# SOOBML: Switch Out-of-Band Management Library

## Requirements and Design Document

Version 2.0 – 2009.07.14

*Created 2009.07.10*

Pascal Meunier,
Purdue University CERIAS

# Table of Contents

# 1 Introduction

## 1.1 Document Scope

This document presents the design of version 2 of the soobml, which is a library supporting serial connections to a Cisco switch for the purpose of automatically managing VLANs and sending arbitrary commands to the switch. Requirements influencing the design are also discussed. The scope of the discussion of requirements is broader than is usual in a design document, because there is no separate requirements specification document at the current time. As the library is small the discussion isn't overly complicated by this bundling.

## 1.2 Intended audience

The intended audience of this document is developers or system administrators who would use the library to control Cisco switches on-the-fly, manually, with scripts or as part of more complex systems. The library is coded in Python so a basic knowledge of Python is assumed.

## 1.3 Project Overview

Cisco switches support both in-band management and out-of-band management (OOBM). Out-of-band management interfaces are not connected to the switching fabric and do not participate in any of the traffic management functions. Out-of-band management is advantageous for security reasons as well as reliability: it remains manageable even if its configuration has been mangled. A misconfiguration of the switch could open a switch to attacks on an in-band management interface, or prevent any more commands from being sent to the switch, thus disabling management. OOBM is safer and more secure, but is more difficult to automate. The only interface provided on many switches is a mere serial port.

The soobml solves the need for scripting support for true out-of-band management of Cisco switches that only have a serial port. Note that the "management port" of Cisco switches is nothing more than an emergency recovery port, and is active only while the switch is booting, in ronmon mode, and so it can't be used for our purpose. We use the library to dynamically create and destroy VLANs to contain security experiments, within the Purdue University CERIAS ReAssure project.

Version 2 of the soobml is thread-safe, object-oriented, and carefully designed and tested. It can manage several switches using different serial ports. Options can be changed on-the-fly when creating a class instance, instead of being hard-coded as in version 1.

# 2 High-Level Design

## 2.1 Goals and Guidelines

Major goals that influenced the design were the requirements for the library to:

- Make switch management operations atomic in effect. This is needed to support a multi-threaded automation architecture, and allow supervisory access while automated scripts may be running at the same time. To achieve this all accesses to the switch must be made through the library.
- Support for multiple switches each on a different serial port
- Support for administrators to issue commands and retrieve results while automated scripts may be active.

## 2.2 Architectural Strategies

Each switch and serial port are managed through a particular instance of the SwitchManager class. All communications with a particular switch are protected by a mutex. That mutex is an attribute of the relevant SwitchManager instance. Handles to all instances are stored in an associative array, protected by a global mutex. When a new instance is created, it "registers" itself in the global associative array, using the global mutex to prevent another instance from being created for the same purpose. A creation request to manage a switch for which an instance already exists would return that instance instead. The overall effect is similar to having a Singleton software engineering pattern for each serial port.

Errors generate exceptions defined from a generic exception class specific to the soobml. Optionally, events worthy of being logged can also generate exceptions, even though they are not serious errors. An example of such an event is a request for the deletion of a non-existent VLAN, or the request for the creation of an already existing VLAN. This type of unnecessary request may indicate that something is not working correctly. The library does not, and should not, include any user interface components beyond generating appropriate error messages for exceptions. The warnings should not be printed to stderr to avoid job management problems in daemon processes. However, a verbose mode of operation should print plenty of status messages to stderr to help testing and debugging.

## 2.3 High-Level Component View

The library depends on the built-in Python module PySerial, and exposes a SerialManager interface and a function to retrieve an instance managing a specific serial port. Logging is the responsibility of the calling program. Figure 1 shows how it could be used in an experimental testbed (such as the ReAssure testbed at CERIAS).

