INSTITUT FÜR INFORMATIK

DER LUDWIG MAXIMILIAN UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Implementierung eines SNMP-Subagenten zur Überwachung von Syslog Meldungen

Christian Coehn

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Stephen Heilbronner / Boris Gruschke
Abgabedatum:	10.11.1997

Inhaltsverzeichnis

Kapitel 1

Einführung

Einer der wichtigsten Wünsche eines Systemadministrators ist ein integriertes Netz- und Systemmanagement, welches die Betreuung eines großen und heterogenen Netzes und der angeschlossenen Workstations vereinfacht. In der Internetwelt wird versucht, diesem Wunsch durch das Internet Management mit seinem standardisierten Protokoll SNMP Rechnung zu tragen.
Das Problem für den Systemadministrator ist dabei zum einen, daß das Internet Management mit der Standard MIB-II derzeit vor allem ein Netz- und weniger ein Systemmanagement ist, und zum anderen, daß viele Anwendungen und Tools, die zum Standardlieferumfang eines jeden UNIX gehören auch heute noch nicht mittels SNMP zu managen sind.
Ziel dieses Fortgeschrittenenpraktikums war es deshalb, dem Systemadministrator ein Werkzeug an die Hand zu geben, welches als Mittler zwischen dem lokalen Log basierten Event Handling der UNIX Tools und dem Trap basierten Event Handling der SNMP Management Tools dient und somit ein zentrales Fault Management ermöglicht.
Es entstand ein SNMP DPI Subagent, der alle Ausgaben des Syslog Daemons überwacht und daraus konfigurierbare SNMP Traps erzeugt. Die Implementierung erfolgte in PERL 5.

1.1 Der Syslog-Daemon

Der Syslog-Daemon ist ein an der Universität von Berkeley entwickeltes Systemwerkzeug, welches den Anwendungs- und Systemprogrammen eine standardisierte Schnittstelle (API) bietet, um Nachrichten, Warnungen und Fehler zu protokollieren. Der Syslog-Daemon ist heute fester Bestandteil von allen BSD und System V Unix-Varianten. Der Vorteil für den Programmierer liegt darin, daß er sich nicht selbst um das Handling von Logdateien und um die Auswahl und Menge von Lognachrichten kümmern muß. Er schickt einfach alle Nachrichten an den Syslog-Daemon, wo die weitere Verarbeitung erfolgt.
Der Vorteil für den Anwender liegt darin, daß er nicht für jedes Programm das Logging einzeln konfigurieren muß. Stattdessen legt er einmal alle Einstellungen zentral in einer Konfigurationsdatei fest. Die Einstellungen gelten sodann für alle Programme, die sich des Syslog API bedienen.
Jede Nachricht, die ein Programm an den Syslog Daemon sendet, wird dabei über die zwei Parameter *Facility* und *Severity* eindeutig klassifiziert.
Der Parameter *Facility* spezifiziert dabei, was für eine Art Programm die Nachricht sendet. Dies erlaubt es, Lognachrichten von verschiedenen Klassen von Programmen, unterschied-

lich zu behandeln. Die folgenden *Facilities* werden dabei in `/usr/include/syslog.h` zur Verfügung gestellt.

- AUTHPRIV : Nachrichten bzgl. Sicherheit und Authentifizierung
- CRON : Nachrichten vom Cron Daemon (cron und at)
- DAMEON : Nachrichten von anderen System Daemonen
- KERN : Nachrichten vom Kernel
- LOCAL0 - LOCAL7 : Reserviert für lokale Anwendungen zur freien Benutzung
- LPR : Nachrichten vom Line Printer Subsystem
- MAIL : Nachrichten vom Mailsystem
- NEWS : Nachrichten vom Newssystem
- SYSLOG : Interne Nachrichten vom Syslog Daemon
- USER : Generische User Nachricht
- UUCP : Nachrichten vom UUCP Subsystem

Der Parameter *Severity* gibt den Informationsgehalt einer Nachricht bzw. die Schwere eines aufgetretenen Fehlers an. Die folgenden *Severities* stehen dabei (in absteigender Wichtigkeit) zur Verfügung:

- EMERG : Das System ist nicht mehr benutzbar
- CRIT : Es ist ein kritischer Zustand aufgetreten
- ERR : Es ist ein Fehler aufgetreten
- WARNING : Es handelt sich um eine Warnmeldung des Systems
- NOTICE : Es handelt sich um eine für den Administrator wichtige Meldung
- INFO : Es handelt sich um eine Infomeldung
- DEBUG : Es handelt sich um eine Debug Meldung.

Wenn ein Programm nun Nachrichten über den Syslog Daemon protokollieren möchte, so benutzt es die Funktionen `openlog()`, `syslog()` und `closelog()` der C Runtime Library. Auch diese Funktionen werden über `/usr/include/syslog.h` eingeschlossen. Der Daemon nimmt die Nachrichten des Programms entgegen und schreibt sie in die entsprechenden Log Dateien.

Um festzulegen, welche Meldungen in welche Logdatei geschrieben werden, konfiguriert der Anwender den Syslog Daemon über die Datei `/etc/syslog.conf`. Diese Konfigurationsdatei setzt sich aus Paaren von Regeln und Zielen zusammen. Das Regelfeld selektiert dabei die Nachrichten, das Zielfeld gibt an, in welche Datei die Nachrichten geschrieben werden sollen. Ein typischer Aufbau könnte z.B. so aussehen:

```
kern.crit           /dev/console
mail.*;mail.!=info  /var/adm/mail
mail,news.=info     /var/adm/info
```

Die erste Pattern erkennt alle Nachrichten der Facility Kernel mit der Priorität Critical und höher und schreibt diese auf die Bildschirmkonsole.
Die zweite Regel erkennt alle Nachrichten der Mail Facility, die nicht die Priorität Info haben und schreibt diese in die Datei /var/adm/mail.
Das letzte Pattern schließlich matcht alle Nachrichten der Mail und News Facilities, die genau die Priorität Info haben und schreibt diese in die Datei /var/adm/info.

1.2 Syslog und SNMP Management

Wie bereits in der Einführung angesprochen ist es ein Problem, daß viele Standard UNIX Tools und Programme nicht über SNMP zu managen sind. Dies hat seine Ursache zu einem Teil darin, daß viele dieser Tools aus den Anfängen der UNIX Zeit stammen, zu der es noch keinen Bedarf eines integrierten Systemmanagements gab. Zum anderen werden viele Tools als Public Domain angeboten und die Autoren konzentrieren ihre Zeit darauf, die Kernfunktionalität ihrer Programme weiterzuentwickeln - für Zusatzfunktionen wie SNMP Management bleibt oft keine Zeit.
Aufgrund der standardisierten Schnittstelle und der einfachen Benutzbarkeit aus fast allen systemrelevanten Programmier- und Skriptsprachen unterstützen aber fast alle Programme das Melden von Fehlern und Informationen an den Syslog-Daemon.
Der Syslog-Daemon ist also eine lokale Schnittstelle, an der alle Meldungen aller im System laufenden Programm bereits klassifiziert zusammenlaufen. So wird so ein Fault- und Security Management auf Einzelsystemebene ermöglicht. Auf der anderen Seite bietet das Internet Management Zusätzlich zum Polling des Agenten durch den Manager die Möglichkeit, Ausnahmezustände und Informationsmeldungen via SNMP Traps vom Agenten an einen entfernten Manager zu schicken. Es drängt sich deshalb geradezu auf, die eintreffenden Systeminformationen an der Syslog Schnittstelle abzugreifen und in ein SNMP kompatibles Format, also Traps umzusetzten.
Durch die Umsetzung von Syslog Meldungen in Traps werden non-SNMP Programme zwar auch nicht vollständig durch SNMP managebar, da der Manager keinen Einfluß auf die Programme nehmen kann, die Programme werden aber zumindest um die Funktionalität eines zentralen Fault- und Securitymanagements im Netz erweitert.

Kapitel 2

Management von Syslog Meldungen

2.1 Anforderungen

Um den Subagenten, der die Umsetzung von Syslog Meldungen in SNMP Traps vornimmt, für den Administrator leicht beherrschbar und universell einsetzbar zu machen, wurden folgende Anforderungen aufgestellt:

- Der Subagent muß komplett vom Manager zu konigurieren sein. Lokale Konfigurationsdateien sind nicht erwünscht.
- Die Umsetzung von Syslog Meldungen in Traps muß konfigurierbar sein. Es bietet sich an, dazu die bereits bestehende Klassifizierung der eingehenden Nachrichte nach *Facility* und *Severity* zu nutzen.
- Der Subagent soll über eine Logfunktion verfügen, die ausgelöste Traps mitprotokolliert.
- Der Subagent soll auch die Verwaltung der Syslog Konfigurationsdatei `/etc/syslog.conf` über den Manager erlauben.

2.2 Die Syslog MIB

Aus den gestellten Anforderungen wurde eine SNMPv2 MIB entwickelt. Die MIB wird hier verkürzt dargestellt, sie findet sich vollständig im Anhang.

```
SYSLOG-MGMT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
Integer32, TimeTicks
        FROM SNMPv2-SMI
    RowStatus, DisplayString
        FROM SNMPv2-TC;

syslogMIB MODULE-IDENTITY
    LAST-UPDATED "9711020000Z"
    ORGANIZATION "MNM Team Munich"
```

```
    CONTACT-INFO "      Christian Coehn
                coehn@informatik.uni-muenchen.de"
    DESCRIPTION  "Management des Syslos Daemons"
    ::= { iso 3 6 1 3 100 8 1 }

-- DIE MIB ist in 4 Gruppen organisiert:
-- 1. Trapkonfiguration
-- 2. Traphistory
-- 3. General
-- 4. Syslog Konfiguration
```

2.2.1 Die Trap Konfiguration

Die erste Gruppe der entwickelten MIB beschäftigen sich ausschließlich mit der Konfiguration der Traperzeugung durch den Subagenten. Dabei werden die Trap Auslöser, also die Konfiguration, welche Syslog Meldungen einen Trap auslösen und welche nicht relevant sind, sowie die Konfiguration der ausgelösten Traps in zwei getrennten Tabellen verwaltet. Die Verknüpfung zwischen Auslöser und Trap findet durch die Variablen `trapSourceID` und `trapDestID` der beiden Tabellen `trapSourceTable` und `trapDestTable` statt. Es handelt sich dabei um eine 1:n Beziehung, d.h. eine bestimmte Syslog Meldung kann beliebig viele Traps auslösen.

```
TrapSourceTableEntry ::=
    SEQUENCE {
        trapSourceID        Integer32,
        trapSourceTableStatus   RowStatus,
        facility        DisplayString,
        severity        DisplayString,
        logstring       DisplayString
    }
```

Um zu bestimmen, ob eine Syslog Meldung einen Trap triggern soll, stehen dem Administrator die Einstellungen *Facility, Severity* und *Logstring* zur Verfügung. Bei *Facility* und *Severity* handelt es sich um die bereits angesprochene Klassifizierung einer Syslog Meldung, *Logstring* ist ein regulärer

Ausdruck nach Perl 5 Definition und wird mit dem geloggten String selbst verglichen. Durch Setzen der Variablen werden Auslöser konfiguriert:

- *Facility*=`AUTHPRIV`, *Severity*=`.*`, *Logstring*=`.*su*` erzeugt bei einem fehlgeschlagenen Versuch, mittels *su* Root Rechte zu erlangen.

- *Facility*=`.*`, *Severity*=`EMERG`, *Logstring*=`.*` erzeugt Traps beim Auftreten von kritischen Systemfehlern.

Die Liste von Beispielen läßt sich beliebig fortführen.

```
TrapDestTableEntry ::=
    SEQUENCE {
        trapDestIndex   Integer32,
```

```
        trapDestID      Integer32,
        trapDestVal     Integer32,
        trapDestTableStatus RowStatus
    }
```

Für jeden ausgelösten Trap kann der Administrator hier einen Integer Wert festlegen, der in das Trap Paket gepackt werden soll. Ursprünglich sollte hier das Trapziel verwaltet werden. Wie sich später jedoch herausstellte erlaubt das DPI Protokoll keine Übergabe von Trapzielen an den Hauptagenten. Auf weitere Probleme bei der DPI Kommunikation gehe ich im Abschnitt *Bekannte Probleme* ein.

Die es sich bei den durch den Subagent ausgelßten Traps nicht um generische Traps handelt, sondern auch ein Wert übermittelt werden soll, wird noch eine Variable vom Typ `NOTIFICATION-TYPE` benötigt. Beim Auslösen eines Traps wird hier die Zeilennummer der TrapSourceTable, die den Trap getriggert hat mitgegeben.

```
trapSyslogNotification NOTIFICATION-TYPE
    OBJECTS {trapSourceID}
    STATUS current
    DESCRIPTION
        "Informiert den Manager, dass dem Syslog Agent ein Zustand gemeldet
     wurde, fuer den ein Trap konfiguriert wurde. Die Notification enthaelt
     dabei die ausloesende Instanz der Trap Source Tabelle."
     ::= { trapConfig 5 }
```

2.2.2 History Management

Alle ausgelösten Traps in einer Tabelle festgehalten, damit der Systemadministrator das Auftreten eines Problems auch später noch nachvollziehen kann. In der Trapihistorytabelle werden eine fortlaufende *trapNummer*, die *trapID* und die genaue *trapZeit* festgehalten.

```
TrapHistoryTableEntry ::=
    SEQUENCE {
        trapNo      Integer32,
        trapID      Integer32,
        trapTime    DateAndTime
```

Damit die Tabelle der ausgelösten Traps nicht zu groß wird, hat der Administrator die Möglichkeit, das Anwachsen über die beiden Variablen *historyMaxAge* und *historyMaxEntries* zu begrenzen.

```
historyMaxAge OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Maximales Alter der Eintraege in der Trap History in Minuten"
    ::= { traphistory 1 }
```

```
historyMaxEntries OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Maximale Anzahl an Eintraegen in der Trap History Tabelle"
```

Dabei werden aus der Traphistorytabelle alle Einträge gelöscht, die älter als *historyMaxAge* Minuten sind. Wächst die Anzahl der Einträge trotzdem über *historyMaxEntries* hinaus, so werden vom Beginn der Tabelle entsprechend Einträge gelöscht.

