Specific Simple Network Management Tools

Jürgen Schönwälder

University of Osnabrück
Albrechtstr. 28
49069 Osnabrück, Germany

Tel.: +49 541 969 2483
Email: <schoenw@informatik.uni-osnabrueck.de>
Web: <http://www.informatik.uni-osnabrueck.de/>

What is SNMP?

- The Simple Network Management Protocol (SNMP) is used to access and manipulate typed variables organized in conceptual tables or groups of scalars.
- Each variable (either a scalar or a cell in a conceptual table) is uniquely identified by an OID value (in a given context).
- An OID value is a sequence of numbers which identify a path in a registration tree, where the nodes in each level are uniquely identified by numbers.
- SNMP operates on an ordered list (varbind list) of variables (lexicographic ordering).
- Each element of a varbind list contains an OID value identifying a variable and its value.
- SNMP usually runs over UDP (stateless, retransmision control).

Generic vs. Specific SNMP Tools

- What we have today:
  1. Generic low-level SNMP Tools (snmpget, snmpwalk, ...)
  2. Generic low-level SNMP APIs (WinSnmp, SNMP++, Tnm, ...)
  3. Generic MIB Browsers (...)
  4. Generic Monitoring Tools (mrtg, ...)
  5. Generic Management Platforms (OpenView, Spectrum, ...)
- What is really needed:
  - Specific SNMP Tools that focus on doing one thing and doing it right.

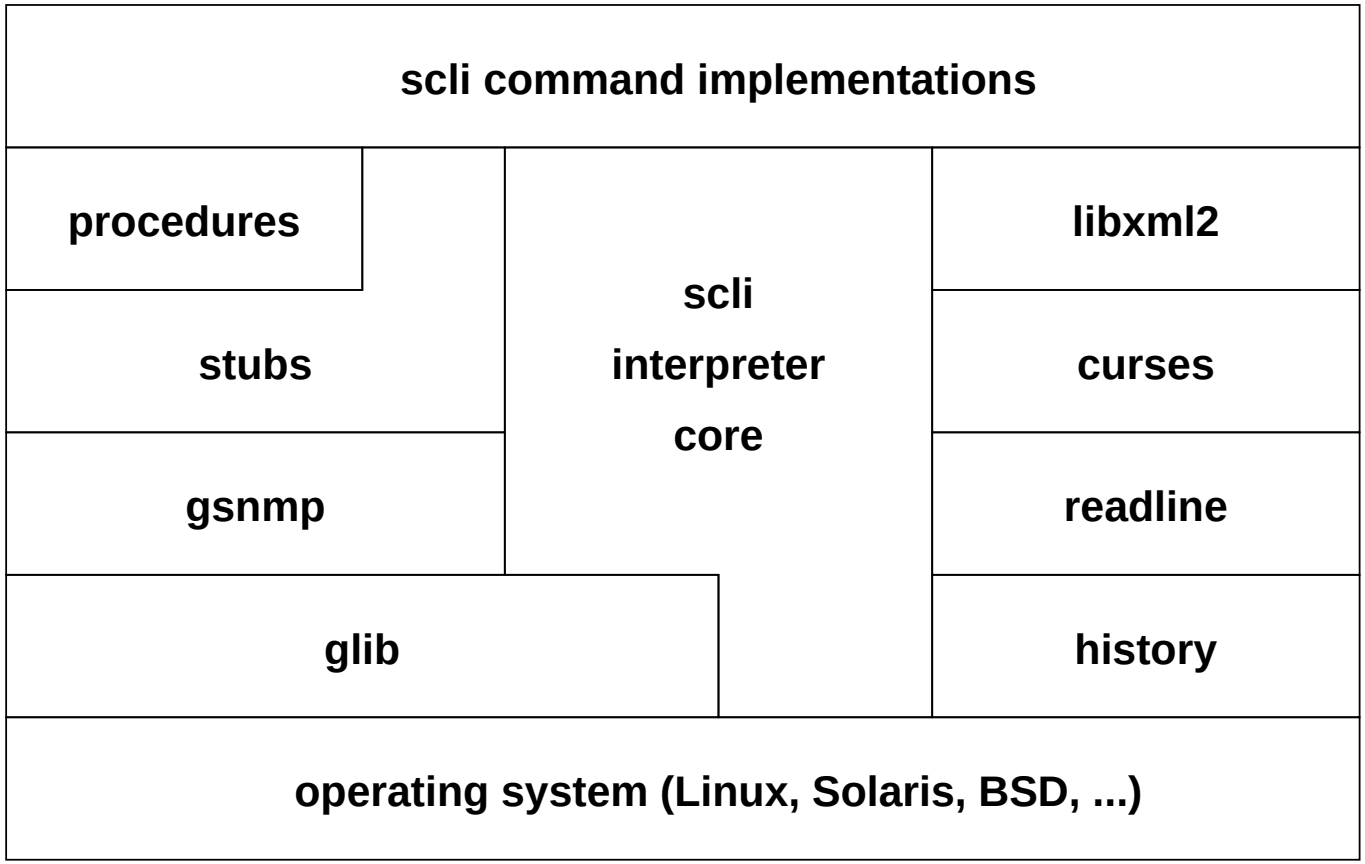
SNMP Command Line Interface (scli)