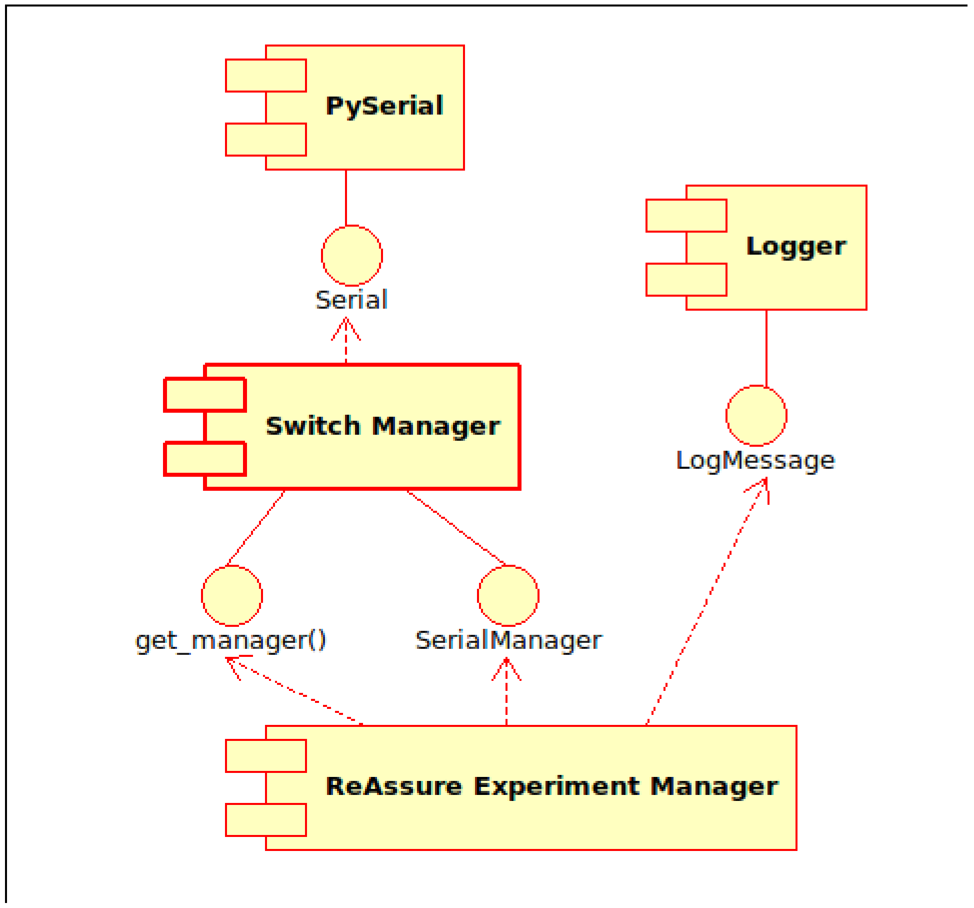


Figure 1, Example component diagram of a system using the soobml.

# 3 Detailed Design

## 3.1 Logical View

In soobml version 2, there are no special return values signifying an error; instead methods raise exceptions. There is one generic exception class for the library (SMError), which helps simplify the management of exceptions for code calling the soobml. Then there are specific exceptions for communication problems between the switch and the soobml (SMComError), for internal errors (e.g., the violation of invariants: SMInternalError), and for incorrect inputs provided to methods (SMInputError). Additionally, one exception (SMLog) reports noteworthy, non-fatal events for logging.