2.2.3 Syslog Konfiguration

```
SyslogConfTableEntry ::=
    SEQUENCE {
        syslogLineNo Integer32,
        syslogLine DisplayString,
        syslogConfTableStatus RowStatus
    }
```

Die Tabelle *SyslogConfTable* enthält die gesamte Datei `/etc/syslog.conf` Zeile für Zeile. Bei jeder Änderung der Tabelle wird die Datei neu geschrieben und dem Syslog Daemon das erneute Einlesen der Konfigurationsdatei mittels SIGHUP signalisiert.

2.2.4 Weiteres

Der Subagent verfügt noch über weitere Steuervariablen, die z.B. die Agenten Uptime, das SNMP Verhalten (v1 oder v2) in bestimmten Situationen und weiteres repräsentieren.

```
sysUpTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "Startzeipunkt des Agenten"
    ::= { general 1 }

sysStatus OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Steuervariable, steht beim Start des Subagenten auf 1, zum
         Beenden des Subagenten auf 0 setzen."
    ::= { general 2 }

sysSnmpVersion OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
```

```
STATUS current
DESCRIPTION
    "Steuervariable, die das Verhalten des Subagenten bei bestimmten
     Fehlermeldungen (z.B. no such instance) bestimmt. Wird die
     Variable auf 1 gesetzt, werden SNMPv1 Fehler generiert, wird
     die Variable auf 2 gesetzt, werden SNMPv2 Fehler generiert.
     Die Variable hat keinen Einfluss darauf, ob der Hauptagent
     SNMPv1 oder SNMPv2 als Protokoll benutzt.
     Default beim Start des Subagenten ist SNMPv2."
::= { general 3 }
```

Kapitel 3

Implementierung

3.1 Das DPI Protokoll

Das von IBM entwickelte DPI (*Distributed Program Interface*) Protokoll liegt derzeit in der Version 2.0 vor und ist in RFC-1592 dokumentiert. Das Protkoll erlaubt es, den Hauptagenten zu modularisieren und die Funktionalität, für den Manager transparent, auf viele Subagenten zu verteilen.

Ohne die Möglichkeit, Aufgaben an spezialisierte Subagenten verteilen zu können, müßte bei jeder Änderung an der MIB der Hauptagent geändert werden. Dies ist aus mehreren Gründen nicht wünschenswert: zum einen liegt der SNMP Agent bei einer kommerziellen Implementierung oft nicht im Source Code vor und eine Erweiterung ist entweder gar nicht möglich, oder mit erheblichen Kosten verbunden. Bedient man sich einer freien Implementierung, so ist die ständige Erweiterung des Agenten dennoch problematisch, da der Agent zu einem großen Monolith anwächst und immer schwerer zu warten wird. Eine Erweiterung der Standard MIB um Technologie- und Firmen-MIBs, wie es das Informationsmodell des Internet Managements vorsieht, läßt sich so nur schwer realisieren.

Da der SNMP Standard keinerlei Erweiterungen der Fähigkeiten des Agenten durch den Benutzer vorsieht, wurde bei IBM eine Möglichkeit dafür geschaffen: spezialisierte Subagenten melden sich beim Hauptagent an und registrieren den Teilbaum der MIB, für den sie verantwortlich sind. Empfängt nun der Hauptagent eine SNMP Anfrage, die im Zuständigkeitsbereich des Subagenten liegt, so delegiert er die Anfrage und sendet die Antwort vom Subagenten an den Manager zurück. Die Subagenten bleiben dabei verborgen, aus der Sicht des Managers findet die Kommunikation nur mit einem Partner statt.

Der Ablauf einer DPI Kommunikation findet wie folgt statt:

- Der Subagent sendet ein spezielles SNMP Paket an den Hauptagenten. Ist dieser DPI fähig, so antwortet er mit dem TCP Port, auf dem DPI Requests angenommen werden.

- Der Subagent sendet dann einen DPI Open Request, der die Object ID, also die Wurzel des Teilbaumes, für die der Agent zuständig ist, enthält.

- Nach der Quittierung des Open Paketes durch den Hauptagenten registriert der Subagent sämtliche Variablen, für die er zuständig ist, beim Hauptagent.

- Der Subagent ist nun einsatzbereit und nimmt die SNMP Befehle GET, GETNEXT, GETBULK und SET entgegen. Er reagiert ferner auf die speziellen DPI Befehle COM-

MIT und UNDO.

- Beim Beenden des Subagenten deregistriert dieser seine Variablen beim Hauptagenten und sendet einen Close Request.

Wozu nun die Befehle COMMIT und UNDO? Geht beim Hauptagenten ein SNMP Paket mit mehreren SET Befehlen ein, so kann es sein, daß diese Befehle auf mehrere Subagenten verteilt werden müssen. Tritt nun bei einem Subagenten ein Fehler auf, d.h. der SET Befehl kann nicht ausgeführt werden, so müssen auch die anderen Subagenten rückgesetzt werden, um Inkonsistenzen zu vermeiden. Um ein derartiges Rollback zu ermöglichen, arbeitet DPI mit einer 2 Level Commit Strategie. Ein eingehendes SET wird auf die Subagenten verteilt. Diese treffen alle Vorbereitungen für das Setzen der Variablen und melden ein OK an den Hauptagenten. Sind alle Subagenten fertig, sendet der Hauptagent ein COMMIT, um das Setzen der Variablen endgültig zu machen. Meldet hingegen ein Subagent einen Fehler beim SET, so sendet der Hauptagent an die anderen Subagenten ein UNDO, um die Änderungen wieder rückgängig zu machen.

Tritt bei einem Subagent der Fehler erst beim COMMIT auf, so liegt das weitere Vorgehen beim Hauptagenten, eine genauere Verfahrensweise ist nicht beschrieben.

Zur DPI Kommunikation mit dem Hauptagenten wurde die von Frank Schütz im Rahmen eines Fopras in PERL 5 implementierte DPI Library benutzt und erweitert.

3.2 Der Syslog Subagent im Einsatz

Vor dem ersten Start des Agenten muß der Administrator das Startup Skript *subagent_start.pl* durchgehen und eventuell die Variablen anpassen. Zu konfigurieren sind insbesondere:

- SNMP Port und SNMP Trap Port (`$SNMP_PORT`, `$SNMP_TRAP_PORT`).
- SNMP Community und Trap Community Strings (`$SNMP_COMMUNITY`, `$SNMP_TRAP_COMMUNITY`).
- SNMP OID (`$GROUPID`, `$OID`). Bei Änderung ist auch die MIB anzupassen.
- Loglevel (`$DEBUGLEVEL`).
- Der Hostname des Rechners, auf dem der Hauptagent läuft (`$AGENT_HOST`).
- Der Pfad zum Subagenten (`$ProgPfad`).
- Der Pfad zur Syslog Konfigurationsdatei (`$SyslogConfPfad`).
- Der Syslog Socket/Pipe, von dem Meldungen gelesen werden sollen (`$SYSLOGPIPESOCK`).
- Optional ein Socket, an den die abgefangenen Meldungen weitergereicht werden sollen (`$SYSLOGSOCKNEW`).

Anschließend kann der Syslog Agent durch den Aufruf von *./subagent_start* gestartet werden. Der Subagent benötigt dabei prinzipiell keine Root Privilegien. Soll jedoch auch die Datei */etc/syslog.conf* über den Subagent gemanaged werden, so sollte die Gruppenzugehörigkeit der Skripten Schreibrechte auf diese Datei erlauben. Zu Testzwecken kann dem Agent durch Setzen der Variable `$SyslogConfPfadWrite` mitgeteilt werden, die Syslog Konfiguration nicht zu überschreiben, sondern in eine andere Datei zu sichern.

Beim Start des Subagenten können zwei Parameter mitgegeben werden:

- -h ¡host¿: Host, auf dem der DPI Hauptagent läuft. Hat Vorrang vor `$AGENT_HOST`
- -d: Veranlaßt den Subagent dazu, Debugmeldungen nach STDOUT zu schreiben. Wird diese Option nicht angegeben, so läuft der Subagent im Hintergrund und gibt keine Meldungen aus.

Der Subagent meldet sich nun beim Hauptagenten an und registriert seinen MIB Teilbaum. Sollte dabei ein Fehler auftreten, so wird das Programm beendet. Gelingt das Anmelden hingegen, so ist der Subagent betriebsbereit und kann über einen Manager oder auch über eine Skriptdatei, die das Programm SNMPSET verwendet, konfiguriert werden. Die Trapkonfiguration, die History Konfiguration und das Management der Syslog Konfiguration wurden dabei bereits bei der Vorstellung der MIB besprochen. Drei weitere Variablen der MIB steuern das allgemeine Verhalten des Subagenten: `sysUpTime` beinhaltet die Startzeit des Subagenten, `sysStatus` dient zum Beenden des Agenten und `sysSnmpVersion` bestimmt das Verhalten des Subagenten im Fehlerfall.

Nur die letzten beiden Variablen bedürfen der Erläuterung. Wird `sysStatus` auf 0 gesetzt, so deregistriert sich der Subagent beim Hauptagent und beendet sich. Alle anderen Werte bewirken keine Veränderung.
Die Variable `sysSnmpVersion` kann auf 1 oder 2 (entsprechend SNMPv1 und SNMPv2) gesetzt werden. Damit hat der Administrator die Möglichkeit, auf die Art der Tabellenverwaltung und die generierten Fehlermeldungen Einfluß zu nehmen. Der Unterschied zwischen SNMPv1 und SNMPv2 liegt hier im Informationsgehalt der Rückmeldungen. Vereinfacht gesagt meldet ein SNMPv1 Agent bei einem fehlerhaften GET oder SET einfach ein *Fehler*, ein SNMPv2 jedoch ein *Fehler, weil ...* zurück.
Die Variable `sysSnmpVersion` hat jedoch keinen Einfluß darauf, ob SNMPv1 oder SNMPv2 als Kommunikationsprotokoll zwischen Agent und Manager verwendet wird. Das Verwendet von SNMPv2 Semantik bei Fehlermeldungen mit SNMPv1 Kommunikation ist problemlos möglich und wird auch von Marshall Rose empfohlen. Der Defaultwert der Variable ist deshalb 2 und sollte nicht verändert werden.

Kapitel 4

Entwicklung eines eigenen Subagenten

Die modulare Struktur des Subagenten sollte es erlauben, durch vergleichsweise geringe Anpassungsarbeit, eigene DPI Subagenten zu bauen. Das folgende Kapitel erklärt dazu den Aufbau der verwendeten Module und die verwendeten Funktionen.

4.1 Genereller Programmablauf

Das Programm wird durch Aufruf des PERL Skripts *subagent_start* gestartet. PERL 5.0 oder neuer ist dabei Vorraussetzung. Es werden nun alle weiteren benötigten Programmteile mittels `require` eingebunden, die Konfigurationsvariablen gesetzt, die syntaktische Struktur der MIB konfiguriert und alle MIB Variablen, die schon zu Programmstart einen Wert haben sollen, initialisiert. Anschließend wird die Prozedur `Subagent()` gestartet.

Die Prozedur `Subagent()` öffnet die Verbindung zum Hauptagenten und registriert dort die MIB. Anschließend wird der DPI Filehandle bei einem Event Dispatcher registriert und die Kontrolle an diesen Dispatcher abgegeben.

Trifft eine DPI Meldung ein, so wird die Funktion `DPI_Handler()` aufgerufen. Die Routine dekodiert das eingegangene Paket und ruft, je nach Art des Requests eine der Funktionen `answer_get` für einen GET Request, `answer_getnext` für einen GETNEXT Request oder `answer_setgrp` für einen Request aus der SET Gruppe (SET, COMMIT und UNDO).

Die entsprechenden Funktionen verzweigen nun in eines der Moduln *mib_gets.pl* und *mib_sets.pl*, wo kaskadierte if-then-else Schleifen durchlaufen werden. An dieser Stelle wird die OID dekodiert und dann die entsprechende GET/SET/COMMIT/UNDO Funktion in den Moduln *sub_gets.pl*, *sub_sets.pl*, *sub_coms.pl* und *sub_undos.pl* aufgerufen. In diesen Moduln existieren generische Funktionen für GET, SET, COMMIT und UNDO von Skalar- und Tabellenwerten. In diesen Moduln befinden sich ebenfalls spezielle GET/SET Funktionen für ganz bestimmte MIB Variablen.

Nach der Abarbeitung des DPI Kommandos kehrt das Programm zur Hauptprozedur `Subagent()` zurück. Diese Endlosschleife wird nur verlassen, wenn eine Steuervariable, die in der MIB bereitgestellt werden muß, auf 0 gesetzt wird:

```
#-------------------------------------------------------------
# Warten auf eine Anfrage
```

```
while(!($Wert{'3.2.0'} == 0)) {   # Warten auf DPI-Anfragen des Agenten
    &dispatch_now (30);
}
#---------------------------------------------------------------
```

Wird die Agentensteuervariable auf 0 gesetzt, so wird `DPI_Handler()` beim Event Dispatcher und die MIB Variablen beim Hauptagenten deregistriert. Anschließend wird die Verbindung zum Hauptagent geschlossen und das Programm beendet.

Hinweise für die eigene Implementierung:

Die Übertragung des DPI Handlings an einen Event Dispatcher war nötig, da der Syslog Agent neben dem Socket zur DPI Kommunikation noch weitere asynchrone Ereignisse überwachen muß. Wird diese Funktionalität nicht benötigt, so kann auf den Event Dispatcher komplett verzichtet werden und ein direktes Lesen vom DPI Port stattfinden.

