



Developing Linux Applications for the EtherTRAK Industrial Ethernet Managed Switch

Abstract

The ET-9MS and ET-5MS series of EtherTRAK Industrial Ethernet Managed Switches are based on IPm Linux, which is also used in the SIXNET IPm products. With proper tools developers and integrators can build and install Linux applications in the switch's CPU. The general concepts of the IADK (IPm Applications Development Kit) or ELDK (Embedded Linux Development Kit) apply. The IADK is used for development of applications for the PPC-based switches. The ELDK is used for development of applications for the ARM-based switches. This document provides details on developing such applications specifically for the managed switch.

Environment

The SIXNET managed switch provides hardware functionality similar to that provided by IPm controllers and RTUs with the following restrictions and differences.

- Only one serial port is available
- Only one Ethernet port is available. (eth0 for arm based switches and eth1 for ppc based switches)
- Only switches with the -IPM option provide a real-time (time-of-day) clock
- The IPm I/O database is only available on switches with the -IPM option.
- Ethernet switches do not include an ST-Bus (SixTRAK I/O) interface. (EtherTRAK I/O modules are recommended for use with EtherTRAK switches.)
- Only -IPM versions include battery backed RAM (/nvram directory).
- ISaGRAF (IEC 6113-3 programming) is not available.
- Only the -IPM option provides support for SIXNET I/O Toolkit features (e.g., datalogging, I/O transfers, SIXNET Universal protocol, Modbus protocol and I/O registers).

Note: The I/O Tool Kit software can be used for firmware image loading in all versions of the SIXNET Industrial Managed Ethernet Switches.

Required Files

Note: These files are supplied by SIXNET to qualified OEM development partners. It is suggested that users who wish to have functionality added to an EtherTRAK Managed Switch contact a qualified IPm Resource Center for assistance.

msdev.tgz – Development Support Update (see below)
mstest.tgz – Example update file
sxsntp.h – Include file for SIXNET SNMP access library
libsxsntp.a – SIXNET SNMP access library
net-snmp libraries – Source available from net-snmp.org; pre-built libraries available on the SIXNET websites:

www.sixnetio.com or www.get2support.com



Development Support Update

By default, the only login available on the managed switch is admin which directly accesses the terminal-based configuration menu. To support development, a root login may be enabled by installing the development update, **msdev.tgz** (supplied with this technical note). This enhancement may be applied using the Firmware Update option of the terminal or graphical interface. Doing so enables a root login (with no password) and also enables ftp (for transferring files into the system). Restoring factory defaults or doing a firmware load from the I/O Tool Kit disables development support.

Notes:

1. This development support enhancement need only be done on development systems to provide greater access to the environment for programming and debugging. The **msdev.tgz** update is **not** required on target systems.
2. For security reasons, you will likely want to set a password on the root account.

Building an Installable Package

An installable package for a Managed Switch is a compressed tar file similar to those described in IPm Technical Note entitled, “Building Slackware-style Packages (.tar files with scripts)”. It consists of zero or more files to install and an optional installation script. The **msdev.tgz** update is a simple example that includes only an install script to reconfigure the system. Another example – that includes files to install – is **mstest.tgz**. You may examine these files with **tar zvtf tarfilename**.

The files should be included with relative paths and will be installed relative to the root directory. For example, **usr/local/bin/myprogram** will be installed to **/usr/local/bin/myprogram**. The attributes of the files (e.g., modification time, permission bits) will be preserved when they are restored.

The install script must be named **install/doinst.sh**. It should have the execute permission bit set and will be invoked after all the files in the package have been put in place. It should do whatever is necessary to activate your package: merge configuration files, copy files to new locations, etc. For example, program files are often stored in **/usr/local/bin** but a running program cannot be replaced directly. To minimize downtime, you might place a new **myprogram** in **/usr/local/bin/myprogram.new** then have the install script move (using **mv**, *not* **cp**) **myprogram.new** to **myprogram**, stop the running version and start the new version.

Execution Environment

The product family and firmware build can be found in **/etc/sxbuildinfo.txt**. Managed switches will have “ET-9MS” or “ET-5MS” at the start of the file. The model number of the device where your program is running can be determined by examining the **sxid** field in **/proc/cmdline**. The leading “0190” or “0150” indicates a managed switch. The rest of the field indicates the model.

Starting Your Application

For your application to start automatically each time the switch is restarted, you must update **/etc/inittab** or place a script in **/etc/rc.d/rc3.d/**. Consult Linux system administration references for details on editing **inittab** or creating startup scripts.

Note: The single serial port on the managed switch is configured for interactive login. To use the serial port for other purposes, you will have to modify **inittab** to disable **getty** on the serial port.



SNMP Access

Many aspects of the managed switch's status are available via SNMP (Simple Network Management Protocol). Some configuration values may also be set via SNMP. See the switch's Software User Manual for details on the SNMP MIBs (Management Information Bases) supported by the switch.

Remote SNMP Access

Any SNMP-aware tool – including custom programs – can be used to remotely access SNMP values in the switch. It is necessary to have proper credentials. SNMPv2 requires only a community string. SNMPv3 requires a user name and password. Note that SNMPv3 is not supported prior to firmware revision 2.1.