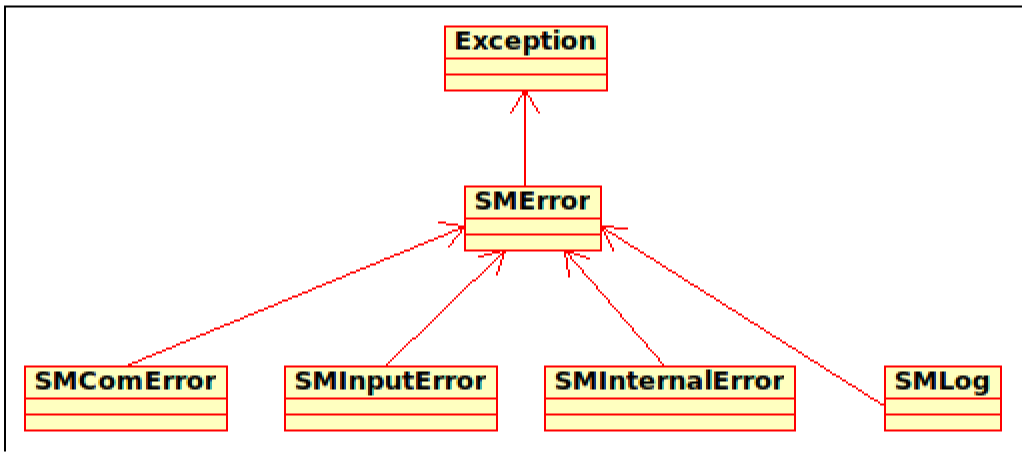
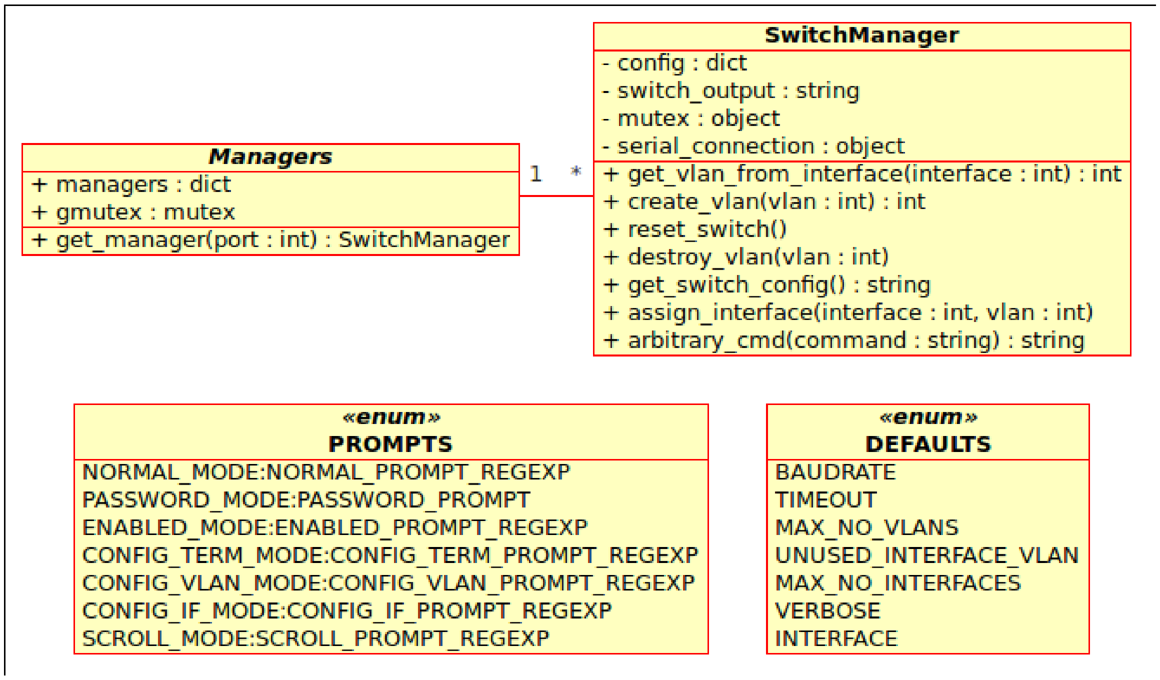


Figure 2, Exception hierarchy in the soobml.

The SwitchManager class is the heart of the library. The constructor accepts a dictionary of values; any values not specified will be filled using the DEFAULTS dictionary of constants. The abstract singleton Managers keeps track of SwitchManager instances by port number. Using a SwitchManager instance, vlans can be created and destroyed, and interfaces can be assigned to vlans. The current configuration of the switch can be retrieved and arbitrary commands given. Finally, the switch can be reset, which deletes all vlans except the management vlan.


SwitchManager
- config : dict
- switch_output : string
- mutex : object
- serial_connection : object
+ get_vlan_from_interface(interface : int) : int
+ create_vlan(vlan : int) : int
+ reset_switch()
+ destroy_vlan(vlan : int)
+ get_switch_config() : string
+ assign_interface(interface : int, vlan : int)
+ arbitrary_cmd(command : string) : string
Managers
+ managers : dict
+ gmutex : mutex
+ get_manager(port : int) : SwitchManager
1
*
«enum»
PROMPTS
NORMAL_MODE:NORMAL_PROMPT_REGEXP
PASSWORD_MODE:PASSWORD_PROMPT
ENABLED_MODE:ENABLED_PROMPT_REGEXP
CONFIG_TERM_MODE:CONFIG_TERM_PROMPT_REGEXP
CONFIG_VLAN_MODE:CONFIG_VLAN_PROMPT_REGEXP
CONFIG_IF_MODE:CONFIG_IF_PROMPT_REGEXP
SCROLL_MODE:SCROLL_PROMPT_REGEXP
«enum»
DEFAULTS
BAUDRATE
TIMEOUT
MAX_NO_VLANS
UNUSED_INTERFACE_VLAN
MAX_NO_INTERFACES
VERBOSE
INTERFACE
Figure 3, The SwitchManager, the registry and supporting constants.