Anstatt

```
...
# Registrieren des DPI Handlers beim Event Dispatcher
print_debug (3, "Registriere DPI_Handler beim Dispatcher");
&dispatch_addfunc (fileno(DPI_Anschluss), \&DPI_Handler);

#---------------------------------------------------------------
# Warten auf eine Anfrage
while(!($Wert{'3.2.0'} == 0)) {   # Warten auf DPI-Anfragen des Agenten
    &dispatch_now (30);
}
#---------------------------------------------------------------
# Unregistrieren des Teilbaums
&DPI_UnRegister;
...
```

kann dann

```
...
#---------------------------------------------------------------
# Warten auf eine Anfrage
while(!($Wert{'3.2.0'} == 0)) {   # Warten auf DPI-Anfragen des Agenten
    recv(DPI_Anschluss,$Laenge,2,0);
    $Laenge = unpack("n",$Laenge);
    recv(DPI_Anschluss,$dpi_packet,$Laenge,0);
    @dpi_header = unpack("H2H2H2nH2",$dpi_packet);  # Header auslesen

    if ($dpi_header[4] eq "01") # Falls get-Request
        {
        print "Get-Request empfangen\n";
        &answer_get;        # Beantworte get-Request
        }
    elsif ($dpi_header[4] eq "02")  # falls getnext-Request
        {
```

```
        print "Getnext-Request empfangen\n";
        &answer_getnext;    # Beantworte getnext-Request
        }
    elsif ($dpi_header[4] eq "03" || $dpi_header[4] eq "0a" || $dpi_header[4] eq
 "0b")
        {   # falls set-Request Gruppe
        &answer_setgrp($dpi_header[4]);
        }
    elsif ($dpi_header[4] eq "05")  # falls Response
        {
        print "Response empfangen\n";
        next;
        }
    else
        {
        print "Undefinierbare Nachricht empfangen\n";
        print "gemeldeter Packettyp: $dpi_header[5]\n";
        }
}
#----------------------------------------------------------------
...
```

benutzt werden. Die Funktion `DPI_Handler()` und der gesamte Event Dispatcher entfallen.

4.2 Darstellung der MIB im PERL Programm

Zur Übertragung der MIB Struktur in den Subagenten und zur Typisierung der einzelnen MIB Variablen wird das global assoziative Array %MIB benutzt:

```
    $MIB{'1.1.1.1.0'} = "81";    # trapSourceID
    $MIB{'1.1.1.2.0'} = "81";    # trapSourceTableStatus
    $MIB{'1.1.1.3.0'} = "02";    # facility
    $MIB{'1.1.1.4.0'} = "02";    # severity
    $MIB{'1.1.1.5.0'} = "02";    # logstring
    $MIB{'1.2.0'}     = "81";    # trapSourceTableNextInstance
    ...
```

Dabei ist folgendes zu beachten:

- Die Instanz-ID der jeweiligen MIB Variable dient als Zugriffsindex im Hash. Dabei ist darauf zu achten, daß die Instanz-ID in ' zu klammern ist, damit PERL die Punkte nicht interpretiert.
- Die eigentlichen Werte im Hash repräsentieren die Typen (Integer, DisplayString, etc.) der einzelnen MIB Variablen. Dabei entspricht der Typ dem Nummerncode der Tabelle 17 im RFC 1592, jedoch in hexadezimaler Darstellung. Der Typ ist in Anführungszeichen zu setzen.

- Bei Tabellen reicht es, pro Variable nur eine Instanz anzugeben (die Zeile 0).

Für das Beantworten eines GETNEXT Requests ist es notwendig, daß die MIB Variablen auch in sortierter Reihenfolge vorliegen. Dazu nimmt das globale Array @MIB die durch die Funktionen `by_hierarchy()` und `less_by_hierarchy()` sortierten Variablen des Hashes

```
# Liste der Keys sortieren
@MIB = sort by_hierarchy keys(%MIB);    # sortierte Liste der MIB-Variablen
```

Die Aktualisierung von @MIB wird dabei nicht nur einmal bei der Initalisierung der MIB Variablen zu Beginn des Programms durchgeführt, sondern immer dann, wenn dynamisch Variablen enstehen und gelöscht werden, also z.B. beim Erzeugen oder Löschen von Tabellenzeilen.

4.3 Low Level DPI Funktionen

Der grundsätzliche Ablauf der DPI Kommuniktion wurde bereits im vorigen Kapitel abgehandelt. An dieser Stelle soll nur gezeigt werden, welche Funktionen der Subagent zur DPI Kommunikation zur Verfügung stellt.

- `qDPIport()`
  qDPIPort() ermittelt über eine SNMP Verbindung unter UDP zum Hauptagenten den Port, auf dem der Agent bereit ist, eine DPI Verbindung aufzubauen. Die Funktion benötigt den Hostnamen des Rechners, auf dem der Hauptagent läuft in der globalen Variable $AGENT_HOST.

- `TCPConnect()`
  TCPConnect() baut auf dem übermittelten Port eine TCP Verbindung zum Rechner, der in der globalen Variable $AGENT_HOST genannt ist, auf. Nachdem die Funktion erfolgreich beendet wurde, existiert ein PERL Filehandle namens *DPI_Anschluss* für die weitere Kommunikation.

- `DPI_Open()`
  Diese Funktion dient dazu, eine Verbindung mit dem DPI fähigen SNMP Hauptagente aufzubauen. Sie erstellt ein DPI-Open-Request Paket und verschickt dieses an das Filehandle DPI_Anschluss. Wird das Anfragepaket durch ein Responsepaket vom Hauptagenten positiv beantwortet, so wird hinter den Aufruf der Funktion zurückgekehrt. Andernfalls wird das gesamte Skript mit einer Fehlermeldung beendet.

- `DPI_Register()`
  Diese Funktion registriert den Teilbaum der MIB, für den der Subagent zuständig ist, beim Hauptagenten. Dazu wird dem Hauptagenten in einem DPI-Register-Request Paket die Wurzel des Teilbaums übermittelt. Diese Funktion erwartet die Wurzel in der globalen Variable $GROUPID. Wird die Registrierung durch ein Responsepaket vom Agenten positiv beantwortet, so wird hinter den Aufruf der Funktion zurückgekehrt, andernfalls wird das Programm beendet.

- `DPI_Unregister()`
  Meldet den zuvor registrierten Teilbaum wieder beim Agenten ab. Auch diese Funktion erwartet die Wurzel des Teilbaums in der globalen Variablen $GROUPID. Die Funktion wartet auf eine positive Antwort vom Hauptagenten und wird dann normal beendet. Tritt ein Fehler auf, oder weigert sich der Agent, die Abmeldung anzunehmen, wird ein DPI_Close() verschickt und das Programm beendet.

- `DPI_Close()`
  Diese Funktion meldet den Sub- beim Hauptagenten ab. Die Funktion erwartet keine Parameter und hat keinen Rückgabewert.

Dieser Funktionsblock darf bei der Implementierung als Black Box betrachet werden. Er wird nur einmal im Modul *subagent_standard.pl* abgearbeitet:

```
# Ermitteln des TCP-Port's, der fuer die DPI-Kommunikation
# mit dem Agenten benutzt werden soll.
$Port = &qDPIport;
print "Port:$Port\n";

# Ueber den ermittelten Port eine TCP-Verbindung zum Agenten
# aufbauen (fuer die DPI-Kommunikation) => Handle: DPI_Anschluss
&TCPConnect($Port);

# Oeffnen der DPI-Verbindung
&DPI_Open;
sleep 1;

# Registrieren des Teilbaums
&DPI_Register;

...

# Unregistrieren des Teilbaums
&DPI_UnRegister;

# Schliessen der DPI-Verbindung
&DPI_Close;

# Schliessen des Handles fuer die TCP-Verbindung zum Agenten
close(DPI_Anschluss);
}
########################################################
# Ende des Hauptprogramms
########################################################
```

Interessanter sind die beiden folgenden Funktionen, die z.B. beim Implementieren eigener, von den generischen Funktionen abweichenden, GET und SET Funktionen benötigt werden:

- `ResponseError()`
  ResponseError verschickt ein Fehlerpaket an den Hauptagenten. Die Funktion erwartet dabei als Parameter die Paket-ID des Pakets, welches den Fehler verursachte und einen Fehlercode nach Tabelle 18 im RFC 1592.

- `ResponseSuccess()`
  Hat der Hauptagent den Wert einer Variablen angefordert, wo wird dieser mittels ResponseSuccess() an den Agenten gesendet. Dabei erwartet die Funktion folgende Parameter:
  - Paket-ID. Dies ist die ID des Anfragepakets, welches beantwortet werden soll.
  - Group-ID. Die ID der Gruppe, in der sich die Variable befindet.
  - Instanz-ID. Die ID der Instanz, also die Bezeichnung, welche die Variable in dem vom Subagent verwalteten Teilbaum besitzt. Hängt man die Instanz-ID an die Gruppen-ID, so erhält man die komplette Object ID in der MIB.
  - Wert der Variable. Hier wird der Wert der Variable in der jeweiligen Darstellung übergeben. Wird in einer Reihe von GETNEXT Requests das Ende der MIB erreicht, so ist der Funktion ResponseSuccess() der Wert *ENDofMIB* zu übergeben.
  - Typ der Variable. Typ der Variable (Nummerncode aus Tabelle 17 in RFC 1592, aber Hex).

Hinweise für die eigene Implementierung:

Es ist wichtig zu beachten, daß das Beantworten von Fehlern z.B. beim GET Zugriff auf nicht vorhandene MIB Variablen mittels ResponseError() SNMPv1 Verhalten ist und nach Möglichkeit `nicht` genutzt werden sollte.
Unter SNMPv2 wird stattdessen ein detaillierter Fehlercode als eigener Typ zurückgeschickt. Die generischen GET und SET Funktionen dieses Subagenten reagieren unterschiedlich, je nachdem, wie die globale Steuervariable *snmpversion* gesetzt ist:

```
...
$Wert = &get($InstID);
if (defined ($Wert)) {
    &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert);
}
else {  # Nicht vorhanden
    if ($$snmpversion == 1) {
        # SNMP_ERROR_noSuchName
        &ResponseError($dpi_get_packet_id,"02")
    }
    else {
        # SNMP_TYPE_noSuchInstance
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert,"10");
    }
}
...
```

Dieses Verhalten sollte auch beim Implementieren eigener GET und SET Funktionen, die Gebrauch von ResponseError() oder ResponseSuccess() machen, modelliert werden.

Weiterhin ist zu beachten, daß nur die folgenden SNMP Typen für ein ResponseSuccess() implementiert sind:

- SNMP_TYPE_OCTET_STRING
- SNMP_TYPE_OBJECT_IDENTIFIER
- SNMP_TYPE_endOfMibView
- SNMP_TYPE_Integer32
- SNMP_TYPE_noSuchObject
- SNMP_TYPE_noSuchInstance

Die Funktion ResponseSuccess() läßt sich aber leicht um weitere Typen erweitern, wenn man die if-then Schleife entsprechend erweitert. Die Konstruktion eines entsprechenden Packstrings, also die Kodierung des neuen Typen findet man im RFC 1592.

4.4 High Level Funktionen für Skalar- und Tabellenvariablen

Nachdem die eingehenden DPI Requests dekodiert wurden, werden die entsprechenden High Level Funktionen ausgeführt. Diese sind ausschließlich in den Moduln *mib_gets.pl*, *mib_sets.pl* sowie *sub_gets.pl*, *sub_sets.pl*, *sub_coms.pl* und *sub_undos.pl* enthalten.

Die ersten beiden Moduln enthalten kaskadierte if-then-else Schleifen, in denen die InstanzID, also die MIB-Variable ermittelt wird:

```
...
elsif ($InstID =~ /^1\.1\.1\.1/)    # trapSourceID
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.1\.1\.2/)    # trapSourceTableStatus
    {
    &do_get ($InstID);
    }
...
```

Dieses Vorgehen erlaubt es, für jede MIB Variable eine separate GET Prozedur zu definieren.

Das Modul *mib_sets.pl* ist identisch aufgebaut, es wird allerdings noch zusätzlich nach der Art des Requests (SET/COMMIT/UNDO) unterschieden:

```
elsif ($InstID =~ /^1\.1\.1\.2/)
    {
    if ($Art eq "03")            # Set - Request
        {
```

```
        $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
        }
    elsif ( $Art eq "0a")        # Commit - Request
        {
        $Fehler = &com($InstID,2,5)  # Commit-Request
        }
    elsif ( $Art eq "0b")        # Undo - Request
        {
        $Fehler = &undo($InstID);
        }
    if(!$Fehler)
        {
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
        }
    else
        {
        &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
        }
    }
```

Die Auskodierung der jeweiligen Funktionen befinden sich in den entsprechenden *sub_*.pl* Moduln. Dabei stehen folgende Funktionen zur Verfügung:

- `get()`
  Generische GET Funktion für Skalarvariablen und Tabelleneinträge. Die Funktion erwartet als Parameter die Instanz-ID der Variablen und gibt den Wert der Variablen zurück. Existiert die angeforderte Variable nicht, so wird undef zurückgegeben.

- `get_inc()`
  GET Funktion für das Anfordern von Indexvariablen. Parameter und Ergebnisse wie oben, aber die Funktion erhöht den Wert der angefragten Variablen um 1.

- `set()`
  Generische SET Funktion für Skalarvariablen. Die Funktion erwartet als Parameter die Instanz-ID der zu setzenden Variablen, den Typ der Variablen, die Länge des Setwertes in Bytes (wird automatisch beim Empfang des DPI Pakets ermittelt und wird nur durchgereicht) und natürlich die Variable selbst. Bei Erfolg gibt diese Funktion den Wert undef zurück, andernfalls den SNMP Error Code.