- Command line interface that works on devices produced by different vendors.
- Commands are structured in a hierarchy, supports recursive command evaluation.
- Runs locally and communicates with the device using standard SNMP interactions.
- Displays information in formats that make sense for humans.
- Support for simple online monitoring activities.
- Selecting objects using names and regular expressions.
- Support of readline and history mechanisms and command aliases.
- Scripting capabilities by using an easy to parse XML output format.
- Simple APIs to motivate people to write and contribute extensions.

Software Design Requirements

- Extensibility:
  - Make it relatively easy to add new features (for the average programmer).
  - Hide low-level SNMP communication details as much as possible.
- Robustness:
  - Ensure that errors are detected and handled gracefully where possible.
  - Abort as soon as possible if coders forget to check for error conditions.
- Maintainability:
  - The software will evolve over time, which includes internal API changes.
  - Ensure that the documentation is available and in sync with the implementation.
- Efficiency:
  - Fast startup times so that the tools can be used efficiently in scripts.
- Portability:
  - The tools should run on all major Unix systems.
  - A port to Win32 platforms should be made possible at reasonable costs.

Software Architecture



Interface Mode

```
set interface status <regexp> <status>
set interface alias <regexp> <string>
set interface notifications <regexp> <value>
set interface promiscuous <regexp> <bool>

show interface info [<regexp>]
show interface details [<regexp>]
show interface stack [<regexp>]
show interface stats [<regexp>]

monitor interface stats [<regexp>]

dump interface
```

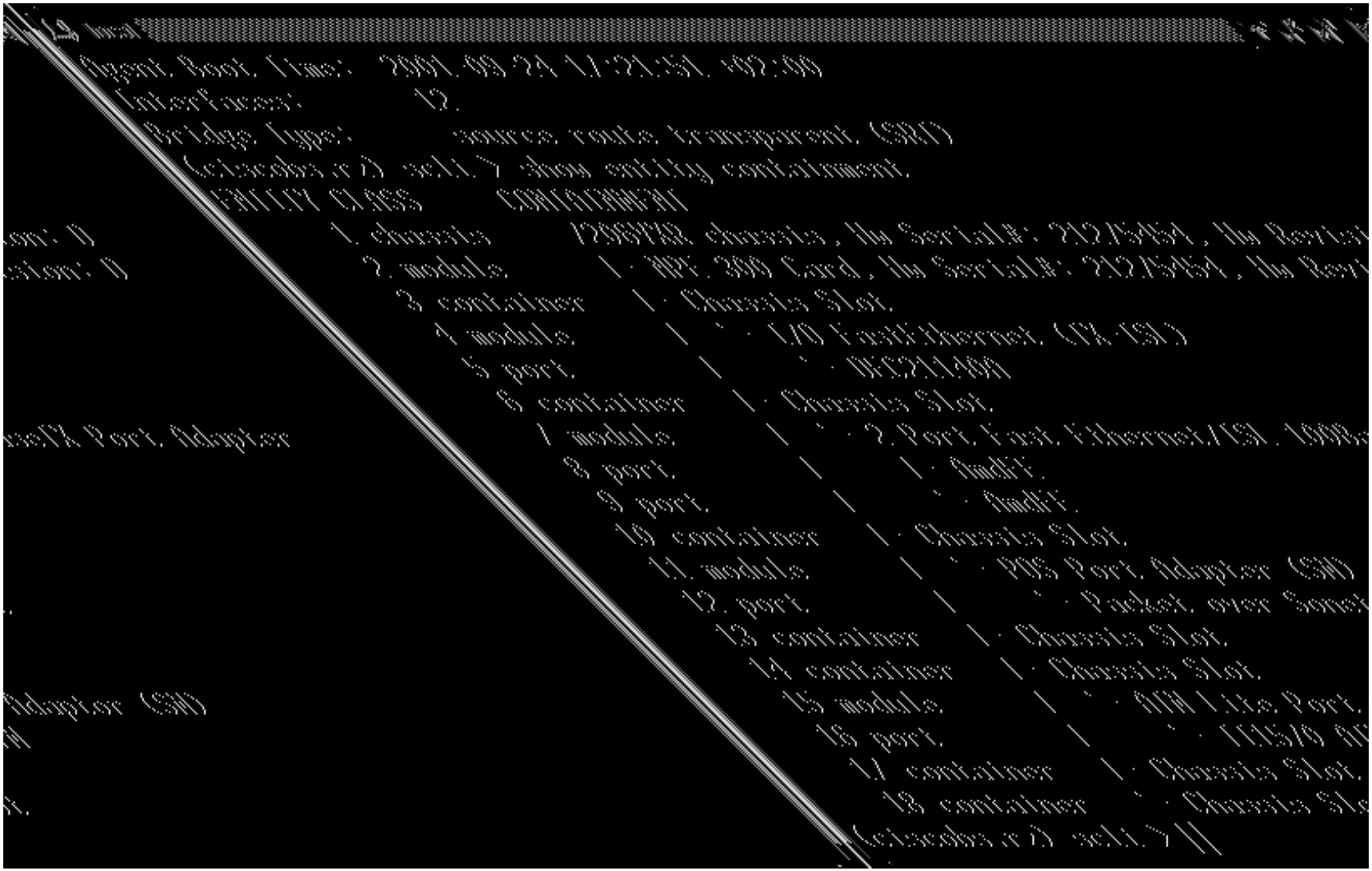
- Provides commands to configure, display and monitor network interfaces.
- Interfaces are identified by regular expressions matched against names.