## 3.2 *Process View*

The switches have several possible "prompt modes". For example, normal mode is used when the source of commands has not provided the appropriate password to access privileged commands. It is important that commands to the Cisco switches be issued to the appropriate prompt. Therefore, the SwitchManager must insure that the mode is well controlled and verified. Changing from one mode to another must be handled automatically depending on the commands given, except for arbitrary commands, which are always assumed to be executed in "enable" mode. The modes are shown as a state diagram in Fig. 4.

As a single instance of the SwitchManager must be in control of a serial port, there has to be a way to retrieve the instance appropriate for a given port and avoid the creation of multiple instances. Python modules are in effect singletons. So, the abstract class "Managers" can be implemented by a Python module. The registry of instances is a dictionary (global within the module) with an associated getter function.

By having SwitchManager instances check the registry during initialization, multiple instances for the same port can be avoided. Python actually performs initialization in two steps (new and init). As instance creation is done in "new", this is where the check for the instance should be done. Refer to Fig. 5 for the initialization steps. A program wanting to use the library should first create the instances it needs, and as needed, call the getter function to retrieve the appropriate instance. It is possible for a program wanting to use the library to avoid having to pass around a reference to the appropriate SwitchManager instance, if the correct port is known.

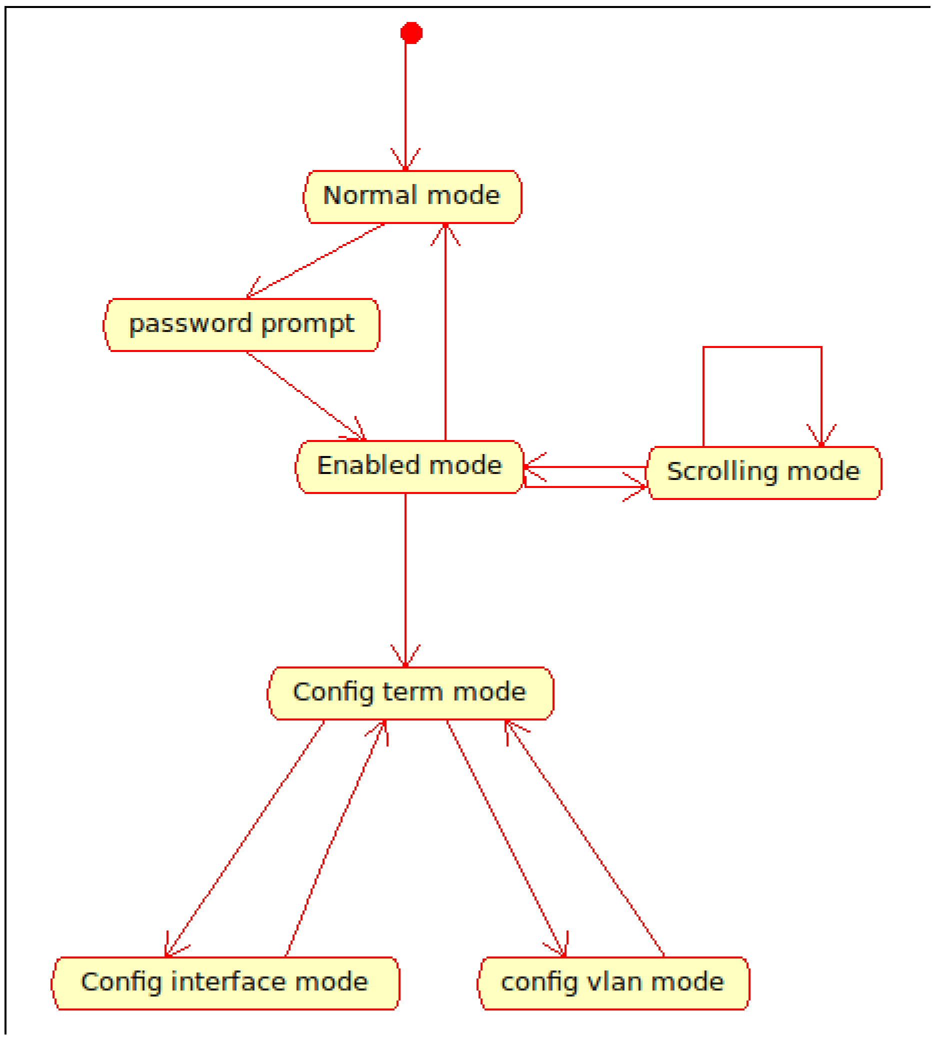


Figure 4.  The states of a Cisco switch, from the point of view of the possible prompts.