- `set_Tabelle()`
  Generische SET Funktion zum Setzen von Variablen in Tabellenzeilen. Die Funktion erledigt auch das Handling der Tabellenstatusspalte, wie im *Simple Book* [1] beschrieben. Als Parameter werden die Instanz-ID, der Variablentyp, die Variablenlänge in Byte, die Nummer der Spalte, die als Status- spalte dient, die Anzahl der Tabellenspalten insgesamt und wiederum die Variable selbst erwartet. Die Funktion gibt bei fehlerfreier Ausführung den Wert undef zurück. Bei Auftreten eines Fehlers wird ein SNMP Error Code entsprechend [1] zurückgegeben.

- `undo()`
  Generische UNDO Funktion. Erwartet als Parameter die Instanz-ID der Variablen und setzt sie auf ihren vorherigen Wert zurück. Gibt als Wert immer undef zurück.

- `com()`
  Generische COMMIT Funktion für Skalarvariablen und Tabellen. Die Funktion erwartet als Parameter die Instanz-ID der zu setzenden Variablen und, falls es sich um eine Variable einer Tabellenzeile handelt, zusätzlich noch die Nummer der Statusspalte und die Anzahl der Spalten. Bei erfolgreicher Ausführung gibt die Funktion undef zurück, ansonsten einen SNMP Fehlercode.

Kapitel 5

Zusammenfassung und offene Probleme

5.1 Zusammenfassung

Zusammenfassend möchte ich erwähnen, daß mir dieses Fopra einen guten Überblick über das Internet Management mit allen seinen Vorteilen, aber auch ungelösten Problemen gab. Es gab mir außerdem einen ersten Einblick in die Implementierung einfacher Protokolle, wie z.B. DPI und SNMP.

Ein interessanter Aspekt dieser Arbeit war PERL 5. Da ich bisher nur in in Hochsprachen programmiert hatte, die ein einigermaßen sauberes Arbeiten erzwingen, stand ich der Sprache anfangs skeptisch gegenüber. Ich stellte aber fest, daß sich mit PERL schnelle Datenstrukturen implementieren lassen, die in anderen Sprachen witaus komplizierter auszuprogrammieren sind. Ich stellte aber auch fest, daß PERL sehr schnell zu unübersichtlichem Programmieren verleiten kann. Inwieweit umfangreiche Projekte fehlerfrei in PERL zu verwirklichen sind, hängt deshalb stark von der Disziplin des Programmieres ab.

Eine wünschenswerte Erweiterung für den Bau weiterer Subagenten wäre die automatische Generierung von zumindest weiten Teilen des Codes: durch den formal exakten Aufbau einer MIB und die starke Anlehung der Codestruktur der *mib_*.pl* Moduln daran, drängt sich dieser Ansatz geradezu auf.

5.2 Offene Probleme

Die DPI Spezifikation läßt leider einige Probleme ungelöst, die beim täglichen Arbeiten durchaus zu Problemen führen können:

- `Tabellenhandling`
  SNMP Pakete können eine große Anzahl von Variable Bindings enthalten. Der DPI Subagent nimmt immer nur ein Paar Variablenname/Variablenwert entgegen. Enthält nun eine SNMP PDU ein Set auf einen Tabellenwert und die zugehörige Statusspalte, so muß der Agent den SET auf den Tabellenwert unbedingt vor dem Set auf die Statusspalte behandeln. Dazu muß der Agent die Reihenfolge der Variable Bindings gegebenfalls ändern.

Das Problem ist nun, daß der Hauptagent die MIB des Subagenten und damit die Semantik hinter den Daten (Tabelle) gar nicht kennt und die Variable Bindings in der angekommenen Reihenfolge an den Subagenten weitergibt. Der Subagent kennt wiederum die PDU als ganzes nicht und weiß nie, ob den gerade angelieferten Daten noch weitere folgen. Er kann also eine Änderung der Statusspalte nicht zurückhalten.
Dies führt in der Praxis beim Set auf Tabellenspalten zu SNMP Fehlern, es sei denn, der Manager verschickt pro Paket nur ein Variablenname/Variablenwert Paar. Selbst dann könnte es noch zu Fehlern kommen, wenn die Pakete nicht in der Sendereihenfolge ankommen. Dies ist jedoch kein Problem von DPI, sondern der verbindungslosen UDP Kommunikation zwischen Manager und Agent.

- `Trapkommunikation`
  Ursprünglich war es für den Syslog Subagenten vorgesehen, dem Benutzer eine Konfiguration der Trapziele zu erlauben. Leider sieht die DPI Spezifikation keine Übermittlung dieser Daten an den Hauptagenten vor. Der Benutzer muß deshalb die Trapziele in einer Konfigurationsdatei auf dem Agentenhost eintragen. Das Senden unterschiedlicher Traps an unterschiedliche Ziele ist gar nicht möglich.
  Eine standardisierte Trap MIB zur Lösung dieser Probleme befindet sich derzeit erst im Draft Status.

- `Sicherheitsproblem`
  Leider sieht das DPI Protokoll keine Zugangsbeschränkung zum Hauptagenten und keinerlei Authentizifierungsmechanismen vor, d.h. es können jederzeit unbekannte Subagenten am Hauptagenten andocken. Damit besteht zumindest die Gefahr eines Denial-of-service Attacks, d.h. ein böswilliger Subagent könnte das Netz mit SNMP Traps überfluten und verstopfen oder zumindest Irritation bei den Netzbetreuern verursachen. Da DPI Kommunikation auf Ports größer 1024 stattfindet, könnte so ein Angriff eventuell sogar von außen kommen, ohne von einem Paketfilter abgefangen zu werden.

Literaturverzeichnis

[1] Marshall T. Rose *The Simple Book. Revised Second Edition* Prentice Hall, Inc., 1996.

[2] B. Wijnen, G. Carpenter, K. Curren, A. Sehgal, and G. Waters. Simple network management protocol distributed protocol interface version 2.0. Rfc1592, T.J. Watson Research Center, IBM Corp., Bell Northern Research, 1994.

[3] Frank Schütz. Implemetierung eines Werkzeugs zur Konsistenzprüfung von HTML-Links. Fortgeschrittenenpraktikum, TU München, 1996.

[4] Randal L. Schwartz. *Learning PERL.* O'Reilly & Associates, Inc., 1993.

[5] Larry Wall and Randal L. Schwartz. *Programming PERL. Revised 2nd Edition.* O'Reilly & Associates, Inc., 1996.

[6] Helmut Kopka and Patrick W. Daly. *A Guide to LaTex.* Addison-Wesley Co. Inc., United States of America, 1993.

Anhang A

Die Syslog MIB

```
SYSLOG-MGMT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Integer32
        FROM SNMPv2-SMI
    RowStatus, DisplayString
        FROM SNMPv2-TC;

syslogMIB MODULE-IDENTITY
    LAST-UPDATED "9706140000Z"
    ORGANIZATION "MNM Team Munich"
    CONTACT-INFO "      Christian Coehn
                coehn@informatik.uni-muenchen.de"
    DESCRIPTION  "Management des Syslos Daemons"
    ::= { iso 3 6 1 3 100 8 1 }

syslogMIBObjects OBJECT IDENTIFIER
    ::= { syslogMIB 1 }

-- DIE MIB ist in 4 Gruppen organisiert:
-- 1. Trapkonfiguration
-- 2. Traphistory
-- 3. General
-- 4. Syslog Konfiguration

-- OIDs fuer die obigen Gruppen:

trapconfig OBJECT IDENTIFIER
    ::= { syslogMIBObjects 1 }
traphistory OBJECT IDENTIFIER
    ::= { syslogMIBObjects 2 }
general OBJECT IDENTIFIER
    ::= { syslogMIBObjects 3 }
```

```
syslogconf OBJECT IDENTIFIER
    ::= { syslogMIBObjects 4 }


-- Die trapconfig Gruppe

trapSourceTable OBJECT-TYPE
    SYNTAX SEQUENCE OF TrapSourceTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Die (konzeptionelle) Tabelle zur Erfassung der Trap Ausloeser"
    ::= { trapconfig 1 }


trapSourceTableEntry OBJECT-TYPE
    SYNTAX TrapSourceTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Ein Eintrag (konzeptionelle Reihe) in der trapSourceTable
         Tabelle."
    INDEX { trapSourceID }
    ::= { trapSourceTable 1 }

TrapSourceTableEntry ::=
    SEQUENCE {
        trapSourceID        Integer32,
        trapSourceTableStatus   RowStatus,
        facility        DisplayString,
        severity        DisplayString,
        logstring       DisplayString
    }

trapSourceID OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Gibt an, um die wievielte Instanz der obigen konzeptionellen
         Reihe es sich handelt. Der Wert ist der Primaerschluessel
         dieser Tabelle und verknuepft einen Eintrag auf eindeutige Weise
         mit einem oder mehreren Eintraegen in der Tabelle trapDestTable
         ueber den dortigen Eintrag trapDestID. Der Manager muss sich diesen
         Wert aus der Variablen trapSourceTableNextInstance holen."
    ::= { trapSourceTableEntry  1 }
```

```
trapSourceTableStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "SNMPv2 Hilfsobjekt zum Manipulieren konzeptioneller Zeilen."
    ::= { trapSourceTableEntry 2 }

facility OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Gibt die Syslog Facility an, die, falls auch severity und logstring
        uebereinstimmen, einen Trap ausloesen soll. Bleibt dieser Eintrag leer,
        so loest jede Facility einen Trap aus.
        Beispiele: AUTH, AUTHPRIV, CRON, DAEMON,..."
    ::= { trapSourceTableEntry 3 }

severity OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Gibt die Syslog Severity an, die, falls auch facility und logstring
        uebereinstimmen, einen Trap ausloesen soll. Bleibt dieser Eintrag leer,
        so loest jede Severity einen Trap aus.
        Beispiele: DEBUG, INFO, NOTICE, WARNING,..."
    ::= { trapSourceTableEntry 4 }

logstring OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Enthaelt einen regulaeren Ausdruck, der, falls dieser Ausdruck den
         an den Syslog Daemon geschickten String matcht und falls auch facility
         und Severity uebereinstimmen, einen Trap ausloest. Bleibt dieser Eintrag
         leer, so wird er wie der regulaere Ausruck '*' behandelt."
    ::= { trapSourceTableEntry 5 }


trapSourceTableNextInstance OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
```

```
        "Hilfsvariable fuer den Manager, um die naechste Instanz zu erfragen."
        ::= { trapconfig 2 }


trapDestTable OBJECT-TYPE
    SYNTAX SEQUENCE OF TrapDestTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Die (konzeptionelle) Tabelle zur Erfassung der Trap Ziele"
    ::= { trapconfig 3 }


trapDestTableEntry OBJECT-TYPE
    SYNTAX TrapDestTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Ein Eintrag (konzeptionelle Reihe) in der trapDestTable
         Tabelle."
    INDEX { trapDestIndex }
    ::= { trapDestTable 1 }

TrapDestTableEntry ::=
    SEQUENCE {
        trapDestIndex   Integer32,
        trapDestID      Integer32,
        trapDestVal     Integer32,
        trapDestTableStatus RowStatus
    }

trapDestIndex OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Hilfsvariable, als Index (Zeilennummer der Tabelle)
         verwendet."
    ::= { trapDestTableEntry 1 }

trapDestID OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Kennzeichnet einen Eintrag der Tabelle trapSourceTable.
         Wird durch einen dort angegeben Eintrag ein Trap generiert,
```

```
        so wird dieser an alle Ziele aus trapDestTable geschickt,
        fuer die gilt: trapSourceID == trapDestID."
    ::= { trapDestTableEntry 2 }

trapDestVal OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Frei definierbarer Wert, der mit dem Trap mitgeschickt wird."
    ::= { trapDestTableEntry 3 }

trapDestTableStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "SNMPv2 Hilfobjekt zum Manipulieren konzeptioneller Zeilen."
    ::= { trapDestTableEntry 4 }

trapDestTableNextInstance OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Hilfsvariable fuer den Manager, um die naechste Instanz zu erfragen."
        ::= { trapconfig 4 }


-- Die Trap History Gruppe

historyMaxAge OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Maximales Alter der Eintraege in der Trap History in Minuten"
    ::= { traphistory 1 }

historyMaxEntries OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Maximale Anzahl an Eintraegen in der Trap History Tabelle"
    ::= { traphistory 2 }
```

```
trapHistoryTable OBJECT-TYPE
    SYNTAX SEQUENCE OF TrapHistoryTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Tabelle der bisher erzeugten Traps (dient ausschliesslich
         als Log, d.h. read-only)"
    ::= { traphistory 3 }

trapHistoryTableEntry OBJECT-TYPE
    SYNTAX TrapHistoryTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Ein Eintrag (konzeptionelle Zeile) der trapHistoryTable
         Tabelle."
    INDEX { trapNummer }
    ::= { trapHistoryTable 1 }

TrapHistoryTableEntry ::=
    SEQUENCE {
        trapNummer  Integer32,
        trapID      Integer32,
        trapZeit    DisplayString
    }

trapNummer OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Hilfsvariable, als Index (Zeilennummer der Tabelle)
         verwendet. Fuer Manager read-only zugaenglich."
    ::= { trapHistoryTableEntry 1 }

trapID OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Entspricht dem Primaerschluessel der Tabelle trapSourceTable
         und identifitiert auf diese Weise den Trap Ausloeser eindeutig."
    ::= { trapHistoryTableEntry 2 }

trapZeit OBJECT-TYPE
    SYNTAX DisplayString
```

```
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Datum und Zeit, zu der der Trap verschickt wurde.
         Aufloesung: 1/10 Sekunde."
    ::= { trapHistoryTableEntry 3 }

-- Die Gruppe General

starttime OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "Startzeipunkt des Agenten"
    ::= { general 1 }

steuerung OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Steuervariable, steht beim Start des Subagenten auf 1, zum
         Beenden des Subagenten auf 0 setzen."
    ::= { general 2 }

snmpversion OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Steuervariable, die das Verhalten des Subagenten bei bestimmten
         Fehlermeldungen (z.B. no such instance) bestimmt. Wird die
         Variable auf 1 gesetzt, werden SNMPv1 Fehler generiert, wird
         die Variable auf 2 gesetzt, werden SNMPv2 Fehler generiert.
         Die Variable hat keinen Einfluss darauf, ob der Hauptagent
         SNMPv1 oder SNMPv2 als Protokoll benutzt.
         Default beim Start des Subagenten ist SNMPv2."
    ::= { general 3 }


-- Die Gruppe Syslog Konfiguration


syslogConfTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SyslogConfTableEntry
    MAX-ACCESS not-accessible
    STATUS current
```

```
    DESCRIPTION
        "Beim Start des Agenten wird die Syslog Konfiguration
         syslog.conf in diese Tabelle gelesen. Eine Zeile im
         syslog.conf entspricht dabei einer Zeile der Tabelle.
         Bei Aenderungen in der Tabelle wird die Konfigurations-
         datei neu geschrieben und der Syslog Daemon per SIGHUP
         Kill zum Erneuten Einlesen der Konfiguration gebracht."
    ::= { syslogconf 1 }

syslogConfTableEntry OBJECT-TYPE
    SYNTAX SyslogConfTableEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Ein Eintrag (konzeptionelle Zeile) der syslogConfTable
         Tabelle."
    INDEX { syslogZeilenNummer }
    ::= { syslogConfTable 1 }

SyslogConfTableEntry ::=
    SEQUENCE {
        syslogZeilenNummer Integer32,
        syslogZeile     DisplayString,
        syslogConfTableStatus RowStatus
    }


syslogZeilenNummer OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Hilfsvariable, als Index (Zeilennummer der Tabelle)
         verwendet. Fuer Manager read-only zugaenglich."
    ::= { syslogConfTableEntry 1 }

syslogZeile OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "String, der einer Zeile im syslog.conf entspricht."
    ::= { syslogConfTableEntry 2 }

syslogConfTableStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
```

```
    STATUS current
    DESCRIPTION
        "SNMPv2 Hilfsobject zum Manipulieren konzeptioneller Zeilen"
    ::= { syslogConfTableEntry 3 }

syslogConfTableNextInstance OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Hilfsvariable fuer den Manager, um die naechste Instanz zu erfragen."
    ::= { syslogconf 2 }

END
```

