command Success
Current Command: router ipv6 rip
Command Success
Current Command: timers basic 20 150 100
Command Success
```

```
Current Command: router ospf
Command Success
Current Command: default-metric 10
Command Success
Current Command: interface ge17
Command Success
Current Command: no switchport
Link-down message(asynchronous) is received by the application
Interface that went down: ge17
Ifindex of the interface went down: 5017
Bandwidth of the interface went down: 125000000.000000
Link-down message(asynchronous) is received by the application
Interface that went down: ge17
Ifindex of the interface went down: 12
Bandwidth of the interface went down: 125000000.000000
Command Success
Current Command: ip address 192.168.1.1/24
Link-down message(asynchronous) is received by the application
Interface that went down: vlan1.22
Ifindex of the interface went down: 8
Bandwidth of the interface went down: 0.000000
Command Success
Current Command: no shutdown
Link-down message(asynchronous) is received by the application
Interface that went down: vlan1.22
Ifindex of the interface went down: 8
Bandwidth of the interface went down: 0.000000
Command Success
Current Command: router vrrp 10 ge17
Command Success
Current Command: virtual-ip 192.168.2.1 master
Command Success
Current Command: mls qos 1 0 2 1 3 2 4 3 5 4 6 5 7 6 0 7
Command Success
Current Command: ip-access-list 1 deny 192.168.5.1
Command Success
```

```
Current Command: class-map cmap1
Command Success
Current Command: match access-group 1
Command Success
Current Command: policy-map pmap1
Command Success
Current Command: class cmap1
Command Success
Current Command: show interface ge10
Command Success
Result of the command:
Interface ge10
  Hardware is Ethernet
  Current HW addr: 00c3.208f.000c
  Physical:00c3.208f.000c
  Description: BC12 - Slot 12
  index 5010 metric 1 mtu 1500 duplex-full arp ageing timeout 0
  <UP,BROADCAST,RUNNING,MULTICAST>
  VRF Binding: Not bound
  Bandwidth 1g
  VRRP Master of :  VRRP is not configured on this interface.
    input packets 00, bytes 00, dropped 00, multicast packets 03
   output packets 016, bytes 05134, multicast packets 03 broadcast packets
00

Current Command: show running-config interface ge10
Command Success
Result of the command:
!
interface ge10
 description BC12 - Slot 12
 bridge-group 1 spanning-tree disable
 switchport mode hybrid
 switchport mode hybrid ingress-filter disable
 switchport hybrid vlan 21
 switchport mode hybrid acceptable-frame-type all
```

```
 switchport hybrid allowed vlan add 21 egress-tagged disable
 no shutdown
!
Current Command: show vlan brief
Command Success
Result of the command:
                                  Bridge Group : 3

Bridge      VLAN ID  Name                 State  Member ports
                                                                    (u)-
Untagged, (t)-Tagged
=============== ======= ================================ =======
================================
3               1       default                          ACTIVE

                                  Bridge Group : 2

2               1       default                          ACTIVE  xe9(t)
2               11      VLAN11                         ACTIVE  ge25(u)
                                                         ge26(u) ge27(u)
                                                               ge28(u)
                                                      xe5(u) xe6(u) xe7(u)
                                                               xe8(u)
                                                    xe9(t) xe10(u) xe11(u)
                                                   xe12(u) xe13(u) xe14(u)
                                                   xe15(u) xe16(u) xe17(u)
                                                   xe18(u) xe19(u) xe20(u)
                                                   xe21(u) xe22(u) xe23(u)
                                                   xe24(u) xe25(u) xe26(u)
                                                          xe27(u) xe28(u)
2               12      VLAN12                           ACTIVE  xe9(t)
2               91      VLAN91                         ACTIVE  xe15(u)
                                                     xe16(u) xe17(u)
                                                   xe18(u) xe21(u) xe22(u)
                                                          xe23(u) xe24(u)

                                  Bridge Group : 1
```

```
1               1       default                         ACTIVE
1               15      VLAN0015                        ACTIVE
1               21      VLAN21                          ACTIVE  ge1(u)
                                                        ge2(u) ge3(u) ge4(u)
                                                        ge5(u)
                                                        ge6(u) ge7(u) ge8(u)
                                                                ge9(u)
                                                      ge10(u) ge11(u) ge12(u)
                                                      ge13(u) ge14(u) ge18(u)
                                                      ge19(u) ge20(u) ge21(u)
                                                                ge22(u)
                                                       ge23(u) ge24(u) xe1(u)
                                                         xe2(u) xe3(u) xe4(u)
1               22      VLAN22                          ACTIVE
1               24      VLAN24                        ACTIVE  ge15(u) ge16(u)
1               93      VLAN93                         ACTIVE  ge21(u)
                                                        ge22(u) ge23(u)
                                                                ge24(u)
                                                        xe1(u) xe2(u) xe3(u)
                                                        xe4(u)