There are two levels of SNMP access: read-only and read/write. In SNMPv2, the read-only community string defaults to “public” but it may be configured to another value at any time. The SNMPv2 read/write community string defaults to “private” but may be configured to another value at any time. The SNMPv3 user names are the same as the corresponding community strings. The default SNMPv3 read-only password is “publicpwd” and the default read/write password is “privatepwd”. These may be configured to another value at any time. See the Software User Manual for details on SNMPv3 encryption, authentication, and privacy.

Local SNMP Access

When building Linux applications for the managed switch, the `sxsnmp` library can simplify access to SNMP values. It provides the utility functions listed below. (You should include `sxsnmp.h` in your source and must link against `libsxsnmp.a` to use these functions.)

Note: `sxsnmp` relies on `net-snmp 5.1.1`, an open-source package available from `net-snmp.org`. Other entries in the SNMP MIB can be read and written with the `net-snmp` API. See the examples at `net-snmp.org` for details.

struct snmp_session* SXSNMPCconnect()

Inputs

None.

Outputs

None.

Return

An `snmp_session` if successful or `NULL` if not.



void SXSNMPPDisconnect(snmp_session* ss)

Inputs

ss – Session returned by SXSNMPCConnect().

Outputs

None.

Return

None.

int SXSNMPPGetLeds(int leds[])

Inputs

None

Outputs

LEDs – a 3-element integer array. On return each element will be 0 (off), 1 (on), or 2 (unknown). leds[0] corresponds to the P1 LED on the switch, leds[1] to P2, and leds[2] to OK.

Return

SXSNMPP_SUCCESS or SXSNMPP_FAILURE

NOTE: SXSNMPCConnect() must be called successfully before this function can be used.

int SXSNMPPGetPortStatus(portStatus_t* statuses, size_t numStatus)

Inputs

statuses – the .port field of each structure in the array must be initialized to indicate which port (0..9) should be accessed.

numStatus – The number of structures in statuses.

Outputs

statuses – The adminStatus, operStatus, speed, and duplex fields of each structure will be set (if successful).

Return

SXSNMPP_SUCCESS or SXSNMPP_FAILURE

NOTE: SXSNMPCConnect() must be called successfully before this function can be used.



int SXSNNMPGetStpStatus(char* root, char* myMac, stpPortStatus_t* statuses, size_t numStatus)

Inputs

statuses – the .port field of each structure in the array must be initialized to indicate which port (0..9) should be accessed.

numStatus – The number of structures in statuses.

Outputs

root – A 4-byte MAC address of the current root bridge in the spanning tree.

myMac – A 4-byte MAC address of this switch

statuses – The state and cost fields of each structure will be set (if successful).

Return

SXSNNMP_SUCCESS or SXSNNMP_FAILURE

NOTE: SXSNNMPConnect() must be called successfully before this function can be used.

int SXSNNMPGetComm(char* rocomm, char* rwcomm)

Inputs

None.

Outputs

rocomm – The read-only community string for this switch. It may be at most SX_COMM_MAX characters long.

rwcomm – The read/write community string for this switch. It may be at most SX_COMM_MAX characters long.

Return

SXSNNMP_SUCCESS or SXSNNMP_FAILURE.

NOTE: These community strings are essential when establishing your own SNMP sessions.

NOTE: This function is deprecated in firmware versions 2.1 and later. Use SXSNNMPGetSec() instead.



int SXSNNMPGetSec(sxSecInfo_t* sec)

Inputs

None.

Outputs

sxSecInfo_t structure filled in as follows:

v2 – 1 if SNMPv2 security is enabled, 0 otherwise

v3 – 1 if SNMPv3 security is enabled, 0 otherwise.

roname – read-only community string and user name.

rwname – read-write community string and user name.

Return

SXSNNMP_SUCCESS or SXSNNMP_FAILURE.

NOTE: It is not possible to retrieve the SNMP passwords.

Sample Code

The following example program demonstrates the basic use of the SXSNNMP API. It may be built with commands such as:

```
cc example.c -I.
```

```
cc -o example example.o libsxsnmp.a -lnetsnmp -ldl -lcrypto0 -lm
```



Source

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sxsnmp.h>

int main()
{
    int i;
    int retval;

    int leds[3];

    char* ledLabels[3] = { "Power 1", "Power 2", "OK" };

    struct snmp_session* ss;

    // Get dynamic data via SNMP
    ss = SXSNMPConnect();
    if (ss == NULL) {
        fprintf(stderr, "Could not connect to SNMP.\n");
        exit(1);
    }

    if (SXSNMPGetLeds(leds) != SXSNMP_SUCCESS) {
        retval = 1;
        fprintf(stderr, "Could not get LED status\n");
    } else {
        retval = 0;
        for (i = 0; i < 3; ++i) {
            fprintf(stdout, "%s\t", ledLabels[i]);
            switch (leds[i]) {
                case 0:
                    fprintf(stdout, "Off\n");
                    break;
                case 1:
                    fprintf(stdout, "On\n");
                    break;
                case 2:
                default:
                    fprintf(stdout, "*Unknown*\n");
                    break;
            }
        }
    }

    SXSNMPDisconnect(ss);

    exit(retval);
}
```