Anhang B

Listings der Programme

B.1 syslogtrapd

```
#!/client/bin/perl
#!/soft/bin/perl
require 5.002;

require "sub_gets.pl";
require "sub_sets.pl";
require "sub_undos.pl";
require "sub_coms.pl";
require "mib_gets.pl";
require "mib_sets.pl";
require "support.pl";
require "handle_syslog.pl";
require "subagent_standard.pl";

######################################################
######################################################
##            Begin User Configuration             ##
##                                                 ##
## Die folgenden Werte sollten vor dem Start des   ##
## Agenten kontrolliert bzw. angepat werden       ##
######################################################
######################################################

# Konstanten fuer die SNMP-Kommunikation zum Agenten
$SNMP_PORT = 161 ;
$SNMP_TRAP_PORT = 162 ;
$SNMP_COMMUNITY = "public" ;
$SNMP_TRAP_COMMUNITY = "public";

# Hier die Wurzel des zu verwaltenden Teilbaums eintragen
# DPI Group ID = SNMP OID mit Punkt (".") angehaengt
```

```
$GROUPID    = "1.3.6.1.3.100.8.1.";
$OID        = "1.3.6.1.3.100.8.1";
$DESCRIPTION    = "syslogSubAgent";

# Host, auf dem der DPI Hauptagent laeuft
# Override ist durch Kommandozeilenparameter '-h host' moeglich
$AGENT_HOST = "hphegerb.nm.informatik.uni-muenchen.de";

# Bildschirmausgabe/Debug Level
# 0 = kein Debug, 1 = Errors, 2 += Warnings, 3 += Informational
$DEBUGLEVEL = 2;

# Pfad zum Log Pipe/Socket (Erkennung automatisch)
$SYSLOGPIPESOCK = '/dev/log';

# Falls der syslogd des Systems mittels Parameter zum Erzeugen eines
# anderen Unix Domain Sockets als /dev/log konfiguriert werden kann,
# so ist hier dieser alternative Socket anzugeben. Der Subagent reicht
# dann alle von $SYSLOGPIPESOCK gelesenen Nachrichten an $SYSLOGSOCKNEW
# durch.
# Das Durchreichen der Nachrichten unterbleibt, wenn die Variable nicht
# definiert wird.
# Die Variable wird ignoriert, wenn $SYSLOGPIPESOCK eine Named Pipe ist.
$SYSLOGSOCKNEW = '/dev/log.syslog';

# Directory, in dem der Subagent selbst liegt
$ProgPfad  = '/usr/stud/coehn/fopra5';

# Die Konfigurationsdatei des Syslog Daemons
$SyslogConfPfad = '/etc/syslog.conf';

# Datei, in die die SyslogD Konfigurationsdatei bei Aenderungen
# zurueckgeschrieben wird (fuer Testzwecke). Wird die Variable nicht
# definiert, so wird $SyslogConfPfad ueberschrieben
$SyslogConfPfadWrite = '/tmp/syslog.conf';

#######################################################
#######################################################
##              End User Configuration              ##
##                                                   ##
##            Ab hier nichts mehr aendern!           ##
#######################################################
#######################################################



#######################################################
```

```
# Definition einiger Konstanten
#######################################################
# Konstanten fuer den Socket
$AF_INET = 2;
$SOCK_STREAM = 1;
$SOCK_DGRAM  = 2;
$SOCKADDR = 'S n a4 x8';
#Konstanten fuers Tabellenhandling (RowStatus)
$RS_ACTIVE = 1;
$RS_NOTINSERVICE = 2;
$RS_NOTREADY = 3;
$RS_CREATEANDGO = 4;
$RS_CREATEANDWAIT = 5;
$RS_DESTROY = 6;

########################################################
# Definition globaler Variablen
########################################################
$TempStatusColumn = undef;
$anz_traps_triggered = 0;

########################################################
# Initialisierungsroutine fuer die MIB-Variablen, die
# der Subagent anbietet.
########################################################
sub initial_MIB
{
    $MIB{'1.1.1.1.0'} = "81";    # trapSourceID
    $MIB{'1.1.1.2.0'} = "81";    # trapSourceTableStatus
    $MIB{'1.1.1.3.0'} = "02";    # facility
    $MIB{'1.1.1.4.0'} = "02";    # severity
    $MIB{'1.1.1.5.0'} = "02";    # logstring
    $MIB{'1.2.0'}     = "81";    # trapSourceTableNextInstance

    $MIB{'1.3.1.1.0'} = "81";    # trapDestIndex
    $MIB{'1.3.1.2.0'} = "81";    # trapDestID
    $MIB{'1.3.1.3.0'} = "81";    # trapDestVal
    $MIB{'1.3.1.4.0'} = "81";    # trapDestTableStatus
    $MIB{'1.4.0'}     = "81";    # trapDestTableNextInstance

    $MIB{'2.1.0'}     = "81";    # historyMaxAge
    $MIB{'2.2.0'}     = "81";    # historyMaxEntries
    $MIB{'2.3.1.1.0'} = "81";    # trapNummer
    $MIB{'2.3.1.2.0'} = "81";    # trapID
    $MIB{'2.3.1.3.0'} = "02";    # trapZeit

    $MIB{'3.1.0'}     = "02";    # starttime
```

```
    $MIB{'3.2.0'}     = "81";    # steuerung
    $MIB{'3.3.0'}     = "81";    # snmpversion

    $MIB{'4.1.1.1.0'} = "81";    # syslogZeilenNummer
    $MIB{'4.1.1.2.0'} = "02";    # syslogZeile
    $MIB{'4.1.1.3.0'} = "81";    # syslogConfTableStatus
    $MIB{'4.2.0'}     = "81";    # syslogConfTableNextInstance
    @MIB = sort by_hierarchy keys(%MIB);    # sortierte Liste der MIB-Variablen
}

########################################################
# Vorbelegung von Variablen
########################################################
$Wert{'1.2.0'} = 0;                   # trapSourceTableNextInstance
$WertTemp{'1.2.0'} = $Wert{'1.2.0'}; # Temporaeren Wert identisch belegen
$Wert{'1.4.0'} = 0;                   # trapDestTableNextInstance
$WertTemp{'1.4.0'} = $Wert{'1.4.0'}; # Temporaeren Wert identisch belegen
$Wert{'2.1.0'} = 60;                  # historyMaxAge (Minuten)
$WertTemp{'2.1.0'} = $Wert{2.1.0};   # Temporaeren Wert identisch belegen
$Wert{'2.2.0'} = 100;                 # historyMaxEntries
$WertTemp{'2.2.0'} = $Wert{2.2.0};   # Temporaeren Wert identisch belegen
$start_zeit = time;
$Wert{'3.1.0'} = localtime(time);    # Startzeit des Agenten als TimeStamp
$WertTemp{'3.1.0'} = $Wert{'3.1.0'}; # Temporaeren Wert identisch belegen
$Wert{'3.2.0'} = 1;                   # 1=running, 0=shutdown
$WertTemp{'3.2.0'} = $Wert{'3.2.0'}; # Temporaeren Wert identisch belegen
$Wert{'3.3.0'} = 2;                   # 1=SNMPv1, 2=SNMPv2
$snmpversion = \$Wert{'3.3.0'};      # Referenz zur einfacheren Benutzung
$WertTemp{'3.3.0'} = $Wert{'3.3.0'}; # Temporaeren Wert identisch belegen
$Wert{'4.2.0'} = 0;                   # syslogConfTableNextInstance
$WertTemp{'4.2.0'} = $Wert{'4.2.0'}; # Temporaeren Wert identisch belegen

########################################################
# Einlesen von syslog.conf
########################################################
if (!open (SYSLOGCONF, $SyslogConfPfad)) {
    print "Warnung - konnte $SyslogConfPfad nicht oeffnen!";
}
else {
    $i = 0;
    while (<SYSLOGCONF>) {
        chop;
    $Wert{'4.1.1.1.'.$i} = $i;
        $MIB{'4.1.1.1.'.$i} = "81";
    $Wert{'4.1.1.2.'.$i} = $_;
        $MIB{'4.1.1.2.'.$i} = "02";
        $Wert{'4.1.1.3.'.$i} = $RS_ACTIVE;
```

```
        $MIB{'4.1.1.3.'.$i} = "81";
        $i += 1;
    }
    close (SYSLOGCONF);
}

# Check, ob Dumpfiles einer alten Config vorliegen
my ($ke, $va);
if (open (MIBDUMP, "mib_dump.txt") and
    open (WERTDUMP, "wert_dump.txt") and
    open (WERTDEF, "wert_def.txt")) {
    print "Rereading previously dumped config...\n";
    while (<MIBDUMP>) {
        chop;
        ($ke, $va) = split /\//, $_, 2;
        $MIB{$ke} = $va;
    }
    while (<WERTDUMP>) {
        chop;
        ($ke, $va) = split /\//, $_, 2;
        $Wert{$ke} = $va;
    }
    while (<WERTDEF>) {
        chop;
        ($ke, $va) = split /\//, $_, 2;
        $WertDef{$ke} = $va;
    }
    close (MIBDUMP);
    close (WERTDUMP);
    close (WERTDEF);
}

# Liste der Keys sortieren
@MIB = sort by_hierarchy keys(%MIB);    # sortierte Liste der MIB-Variablen

# Signal Handler fuer INT und TERM auf eigene
# Funktion (Config Backup) umbiegen
$SIG{INT} = \&catch_term_int;
$SIG{TERM} = \&catch_term_int;

# Kommandozeile auslesen...
$dbforeground = 0;
my $agclhost;
my $pid;
while ($ARGV = shift) {
    if ($ARGV eq "-h" or $ARGV eq "-H") {
        $agclhost = shift;
```

```
    }
    if ($ARGV eq "-d" or $ARGV eq "-D") {
        $dbforeground = 1;
    }
}

# ...und auswerten
if ($agclhost) {
    $AGENT_HOST = $agclhost;  # AGENT_HOST command line override
}
if ($dbforeground) {
    &Subagent;
} else {
    print "Agent running in background...\n";
    if ($pid = fork) {
        exit 0;
    } elsif (defined $pid) {
        &Subagent;
    } else {
        die "Can't fork: $!\nTerminating.\n";
    }
}
```