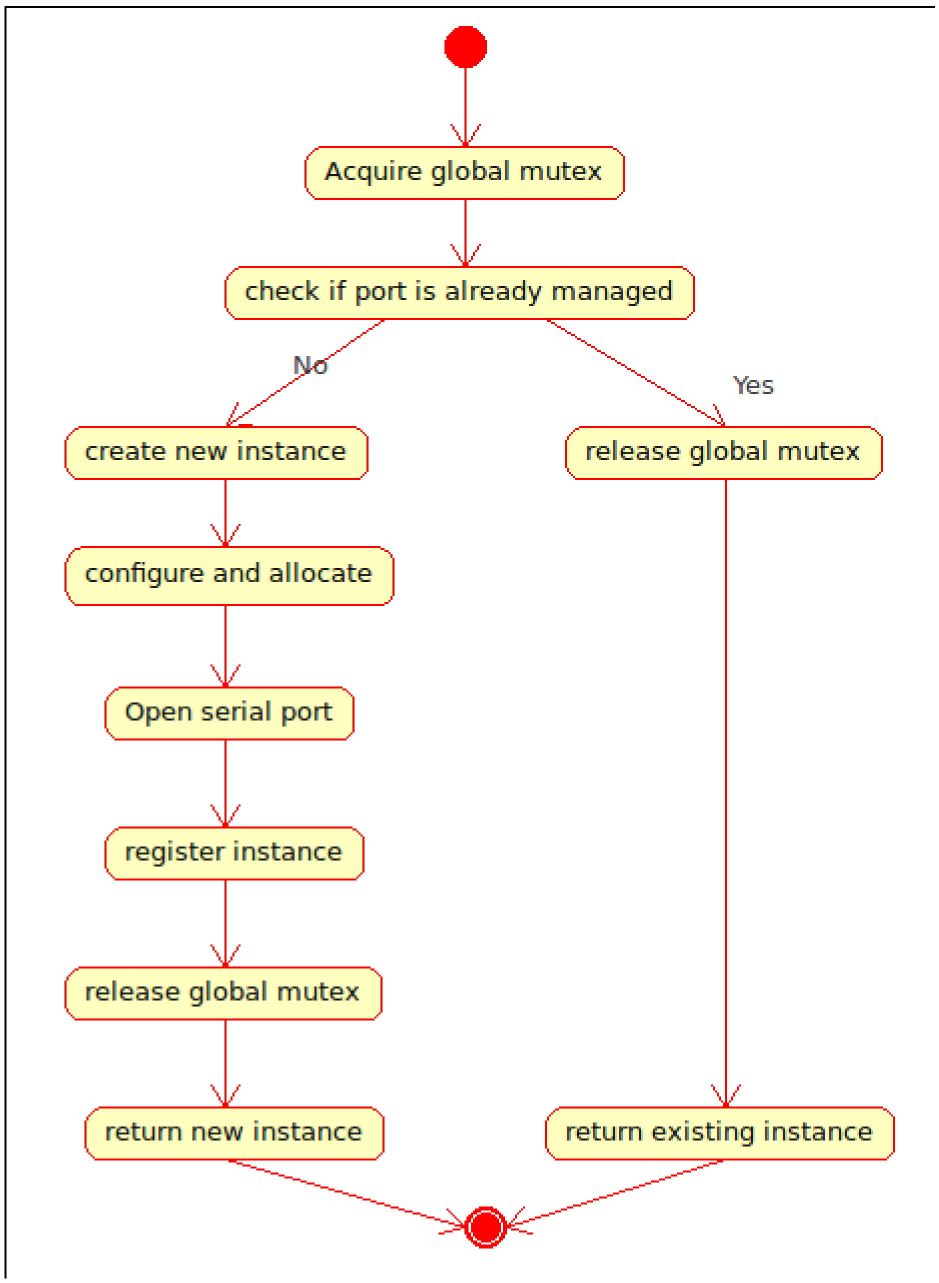


Figure 5. Initialization and registration of SwitchManager instances.

## 4 Appendix

This section is deliberately empty.

## 5 Glossary

This section is deliberately empty.

## 6 Revisions to this Document

*Version 1 of the soobml was designed by Mayank Ramkishore Gupta, Patrick Perrone and Pascal Meunier but didn't have a formal design document. This design document is numbered version 2 to make clear that it applies to version 2 of the soobml. Some of the text in this document was borrowed from the documentation for version 1 of the soobml.*