Configuring Virtual LANs (VLANs)

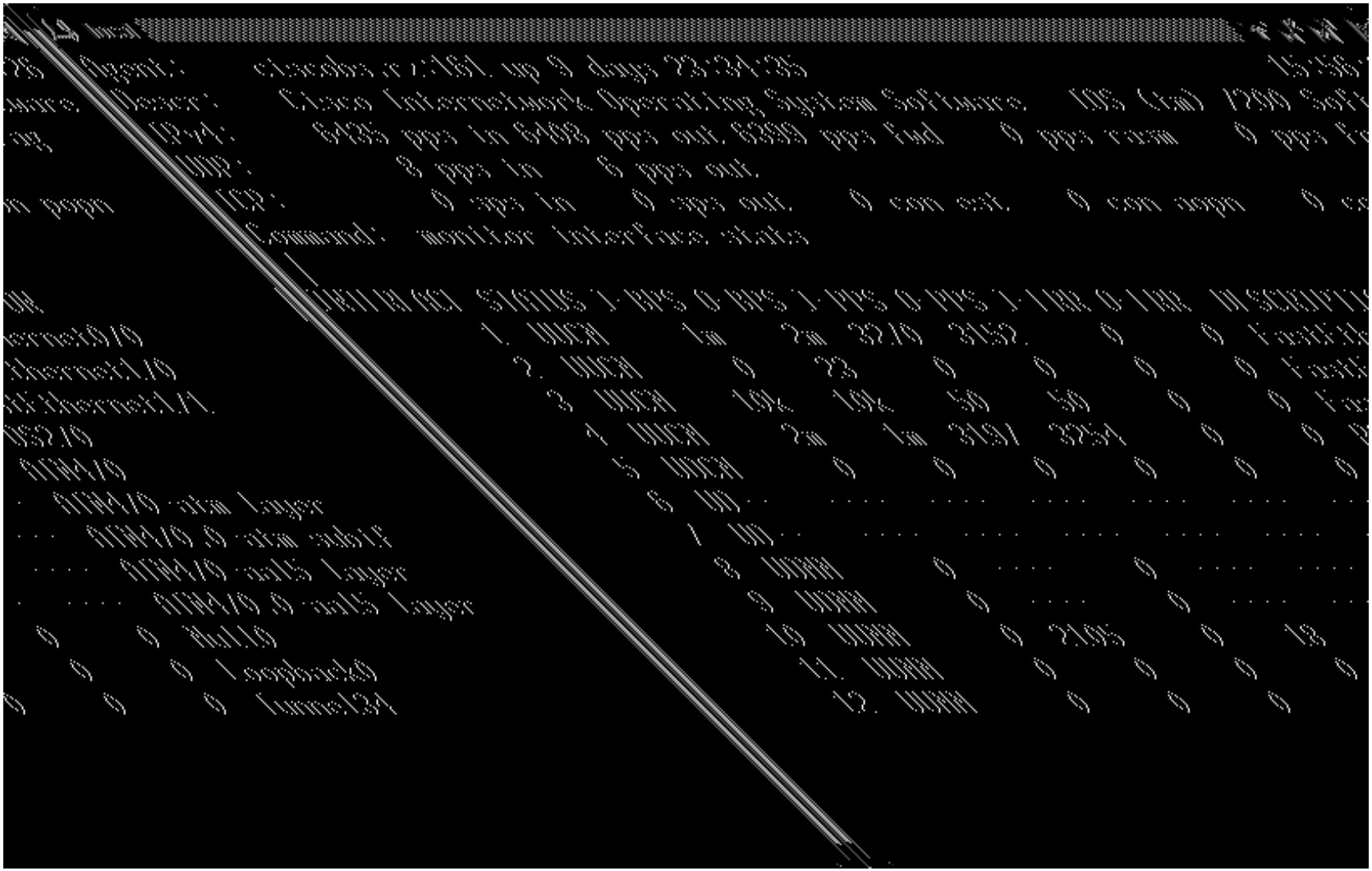
```
delete nortel bridge vlan "^(134|ibr-)"     # regexps are cool :-)

create nortel bridge vlan 544 ibr-core
create nortel bridge vlan 545 ibr-cip
create nortel bridge vlan 546 ibr-test
create nortel bridge vlan 547 ibr-wlan
```

```
define(UP,‘25,185’)                 # uplink ports
define(WLAN,‘2,56’)                 # wireless vlan
define(CORE,‘1,3-24,33-55,65-88’) # core vlan

include(vlan-all.scli)              # create the vlans

set nortel bridge vlan ports ibr-core UP,CORE
set nortel bridge vlan default ibr-core CORE
set nortel bridge vlan ports ibr-wlan UP,WLAN
set nortel bridge vlan default ibr-wlan UP,WLAN
```