B.2 subagent_standard.pl

```
# snmp - Subagent in Perl implementiert

sub Subagent
{

########################################################
# Beginn des Hauptprogramms
########################################################

&initial_MIB;   # Intialisiere die MIB fuer den Subagenten

# Ermitteln des TCP-Port's, der fuer die DPI-Kommunikation
# mit dem Agenten benutzt werden soll.
$Port = &qDPIport;
&print_debug (0, "Port:$Port");

# Ueber den ermittelten Port eine TCP-Verbindung zum Agenten
# aufbauen (fuer die DPI-Kommunikation) => Handle: DPI_Anschluss
&TCPConnect($Port);

# Oeffnen der DPI-Verbindung
```

```
&DPI_Open;
sleep 1;

# Registrieren des Teilbaums
&DPI_Register;

# Registrieren des DPI Handlers beim Event Dispatcher
&print_debug (3, "Registriere DPI_Handler beim Dispatcher");
&dispatch_addfunc (fileno(DPI_Anschluss), \&DPI_Handler);

# Oeffnen des syslogd pipe/socket
&open_logger();

# Registrieren des Syslog Handlers beim Event Dispatcher
&print_debug (3, "Registriere Syslog Handler beim Dispatcher");
&dispatch_addfunc (fileno(SL_HANDLE), \&read_logger);

#---------------------------------------------------------------
# Warten auf eine Anfrage
while(!($Wert{'3.2.0'} == 0)) {   # Warten auf DPI-Anfragen des Agenten
    &dispatch_now (30);
}
#---------------------------------------------------------------
# Save config
&print_debug (1, "Saving config...");
&dump_config_hashes();

# Unregistrieren des Teilbaums
&DPI_UnRegister;

# Schliessen der DPI-Verbindung
&DPI_Close;

# Schliessen des Handles fuer die TCP-Verbindung zum Agenten
close(DPI_Anschluss);
}
########################################################
# Ende des Hauptprogramms
########################################################

########################################################
# DPI Request Handler
########################################################
sub DPI_Handler {
    recv(DPI_Anschluss,$Laenge,2,0);
    $Laenge = unpack("n",$Laenge);
    if (!defined $Laenge or $Laenge == 0) {
```

```
     die "Error in DPI Kommunikation, Hauptagent down?\nTerminating";
    }
    recv(DPI_Anschluss,$dpi_packet,$Laenge,0);
    @dpi_header = unpack("H2H2H2nH2",$dpi_packet);  # Header auslesen

    if ($dpi_header[4] eq "01") # Falls get-Request
        {
        &print_debug (0, "Get-Request empfangen");
        &answer_get;        # Beantworte get-Request
        }
    elsif ($dpi_header[4] eq "02")  # falls getnext-Request
        {
        &print_debug (0, "Getnext-Request empfangen");
        &answer_getnext;    # Beantworte getnext-Request
        }
    elsif ($dpi_header[4] eq "03" || $dpi_header[4] eq "0a" || $dpi_header[4] eq "0b")
        {   # falls set-Request Gruppe
        &answer_setgrp($dpi_header[4]);
        }
    elsif ($dpi_header[4] eq "05")  # falls Response
        {
        &print_debug (2, "Response empfangen");
        next;
        }
    else
        {
        &print_debug (0, "Undefinierbare Nachricht empfangen");
        &print_debug (0, "gemeldeter Packettyp: $dpi_header[5]");
        }
}

#############################################################
# answer_setgrp(Art)
# Zerlegen des jeweiligen Packets und wenn moeglich Beantwortung
# Parameter: Art,   Set ="03", Commit="0a", Undo="0b"
#############################################################
sub answer_setgrp
    {
    # Uebernahme der Parameter
    local($Art)=@_;
    # Original
    # Lokale Variablen
    #local($GID,$GID_Laenge,$InstID,$InstID_Laenge,$WertTemp);
    # Neu: echte lokale Variblen durch my
    my ($GID,$GID_Laenge,$InstID,$InstID_Laenge,$WertTemp);
    my (@dpi_get, @Set_Wert);
    @dpi_get = unpack("H2H2H2nH2n",$dpi_packet);
```

```
    $dpi_get_packet_id = $dpi_get[3]; # Fuer Antwort merken
    $ComNameLaenge = $dpi_get[5];
    if ($ComNameLaenge) # Falls ein Community-Name vorhanden
        {
        $Position = 7;  # Beginn der GID im Array
        $UnpackString = "H2H2H2nH2nA$dpi_get[5]"."H2" x ($Laenge-8-$ComNameLaenge);
        }
    else            # Falls kein Community-Name vorhanden
        {
        $Position = 6;  # Beginn der GID im Array
        $UnpackString = "H2H2H2nH2n"."H2" x ($Laenge-8);
        }

    @dpi_get = unpack($UnpackString,$dpi_packet);
    while (!($dpi_get[$Position] eq "00")) # GID ermitteln (vorerst hex)
        {
        $GID_Laenge++;
        $NewGID = unpack("A",pack("H2",$dpi_get[$Position]));
        $GID = $GID.$NewGID;
        $Position++;
        }
    $Position++;
    while (!($dpi_get[$Position] eq "00"))
        {
        $InstID_Laenge++;
        $NewInstID = unpack("A",pack("H2",$dpi_get[$Position]));
        $InstID = $InstID.$NewInstID;
        $Position++;
        }
    $Position++;
    $Set_Typ = $dpi_get[$Position];
    $Position++;
    $Set_Laenge = (hex($dpi_get[$Position]))*16 + hex($dpi_get[$Position+1]);
    $Position = $Position + 2;
    for ($i = $Position; $i <= @dpi_get; $i++)
        {
        $Set_Wert[$i-$Position] = $dpi_get[$i];
        }

    &answer_set2($GID,$InstID,$Art,$dpi_get_packet_id,$Set_Typ,$Set_Laenge,@Set_Wert);
    }

############################################################
# answer_get
# Zerlegen des get-Packets und wenn moeglich Beantwortung
############################################################
sub answer_get
```

```
    {
    # Lokale Variablen
    local($GID,$GID_Laenge,$InstID,$InstID_Laenge);

    @dpi_get = unpack("H2H2H2nH2n",$dpi_packet);
    $dpi_get_packet_id = $dpi_get[3]; # Fuer Antwort merken
    $ComNameLaenge = $dpi_get[5];
    if ($ComNameLaenge) # Falls ein Community-Name vorhanden
        {
        $Position = 7;  # Beginn der GID im Array
        $UnpackString = "H2H2H2nH2nA$dpi_get[5]"."H2" x ($Laenge-8-$ComNameLaenge);
        }
    else            # Falls kein Community-Name vorhanden
        {
        $Position = 6;  # Beginn der GID im Array
        $UnpackString = "H2H2H2nH2n"."H2" x ($Laenge-8);
        }

    @dpi_get = unpack($UnpackString,$dpi_packet);
    while (!($dpi_get[$Position] eq "00")) # GID ermitteln (vorerst hex)
        {
        $GID_Laenge++;
        $NewGID = unpack("A",pack("H2",$dpi_get[$Position]));
        $GID = $GID.$NewGID;
        $Position++;
        }
    $Position++;
    while (!($dpi_get[$Position] eq "00"))
        {
        $InstID_Laenge++;
        $NewInstID = unpack("A",pack("H2",$dpi_get[$Position]));
        $InstID = $InstID.$NewInstID;
        $Position++;
        }
    &print_debug (1, $GID."/".$InstID);
    #print "$InstID\n";
    &answer_get2($GID,$InstID,$dpi_get_packet_id);  # Auf ermittelte Anfrage antworten
    }

############################################################
# answer_getnext
# Zerlegen des getnext-Packets und wenn moeglich Beantwortung
############################################################
sub answer_getnext
    {
    # Lokale Variablen
    local($GID,$GID_Laenge,$InstID,$InstID_Laenge,$MIBInst);
```

```
@dpi_get = unpack("H2H2H2nH2n",$dpi_packet);
$dpi_get_packet_id = $dpi_get[3]; # Fuer Antwort merken
$ComNameLaenge = $dpi_get[5];
if ($ComNameLaenge) # Falls ein Community-Name vorhanden
    {
    $Position = 7;  # Beginn der GID im Array
    $UnpackString = "H2H2H2nH2nA$dpi_get[5]"."H2" x ($Laenge-8-$ComNameLaenge);
    }
else            # Falls kein Community-Name vorhanden
    {
    $Position = 6;  # Beginn der GID im Array
    $UnpackString = "H2H2H2nH2n"."H2" x ($Laenge-8);
    }

@dpi_get = unpack($UnpackString,$dpi_packet);
while (!($dpi_get[$Position] eq "00")) # GID ermitteln (vorerst hex)
    {
    $GID_Laenge++;
    $NewGID = unpack("A",pack("H2",$dpi_get[$Position]));
    $GID = $GID.$NewGID;
    $Position++;
    }
$Position++;
while (!($dpi_get[$Position] eq "00"))
    {
    $InstID_Laenge++;
    $NewInstID = unpack("A",pack("H2",$dpi_get[$Position]));
    $InstID = $InstID.$NewInstID;
    $Position++;
    }
&print_debug (3, $GID."/".$InstID);
#print "   $InstID\n";

#################################################################
# Gesuchten Wert ermitteln
if (!($GID eq $GROUPID))    # Falls der Subagent fuer diese Group-ID
    {           # nicht zustaendig
    &ResponseError($dpi_get_packet_id,"05") # Allgemeiner Fehler
    }

$NextID = "";

foreach $MIBInst (@MIB)
    {
    &print_debug (3, "$MIBInst");
if ((&less_by_hierarchy($MIBInst,$InstID) <= 0) || !defined($Wert{$MIBInst}))
```

```
            {
            next;
            }

        $NextID = $MIBInst;
        last;
        }
    if ($NextID eq "")
        {
        $Next = "11";   # SNMP_TYPE_endOfMibView
        }
    else
        {
        $Next = undef;
        }
    &answer_get2($GID,$NextID,$dpi_get_packet_id,$Next); # Auf ermittelte Anfrage antworten
    }

######################################################################
# ResponseSuccess(Paket-Id,GID,InstID,Wert,Typ)
# Aufbauen eines Response-Pakets als Antwort auf eine Anfrage.
# Parameter:    Paket-Id, Paket-ID der Anfrage
#       GID,      Group-ID nach der gefragt wurde
#       InstID,   Instanz-ID nach der gefragt wurde
#       Wert,     Rueckgabewert
#       Typ,      Typ des Rueckgabewerts, vgl. RFC1592,Tab.17 aber hex
#                 falls Typ nicht uebermittelt wird, erfolgt automatische
#                 Ermittlung
######################################################################
sub ResponseSuccess
    {
    # Uebernahme der Parameter
    local($dpi_Packet_ID,$GID,$InstID,$Wert,$Typ) = @_;

    unless (defined ($Typ)) {           # Typ automatisch ermitteln
        if(!($Wert eq "ENDofMIB")) {
            $Typ = $MIB{$InstID};
        }
        else {
            $Typ = "11";
        }
    }

    # Bestimmen der Laengen der Strings (auch abhaengig vom Typ)
    $GID_Laenge = length($GID);
    $Inst_Laenge = length($InstID);
    if ($Typ eq "02")   # Falls Typ = SNMP_TYPE_OCTET_STRING
```

```
    {
    $Wert_Laenge = length($Wert);
    $Wert_PackString = "A".$Wert_Laenge;
    }
elsif ($Typ eq "03")    # Falls Typ = SNMP_TYPE_OBJECT_IDENTIFIER
    {
    $Wert_Laenge = length($Wert);
    $Wert_PackString = "A".$Wert_Laenge;
    }
elsif ($Typ eq "11")    # Falls Typ = SNMP_TYPE_endOfMibView
    {
    &print_debug (1, "EndofMib");
    $Wert_Laenge = 0;
    $Wert_PackString = "";
    }
elsif ($Typ eq "81")    # Falls Typ = SNMP_TYPE_Integer32
    {
    $Wert_Laenge = 4;
    $Wert_PackString = "N";
    }
elsif ($Typ eq "0F")    # Falls Typ = SNMP_TYPE_noSuchObject
    {
    &print_debug (1, "NoSuchObject");
    $Wert_Laenge = 0;
    $WertPackString = "";
    }
elsif ($Typ eq "10")    # Falls Typ = SNMP_TYPE_noSuchInstance
    {
    &print_debug (1, "NoSuchInstance");
    $Wert_Laenge = 0;
    $WertPackString = "";
    }
else            # Falls unbekannter Typ
    {
    &print_debug (1, "Unbekannter Typ: $Typ");
    return;
    }

# Aufbauen des Pack-Strings
$PackString = "H2H2H2nH2H2H8A".$GID_Laenge."xA".$Inst_Laenge."xH2n".$Wert_PackString;

####################################################
# Paket ohne Feld Paketlaenge zusammensetzen
$dpi_Packet = pack($PackString,
            # Head-Beginn
            "02",   # Protokol Haupt Version
            "02",   # Protokol Niedere Version
```

```
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "05",   # Typ: Response-Packet
                # Head-Ende
                "00",   # Fehlercode
                "00000000", # Fehlerindex
                $GID,   # Group-ID
                $InstID,# Instanz-ID
                $Typ,   # Typ der Variablen
                $Wert_Laenge,   # Laenge des Werts
                $Wert   # Rueckgabewert
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet);      # Laenge des Packets ermitteln
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    # Pakete verschicken
    unless (print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet) {
        die "Unexpected error in DPI communication: $! (main agent down?)\nTerminating";
    }
    &print_debug (1, "Success-Response (Typ: $Typ) verschickt");
    }

######################################################################
# SendTrap(GenTrapCode,SpecTrapCode,Enterprise,GID,InstID,Typ,Wert)
# Aufbauen eines Trap-Pakets und senden desselben.
# Parameter:    GenTrapCode,    Allgemeiner Trap Code
#       SpecTrapCode,   Spezieller Trap Code
#       Enterprise, Optionale Enterprise-ID
#       GID,        Group-ID der Trap-Variablen
#       InstID,     Instanz-ID der Trap-Variablen
#       Typ,        Typ des Rueckgabewerts, vgl. RFC1592,Tab.17 aber hex
#       Wert,       Rueckgabewert
######################################################################
sub SendTrap
    {
    # Uebernahme der Parameter
    local($GenTrapCode,$SpecTrapCode,$EnterpID,$GID,$InstID,$Typ,$Wert) = @_;

    $dpi_Packet_ID++;   # Packet-ID hochzaehlen
    # Bestimmen der Laengen der Strings (auch abhaengig vom Typ)
    $EnterpID_Laenge = length($EnterpID);
    $GID_Laenge = length($GID);
    $Inst_Laenge = length($InstID);
    if ($Typ eq "02")   # Falls Typ = SNMP_TYPE_OCTET_STRING
        {
```

```
        $Wert_Laenge = length($Wert);
        $Wert_PackString = "A".$Wert_Laenge;
        }
    elsif ($Typ eq "03")    # Falls Typ = SNMP_TYPE_OBJECT_IDENTIFIER
        {
        $Wert_Laenge = length($Wert);
        $Wert_PackString = "A".$Wert_Laenge;
        }
    elsif ($Typ eq "11")    # Falls Typ = SNMP_TYPE_endOfMibView
        {
        &print_debug (1, "EndofMib");
        $Wert_Laenge = 0;
        $Wert_PackString = "";
        }
    elsif ($Typ eq "81")    # Falls Typ = SNMP_TYPE_Integer32
        {
        $Wert_Laenge = 4;
        $Wert_PackString = "N";
        }
    else            # Falls unbekannter Typ
        {
        &print_debug (1, "Unbekannter Typ: $Typ");
        return;
        }
    if ($EnterpID)
        {
        # Aufbauen des Pack-Strings mit EnterpriseID
        $PackString = "H2H2H2nH2NNA".$EnterpID_Laenge."xA".$GID_Laenge."xA".$Inst_Laenge."xH2n".$Wert_PackString;
        }
    else
        {
        # Aufbauen des Pack-Strings ohne EnterpriseID
        $PackString = "H2H2H2nH2NnA1A".$GID_Laenge."xA".$Inst_Laenge."xH2n".$Wert_PackString;
        $EnterpID = "\0";
        }
    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack($PackString,
            # Head-Beginn
            "02",   # Protokol Haupt Version
            "02",   # Protokol Niedere Version
            "00",   # Protokol Release
            $dpi_Packet_ID,
            "04",   # Typ: Trap-Packet
            # Head-Ende
            $GenTrapCode,   # Allgemeiner Trap-Code
            $SpecTrapCode,  # Spezieller Trap-Code
```

```
            $EnterpID,  # optional Enterprise ID
            $GID,   # Group-ID
            $InstID,# Instanz-ID
            $Typ,   # Typ der Variablen
            $Wert_Laenge,   # Laenge des Werts
            $Wert   # Rueckgabewert
          );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet);      # Laenge des Packets ermitteln
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    # Pakete verschicken
    unless (print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet) {
        die "Unexpected error in DPI Communication: $! (main agent down?)\nTerminating.";
    }

    &print_debug (1, "Trap successfully sent!");
    }

#############################################################
# ResponseError(Paket-Id,Fehlercode in hex)
# Aufbauen eines Response wegen Fehler und verschicken dieses.
# Parameter:    Paket-Id, Paket-id der Anfrage, die fehlerhaft
#             war
#       Fehlercode in hex, vgl. rfc1592 Tab. 18
#############################################################
sub ResponseError
    {
    # Uebernahme der Parameter
    local($dpi_Packet_ID,$Error) = @_;

    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack("H2H2H2nH2H2n",
            # Head-Beginn
            "02",   # Protokol Haupt Version
            "02",   # Protokol Niedere Version
            "00",   # Protokol Release
            $dpi_Packet_ID,
            "05",   # Typ: Response-Packet
            # Head-Ende
            $Error, # Fehlercode
            1,  # Fehler bei VarBind 1
          );
    ####################################################
    # Paketlaenge ermitteln
```

```
    $Laenge = length($dpi_Packet);      # Laenge des Packets ermitteln
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    # Pakete verschicken
    unless (print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet) {
        die "Unexpected error in DPI Communication: $! (main agent down?)\nTerminating";
    }
    &print_debug (1, "Error-Response $Error verschickt");
    }

############################################################
# DPI_UnRegister
# Aufbauen eines SNMP_DPI_UnRegister Request Packets und
# verschicken desselben.
############################################################
sub DPI_UnRegister
    {
    $dpi_Packet_ID++;   # Packet-Identifier hochzaehlen
    $GID_Laenge = length($GROUPID);
    $PackString = "H2H2H2nH2H2A".$GID_Laenge."x";

    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack($PackString,
                # Head-Beginn
                "02",   # Protokol Haupt Version
                "02",   # Protokol Niedere Version
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "07",   # Typ: Unregister-Packet
                # Head-Ende
                "02",   # Grund: Programmende
                $GROUPID    # GID ab der der Subagent zustaendig
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet);  # Laenge des Packets ermitteln (+1 wegen des zusaetzl.
                        # Feldes Paketlaenge)
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    if($child = fork)
        {
        # Pakete verschicken
        print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet;
        &print_debug (0, "Unregister-Request: ");
        }
    else
```

```
        {
        # SNMP-DPI Response entgegennehmen und auswerten
        &DPI_OpenResponse;
        }
    waitpid($child,0);
    $Return = $?>>8 ;
    if (!($Return == 0))    # Falls die Unregister-Request nicht angenommen
        {
        &print_debug (0, "nicht angenommen!");
        &print_debug (0, "Fehlercode: $Return");
        &DPI_Close; # Schliessen der DPI-Verbindung
        close(DPI_Anschluss); # Schliessen des Handles fuer die TCP-Verbindung zum Agenten
        die "\n";
        }
    else
        {
        &print_debug (0, "angenommen");
        }
    }

############################################################
# DPI_Register
# Aufbauen eines SNMP_DPI_Register Request Packets und
# verschicken desselben.
############################################################
sub DPI_Register
    {
    $dpi_Packet_ID++;   # Packet-Identifier hochzaehlen
    $GID_Laenge = length($GROUPID);
    $PackString = "H2H2H2nH2nnH2H2H2H2A".$GID_Laenge."x";

    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack($PackString,
                # Head-Beginn
                "02",   # Protokol Haupt Version
                "02",   # Protokol Niedere Version
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "06",   # Typ: Register-Packet
                # Head-Ende
                0,  # Prioritaet
                5,  # Time-Out in sek.
                "00",   # Authentifikation von Anfragen:Agent
                "0a",
                "00",   # GetBulk wird zu GetNext
                "00",
```

```
                $GROUPID    # GID ab der der Subagent zustaendig,
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet);  # Laenge des Packets ermitteln (+1 wegen des zusaetzl.
                        # Feldes Paketlaenge)
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    if($child = fork)
        {
        # Pakete verschicken
        print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet;
        &print_debug (0, "Register-Request: ");
        }
    else
        {
        # SNMP-DPI Response entgegennehmen und auswerten
        &DPI_OpenResponse;
        }
    waitpid($child,0);
    $Return = $?>>8 ;
    if (!($Return == 0))    # Falls die Unregister-Request nicht angenommen
        {
        &print_debug (0, "nicht angenommen!");
        &print_debug (0, "Fehlercode: $Return");
        &DPI_Close; # Schliessen der DPI-Verbindung
        close(DPI_Anschluss); # Schliessen des Handles fuer die TCP-Verbindung zum Agenten
        die "\n";
        }
    else
        {
        &print_debug (0, "angenommen");
        }
    }

############################################################
# DPI_AreYouThere
# Aufbauen eines SNMP_DPI_Are_You_There Request Packets und
# verschicken desselben.
############################################################
sub DPI_AreYouThere
    {
    $dpi_Packet_ID++;   # Packet-Identifier hochzaehlen

    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack("H2H2H2nH2",
```

```
                # Head-Beginn
                "02",   # Protokol Haupt Version
                "02",   # Protokol Niedere Version
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "15",   # Typ: AreYouThere-Packet
                # Head-Ende
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet);  # Laenge des Packets ermitteln (+1 wegen des zusaetzl.
                        # Feldes Paketlaenge)
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    if($child = fork)
        {
        # Pakete verschicken
        print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet;
        &print_debug (1, "Are-You_There_Request verschickt");
        }
    else
        {
        # SNMP-DPI Response entgegennehmen und auswerten
        &DPI_OpenResponse;
        }
    waitpid($child,0);
    $Return = $?>>8 ;
    if (!($Return == 0))    # Falls die Open-Request nicht angenommen
        {
        &print_debug (0, "Verbindung zum Agenten nicht in Ordnung!");
        &print_debug (0, "Fehlercode: $Return");
        die "\n";
        }
    else
        {
        &print_debug (0, "Verbindung zum Agenten in Ordnung!");
        }
    }

############################################################
# DPI_Close
# Aufbauen eines SNMP_DPI_Close Request Packets und
# verschicken desselben.
############################################################
sub DPI_Close
    {
    $dpi_Packet_ID++;   # Packet-Identifier hochzaehlen
```

```
    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack("H2H2H2nH2H2",
                # Head-Beginn
                "02",   # Protokol Haupt Version
                "02",   # Protokol Niedere Version
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "09",   # Typ: Close-Packet
                # Head-Ende
                "02"    # Grund: Programmende
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet)+1;    # Laenge des Packets ermitteln (+1 wegen des zusaetzl.
                        # Feldes Paketlaenge)
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen
    ####################################################
    # Pakete verschicken
    print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet;
    &print_debug (0, "Close-Request verschickt");
    }

############################################################
# DPI_Open
# Aufbauen eines SNMP_DPI_Open Request Packets und
# verschicken desselben.
############################################################
sub DPI_Open
    {
    $dpi_Packet_ID++;   # Packet-Identifier hochzaehlen
    $OID_Laenge = length($OID);
    $DESCRIPTION_Laenge = length($DESCRIPTION);
    $PackString = "H2H2H2nH2nnH2A".$OID_Laenge."xA".$DESCRIPTION_Laenge."xn";

    ####################################################
    # Paket ohne Feld Paketlaenge zusammensetzen
    $dpi_Packet = pack($PackString,
                # Head-Beginn
                "02",   # Protokol Haupt Version
                "02",   # Protokol Niedere Version
                "00",   # Protokol Release
                $dpi_Packet_ID,
                "08",   # Typ: Open-Packet
                # Head-Ende
                5,  # Time-Out in Sekunden
```

```
                1,  # Maximum VarBinds pro Packet
                "01",   # Schriftsatz: eigener
                $OID,   # OID ab der der Subagent zustaendig,
                $DESCRIPTION, # Erklaerung zum Subagenten
                0   # Passwort-Laenge ist Null
              );
    ####################################################
    # Paketlaenge ermitteln
    $Laenge = length($dpi_Packet)+1;    # Laenge des Packets ermitteln (+1 wegen des zusaetzl.
                        # Feldes Paketlaenge)
    $dpi_Packet_Length = pack("n",$Laenge); # Laenge des Packets in eigenes Paket legen

    if($child = fork)
        {
        # Pakete verschicken
        print DPI_Anschluss $dpi_Packet_Length.$dpi_Packet;
        &print_debug (0, "Open-Request: ");
        }
    else
        {
        # SNMP-DPI Response entgegennehmen und auswerten
        &DPI_OpenResponse;
        }
    waitpid($child,0);
    $Return = $?>>8 ;
    if (!($Return == 0))    # Falls die Open-Request nicht angenommen
        {
        &print_debug (0, "nicht angenommen!");
        &print_debug (0,  "Fehlercode: $Return");
        die "\n";
        }
    else
        {
        &print_debug (0, "angenommen");
        }
    }

############################################################
# DPI_OpenRespone
# Auf Response-Paket auf Open- bzw. AreYouThere-Request warten
# und auf Erfolg des Requests ueberpruefen.
############################################################
sub DPI_OpenResponse
    {
    # Lokale Variablen
    local($Reponse,@Response);
```

```
    recv(DPI_Anschluss,$Response,15,0);

    @Response = unpack("nH2H2H2nH2H2H2H2H2H2",$Response);

    if($Response[5] eq "05" && $Response[6] eq "00" )
        {
        exit;
        }
    else
        {
        exit(hex($Response[6]));
        }
    }


############################################################
# TCPConnect($Port)
# Aufbauen einer TCP-Verbindung zum, durch die globale Var.
# $AGENT_HOST gegebenen Rechner ueber Port $Port.
# Bei Erfolg steht DPI_Anschluss als Handle zur Verfuegung.
############################################################
sub TCPConnect
    {
    # Uebernahme der Parameter
    local($Port) = @_ ;

    &print_debug (0, "DPIConnect():");
    # Name des Rechners, von dem aus die Verbindung
    # aufgebaut wird
    $hostname   = `hostname`;

    # Service fuer die Anbindung aufbereiten
    ($name,$aliases,$proto) = getprotobyname('tcp');
    ($name,$aliases,$Port) = getservbyname($Port,'tcp')
    unless $Port =~ /^\d+$/;

    ($name, $aliases, $type, $len, $thisaddr) = gethostbyname($hostname);
    $this = pack($SOCKADDR, $AF_INET, 0, $thisaddr);

    # Gegenadresse auf gewuenschte Domaine setzen
    ($name, $aliases, $type, $len, $thataddr) = gethostbyname($AGENT_HOST);
    $that = pack($SOCKADDR, $AF_INET, $Port, $thataddr);

        # Create a handle to the socket
    socket(DPI_Anschluss, $AF_INET, $SOCK_STREAM, $proto) || die "Socket fehlgeschlagen\n";

    # Assign the socket an address
```

```
    bind(DPI_Anschluss, $this) || die "Bind fehlgeschlagen\n";

    # Connect to the server
    connect(DPI_Anschluss,$that) || die "Connect fehlgeschlagen\n";

    select(DPI_Anschluss);
    $| = 1;     # Auf nicht puffern stellen
    select(STDOUT);
    &print_debug (0, " ok");
    }

#######################################################
# qDPIport
# Unterprogramm zum Ermitteln des Port's, ueber den der
# Subagent mit dem Agenten kommunizieren soll.
# Benoetigt den Hostnamen des Agenten in der globalen
# Variablen $AGENT_HOST
# Rueckgabe: TCP-Port der fuer die weitere DPI-Verbindung
#       vorgeschlagen wurde.
#######################################################
sub qDPIport
    {
    # Definition von lokalen Variablen
    local($snmpPacket,@snmpPacket);

    &print_debug (0, "qDPIport: ");
    # Paket zusammensetzen
    $snmpPacket = pack( "H2H2H6H2H2A6H4H6H6H6H4H4H4H22H4",
                "30",    # ASN.1 Header in Hex
                "29",    # PDU-length
                "020100",# snmp Version
                "04",    # Feld:Community-Name
                "06",    # Laenge Community-Name
                "public",# Community-Name
                "a01c",  # snmp-Get-Request
                "020101",# snmp-Request-Id
                "020100",# snmp-error-Status
                "020100",# snmp-Index
                "3011",  # varBind list
                "300f",  # varBind
                "060b",  # Feld:Object-Id
                "2b060104010202010100", # Object-Id
                "0500"   # null-Value
                  );


    #####################################
```

```
    # UDP-Verbindung zu Agenten aufbauen

    # Name des Rechners, auf dem der Subagent gestartet wird
    $hostname   = `hostname`;

    ($name, $aliases, $type, $len, $addr) = gethostbyname($hostname);
    $this = pack($SOCKADDR, $AF_INET, 0, $addr);
    ($name, $aliases, $type, $len, $thataddr) = gethostbyname($AGENT_HOST);
    $that = pack($SOCKADDR, $AF_INET, $SNMP_PORT, $thataddr);

        # Create a handle to the socket
    socket(SNMP_Anschluss, $AF_INET, $SOCK_DGRAM, 17) || die "Socket fehlgeschlagen\n";

    # Assign the socket an address
    bind(SNMP_Anschluss, $this) || die "Bind fehlgeschlagen\n";

    ##########################################
    # SNMP-Get-Request-Paket schicken
    send(SNMP_Anschluss,$snmpPacket,0,$that);

    ##############################################################
    # SNMP-Response-Paket empfangen die noetigen Informationen
    # ermitteln und zurueckgeben.
    recv(SNMP_Anschluss,$snmp_Packet,54,0);
    close SNMP_Anschluss;
    @snmp_Packet = unpack("H2H6H6H2H2A6H8H6H6H6H8H8H4H22H4n",$snmp_Packet);

    &print_debug (0, "ok");
    return($snmp_Packet[15]);
    }

################################################
# Splittet eine InstID bei Tabellen in Zeilen und
# Spaltennummer und BasisOID auf.
################################################
sub split_InstID
{
    local($InstID)=@_;
    $ZeilenNr = substr($InstID,rindex($InstID,'.')+1);
    $InstID = substr($InstID,0,rindex($InstID,'.'));
    $SpaltenNr = substr($InstID,rindex($InstID,'.')+1);
    $InstID = substr($InstID,0,rindex($InstID,'.'));
    return ($InstID, $ZeilenNr, $SpaltenNr);
}

################################################
# Vergleichsfunktion zum Sortieren
```

```
##################################################
sub by_hierarchy
    {
    local($i,@a,@b);

    @a = split(/\./,$a);
    @b = split(/\./,$b);

    if (@a < @b)
        {
        for ($i=0;$i<@a;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
                {
                return($a[$i]-$b[$i]);
                }
            }
        return(-1);
        }
    elsif (@a > @b)
        {
        for ($i=0;$i<@b;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
                {
                return($a[$i]-$b[$i]);
                }
            }
        return(1);
        }
    else
        {
        for ($i=0;$i<@b;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
```

```
                {
                return($a[$i]-$b[$i]);
                }
            }
        return(0);
        }
    }

##################################################
# Vergleichsfunktion
##################################################
sub less_by_hierarchy
    {
    # Uebernahme der Parameter
    local($a,$b)=@_;
    local($i,@a,@b);

    @a = split(/\./,$a);
    @b = split(/\./,$b);

    if (@a < @b)
        {
        for ($i=0;$i<@a;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
                {
                return($a[$i]-$b[$i]);
                }
            }
        return(-1);
        }
    elsif (@a > @b)
        {
        for ($i=0;$i<@b;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
                {
                return($a[$i]-$b[$i]);
                }
```

```
            }
        return(1);
        }
    else
        {
        for ($i=0;$i<@b;$i++)
            {
            if ($a[$i] == $b[$i])
                {
                next;
                }
            else
                {
                return($a[$i]-$b[$i]);
                }
            }
        return(-1);
        }
    }


1;
```