Current Command: show vrrp ipv6 1 ge20
Command failed
Error Message: % The VRRP session does not exist

Current Command: show vlan
Command failed
Error Message: Improper command

Link-down message(asynchronous) is received by the application
Interface that went down: ge6
Ifindex of the interface went down: 5006
Bandwidth of the interface went down: 0.000000
Link-up message(asynchronous) is received by the application
Interface that went up: ge17
```

```
Ifindex of the interface went up: 12
Bandwidth of the interface went up: 125000000.000000
Link-up message(asynchronous) is received by the application
Interface that went up: ge6
Ifindex of the interface went up: 5006
Bandwidth of the interface went up: 12500000.000000
Link-down message(asynchronous) is received by the application
Interface that went down: ge6
Ifindex of the interface went down: 5006
Bandwidth of the interface went down: 0.000000
Link-up message(asynchronous) is received by the application
Interface that went up: ge6
Ifindex of the interface went up: 5006
Bandwidth of the interface went up: 12500000.000000
Link-down message(asynchronous) is received by the application
Interface that went down: ge6
Ifindex of the interface went down: 5006
Bandwidth of the interface went down: 0.000000
Link-up message(asynchronous) is received by the application
Interface that went up: ge6
Ifindex of the interface went up: 5006
Bandwidth of the interface went up: 12500000.000000
Port error threshold message(asynchronous) is received by the application
with error_code: 1
Interface that crossed threshold: ge1
Ifindex of the interface caused error: 5001
error type : 1 crossed threshold
Configured port threshold :100, shutdown flag:1
```

Supported Command Modes

A.1 Overview

The following are the modes supported by SRS-API. All the SRstackware CLI commands are registered under some mode. While invoking `srs_api_exec_command`, the application has to pass mode-id in which the CLI command needs to be executed.

All the modes relating to Routing are not supported, if LAYER3SRS license is not available on the blade.

Table A-1 Supported Command Modes

Supported Modes	Mode-id
EXEC_MODE	4
CONFIG_MODE	5
INTERFACE_MODE	16
ROUTER_MODE	26
RIP_MODE	33
RIPNG_MODE	35
OSPF_MODE	36
VRRP_MODE	45
IP_MODE	53
PREFIX_MODE	55
ACCESS_MODE	56
L2_ACCESS_MODE	57
IPV6_MODE	58
RMAP_MODE	62
BRIDGE_MODE	65
VLAN_MODE	66

Table A-1 Supported Command Modes

Supported Modes	Mode-id
STP_MODE	67
RSTP_MODE	68
MSTP_MODE	69
MST_CONFIG_MODE	70
GVRP_MODE	72
GMRP_MODE	73
IGMP_MODE	74
QOS_MODE	80
CMAP_MODE	81
PMAP_MODE	82
PMAPC_MODE	83
MLIST_MODE	112
LACP_CONFIG_MODE	113
MLIST_RULE_MODE	114

Related Documentation

B.1 SMART Embedded Computing Documentation

The documentation listed is referenced in this manual. Technical documentation can be found by using the Documentation Search at *https://www.smartembedded.com/ec/support/* or you can obtain electronic copies of SMART EC documentation by contacting your local sales representative.

Table B-1 SMART Embedded Computing Publications

Document Title and Source	Publication Number
SRstackware Intelligent Network Software Troubleshooting Guide	6806800N83
SRstackware Intelligent Network Software VRRP Command Reference	6806800N84
SRstackware Intelligent Network Software RIP Command Reference	6806800N85
SRstackware Intelligent Network Software Layer 2 Configuration Guide	6806800N86
SRstackware Intelligent Network Software OSPF Command Reference	6806800N87
SRstackware Intelligent Network Software Layer 2 Command Reference	6806800N88
SRstackware Intelligent Network Software Layer 3 Configuration Guide	6806800N89
SRstackware Intelligent Network Software Switch Configuration Command Reference	6806800N92
SRstackware Intelligent Network Software Layer 3 Command Reference	6806800N93
SRstackware Intelligent Network Software Protocol Demo Guide	6806800N07
SRstackware FAQ	6806800N91
SRstackware Intelligent Network Software Switch Configuration Command Reference	6806800N92

Related Documentation