Physical Entities (Containment Structure)



Monitoring Network Interfaces



XML Support

- Default output format is optimized for human readability.
- Produce additional XML output format optimized for machine readability.
- XSLT transformations can be used to generate HTML status pages.
- Provides XML Schema definitions for the XML output produced.
- Implementation uses `libxml2` to ensure well-formedness of the XML output.

Stub Code Generator

- Generates stub functions for retrieving and modifying conceptual tables, conceptual rows and groups of scalars.
- Hides all low-level SNMP details such as OID naming.
- Automatic type checking and handling of so called “holes”.
- Data structures use pointers to force programmers to properly check whether data is valid.
- Implementation integrated into `smidump`, part of the `libsmi` package.

Stub Interface for Printer-MIB::prtConsoleLightEntry

```
typedef struct {
    gint32   hrDeviceIndex;
    gint32   prtConsoleLightIndex;
    gint32   *prtConsoleOnTime;
    gint32   *prtConsoleOffTime;
    gint32   *prtConsoleColor;
    guchar   *prtConsoleDescription;
    gsize    _prtConsoleDescriptionLength;
} printer_mib_prtConsoleLightEntry_t;

extern void
printer_mib_get_prtConsoleLightTable(GSnmpSession *s,
        printer_mib_prtConsoleLightEntry_t ***prtConsoleLightEntry, gint mask);

extern void
printer_mib_free_prtConsoleLightTable(printer_mib_prtConsoleLightEntry_t **prtConsoleLightEntry);

extern printer_mib_prtConsoleLightEntry_t *
printer_mib_new_prtConsoleLightEntry(void);

extern void
printer_mib_get_prtConsoleLightEntry(GSnmpSession *s,
        printer_mib_prtConsoleLightEntry_t **prtConsoleLightEntry,
        gint32 hrDeviceIndex, gint32 prtConsoleLightIndex, gint mask);

extern void
printer_mib_set_prtConsoleLightEntry(GSnmpSession *s,
        printer_mib_prtConsoleLightEntry_t *prtConsoleLightEntry, gint mask);

extern void
printer_mib_free_prtConsoleLightEntry(printer_mib_prtConsoleLightEntry_t *prtConsoleLightEntry);
```