B.3 mib_gets.pl

```
############################################################
# Hilfsfunktionen fuer answer_get2
############################################################
sub do_get
{
    local ($InstID)=@_;
    &print_debug (2, "Get auf InstanzID $InstID empfangen");
    $Wert = &get($InstID);
    if (defined ($Wert)) {
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert);
    }
    else {  # Nicht vorhanden
        if ($$snmpversion == 1) {
            # SNMP_ERROR_noSuchName
            &ResponseError($dpi_get_packet_id,"02")
        }
        else {
            # SNMP_TYPE_noSuchInstance
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert,"10");
        }
    }
```

```
}

sub do_get_inc
{
    local ($InstID)=@_;
    &print_debug (2, "GetInc auf InstanzID $InstID empfangen");
    $Wert = &get_inc($InstID);
    if (defined ($Wert)) {
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert);
    }
    else {  # Nicht vorhanden
        if ($$snmpversion == 1) {
            # SNMP_ERROR_noSuchName
            &ResponseError($dpi_get_packet_id,"02")
        }
        else {
            # SNMP_TYPE_noSuchInstance
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert,"10");
        }
    }
}



############################################################
# answer_get2
# Beantwortung eines get's oder getnext's
############################################################
sub answer_get2
    {
    # Parameter uebernehmen
    local($GID,$InstID,$dpi_get_packet_id,$Next)=@_;
    # Lokale Variablen
    local($GID_Laenge,$InstID_Laenge,@Wert);
    ############################################################
    # Gesuchten Wert ermitteln
    if (!($GID eq $GROUPID))    # Falls der Subagent fuer diese Group-ID
        {           # nicht zustaendig
        &ResponseError($dpi_get_packet_id,"05") # Allgemeiner Fehler
        }

    if ($Next)  # End of MIB bei getnext
        {
        &print_debug (1, "EndofMIB");
        $Wert = "ENDofMIB";
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert);
        $Next = undef;
```

```
    }
elsif ($InstID =~ /^1\.1\.1\.1/)    # trapSourceID
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.1\.1\.2/)    # trapSourceTableStatus
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.1\.1\.3/)    # facility
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.1\.1\.4/)    # severity
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.1\.1\.5/)    # logstring
    {
    &do_get ($InstID);
    }
elsif ($InstID eq '1.2.0')          # trapSourceTableNextInstance
    {
    &do_get_inc ($InstID);
    }
elsif ($InstID =~ /^1\.3\.1\.1/)    # trapDestIndex
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.3\.1\.2/)    # trapDestID
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.3\.1\.3/)    # trapDestVal
    {
    &do_get ($InstID);
    }
elsif ($InstID =~ /^1\.3\.1\.4/)    # trapDestTableStatus
    {
    &do_get ($InstID);
    }
elsif ($InstID eq '1.4.0')          # trapDestTableNextInstance
    {
    &do_get_inc ($InstID);
    }
elsif ($InstID eq '2.1.0')          # historyMaxAge
    {
```

```
        &do_get ($InstID);
        }
    elsif ($InstID eq '2.2.0')          # historyMaxEntries
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^2\.3\.1\.1/)    # trapNummer
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^2\.3\.1\.2/)    # trapID
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^2\.3\.1\.3/)    # trapZeit
        {
        &do_get ($InstID);
        }
    elsif ($InstID eq '3.1.0')         # uptime
        {
        &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,(time-$start_zeit)*100);
        }
    elsif ($InstID eq '3.2.0')         # steuerung
        {
        &do_get ($InstID);
        }
    elsif ($InstID eq '3.3.0')         # snmpversion
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^4\.1\.1\.1/)   # syslogZeilenNummer
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^4\.1\.1\.2/)   # syslogZeile
        {
        &do_get ($InstID);
        }
    elsif ($InstID =~ /^4\.1\.1\.3/)   # syslogConfTableStatus
        {
        &do_get ($InstID);
        }
    elsif ($InstID eq '4.2.0')         # syslogConfTableNextInstance
        {
        &do_get_inc ($InstID);
        }
    else {      # Variable nicht vorhanden
```

```
        if ($$snmpversion == 1) {
            &ResponseError($dpi_get_packet_id,"02") # noSuchName
        }
        else { # noSuchObject
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert,"0F");
        }
    }
}

1;
```