Command Implementation

```
static int show_printer_console_lights(scli_interp_t *interp, int argc, char **argv)
{
    printer_mib_prtConsoleLightEntry_t **lightTable;
    int i, light_width = 12;

    if (argc > 1) return SCLI_SYNTAX;

    printer_mib_get_prtConsoleLightTable(interp->peer, &lightTable, 0);
    if (interp->peer->error_status) return SCLI_SNMP;

    if (lightTable) {
        for (i = 0; lightTable[i]; i++) {
            if (lightTable[i]->_prtConsoleDescriptionLength > light_width)
                light_width = lightTable[i]->_prtConsoleDescriptionLength;
        }
        if (! scli_interp_xml(interp)) {
            g_string_sprintfa(interp->header, "PRINTER LIGHT %-*s STATUS COLOR",
                                              light_width, "DESCRIPTION");
        }
        for (i = 0; lightTable[i]; i++) {
            if (scli_interp_xml(interp))
                xml_printer_console_light(interp->xml_node, lightTable[i]);
            else
                fmt_printer_console_light(interp->result, lightTable[i], light_width);
        }
    }

    if (lightTable) printer_mib_free_prtConsoleLightTable(lightTable);
    return SCLI_OK;
}
```

Formatting Function

```
static void
fmt_printer_console_light(GString *s, printer_mib_prtConsoleLightEntry_t *lightEntry, int light_width)
{
    const char *state = "off", *e;

    g_string_sprintfa(s, "%6d  ", lightEntry->hrDeviceIndex);
    g_string_sprintfa(s, "%4d  ", lightEntry->prtConsoleLightIndex);

    if (lightEntry->prtConsoleDescription) {
        g_string_sprintfa(s, "%-*.*s ", light_width,
                    (int) lightEntry->_prtConsoleDescriptionLength,
                          lightEntry->prtConsoleDescription);
    } else {
        g_string_sprintfa(s, "%*s", light_width, "");
    }

    if (*lightEntry->prtConsoleOnTime && !*lightEntry->prtConsoleOffTime) {
        state = "on";
    } else if (!*lightEntry->prtConsoleOnTime && *lightEntry->prtConsoleOffTime) {
        state = "off";
    } else if (*lightEntry->prtConsoleOnTime && *lightEntry->prtConsoleOffTime) {
        state = "blink";
    }
    g_string_sprintfa(s, " %-*s ", 5, state);

    e = fmt_enum(printer_mib_enums_prtConsoleColor, lightEntry->prtConsoleColor);
    g_string_sprintfa(s, "%s\n", e ? e : "");
}
```

Command Registration

```
void scli_init_printer_mode(scli_interp_t * interp)
{
    static scli_cmd_t cmds[] = {
        { "show printer console lights", NULL,
          "The show printer console lights command shows the current\n"
          "status of the printer's lights. [...]",
          SCLI_CMD_FLAG_NEED_PEER | SCLI_CMD_FLAG_XML,
          "printer console",
          "<xsd> <!-- ... --> </xsd>",
          show_printer_console_lights },
        { "monitor printer console lights", NULL,
          "The monitor printer console lights command shows the same\n"
          "information as the show printer console lights command. The\n"
          "information is updated periodically.",
          SCLI_CMD_FLAG_NEED_PEER | SCLI_CMD_FLAG_MONITOR,
          NULL, NULL,
          show_printer_console_lights },
        { NULL, NULL, NULL, 0, NULL, NULL, NULL }
    };

    static scli_mode_t printer_mode = {
        "printer",
        "The scli printer mode is based on the Printer-MIB as published\n"
        "in RFC 1759 and some updates currently being worked on in the\n"
        "IETF Printer MIB working group.",
        cmds
    };

    scli_register_mode(interp, &printer_mode);
}
```

Ideas for Future Work

- More command implementations for additional device types and MIBs.
- Support for SNMPv3 security in the `gsnmp` engine.
- Adopt `gnet` for low-level portable network access?
- Better code generation, e.g. support for spin-lock scalars.
- Automatic caching schemes with intelligent cache validation.
- Code generation for MIB “procedures” from formal MIB annotation.
- Perhaps a `gtk` user interface in addition to the scli and XML output formats.

Try it yourself!

http://www.ibr.cs.tu-bs.de/projects/scli/