B.4 mib_sets.pl

```
############################################################
# answer_set2
# Beantwortung eines Set-, Commit-, Undo-Requests
############################################################
sub answer_set2
{
    # Uebernahme der Parameter
    local($GID,$InstID,$Art,$dpi_get_packet_id,$Set_Typ,$Set_Laenge,@Set_Wert)=@_;

    # print "Set auf $InstID (Art: $Art) empfangen.\n";
    # print "SetLaenge: $Set_Laenge\nSetWert: ";
    # for ($i=0; $i<@Set_Wert; $i++) {print $Set_Wert[$i]}; print "\n";
    if (!$GID eq $GROUPID)  # Falls der Subagent fuer diese Group-ID
        {       # nicht zustaendig
        &print_debug (2,  "Subagent fuer Set nicht verantwortlich: $GID vs. $GROUPID!");
        &ResponseError($dpi_get_packet_id,"05") # Allgemeiner Fehler
        }
    # Set-Variablen
    if ($InstID =~ /^1\.1\.1\.1/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,2,5)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
```

```
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.1\.1\.2/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,2,5)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.1\.1\.3/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,2,5)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
```

```
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.1\.1\.4/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,2,5)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.1\.1\.5/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,2,5,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,2,5)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
```

```
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.3\.1\.1/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,4,4,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,4,4)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.3\.1\.2/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,4,4,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,4,4)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
```

```
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.3\.1\.3/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,4,4,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,4,4)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^1\.3\.1\.4/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,4,4,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,4,4)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
```

```
          {
          &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
          }
      }
  elsif ($InstID eq '2.1.0')
      {
      if ($Art eq "03")            # Set - Request
          {
          $Fehler = &set($InstID,$Set_Typ,$Set_Laenge,@Set_Wert);
          }
      elsif ( $Art eq "0a")        # Commit - Request
          {
          $Fehler = &com($InstID)  # Commit-Request
          }
      elsif ( $Art eq "0b")        # Undo - Request
          {
          $Fehler = &undo($InstID);
          }
      if(!$Fehler)
          {
          &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
          }
      else
          {
          &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
          }
      }
  elsif ($InstID eq '2.2.0')
      {
      if ($Art eq "03")            # Set - Request
          {
          $Fehler = &set($InstID,$Set_Typ,$Set_Laenge,@Set_Wert);
          }
      elsif ( $Art eq "0a")        # Commit - Request
          {
          $Fehler = &com($InstID)  # Commit-Request
          }
      elsif ( $Art eq "0b")        # Undo - Request
          {
          $Fehler = &undo($InstID);
          }
      if(!$Fehler)
          {
          &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
          }
      else
          {
```

```
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID eq '3.1.0')
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set($InstID,$Set_Typ,$Set_Laenge,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID eq '3.2.0')
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set($InstID,$Set_Typ,$Set_Laenge,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
```

```
            }
        }
    elsif ($InstID eq '3.3.0')
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set($InstID,$Set_Typ,$Set_Laenge,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID)  # Commit-Request
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^4\.1\.1\.1/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,3,3,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,3,3)  # Commit-Request
            &dump_syslog_conf();
        }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
```

```
            }
        }
    elsif ($InstID =~ /^4\.1\.1\.2/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,3,3,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,3,3)  # Commit-Request
            &dump_syslog_conf();
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    elsif ($InstID =~ /^4\.1\.1\.3/)
        {
        if ($Art eq "03")            # Set - Request
            {
            $Fehler = &set_Tabelle($InstID,$Set_Typ,$Set_Laenge,3,3,@Set_Wert);
            }
        elsif ( $Art eq "0a")        # Commit - Request
            {
            $Fehler = &com($InstID,3,3)  # Commit-Request
            &dump_syslog_conf();
            }
        elsif ( $Art eq "0b")        # Undo - Request
            {
            $Fehler = &undo($InstID);
            }
        if(!$Fehler)
            {
            &ResponseSuccess($dpi_get_packet_id,$GID,$InstID,$Wert{$InstID},$Set_Typ);
            }
        else
            {
```

```
            &ResponseError($dpi_get_packet_id,$Fehler) # nicht Schreibbar
            }
        }
    else {      # Keine Set-Variable vorhanden
        if ($$snmpversion == 1) {
            &ResponseError($dpi_get_packet_id,"03") # badValue
        }
        else {
        &ResponseError($dpi_get_packet_id,"11") # nicht Schreibbar
        }
    }
}
1;
```

B.5 sub_gets.pl

```
#################################################################
# get
# Generische GET Funktion f
# Skalarwerte und Tabelleneintraege
# Falls Manager nicht existierende Instanz anfordert, wird hier
# automatisch 'undef' zurueckgeliefert, die Auswertung diesbez.
# findet erst in mib_gets.pl statt.
# Parameter: InstID, d.h. Nummer der Variablen
# Rueckgabe: Wert der Variablen
#################################################################
sub get
{
    # Uebernahme der Parameter
    local($InstID)=@_;
    return($Wert{$InstID});
}

#################################################################
# get_inc
# Wie oben, aber Wert der Variablen wird nach Get um 1 erhoeht.
# Dient zum automatischen Erhoehen von NextInstance oder durch-
# laufenden Indexvariablen, die der Manager anfordert.
# Parameter: InstID, d.h. Nummer der Variablen
# Rueckgabe: Wert der Skalarvariablen
#################################################################
sub get_inc
{
    # Uebernahme der Parameter
    local($InstID)=@_;
    if (defined($Wert{$InstID})) {
```

```
        return ($Wert{$InstID}++);
}
    else {
        return (undef);
    }
}

1;
```

B.6 sub_sets.pl

```
#################################################################
# set(InstID,Typ,Laenge,Wert)
# Generische Set-Funktion fuer Skalarvariablen
# Parameter : Typ,  gibt den Typ des zu setzenden Werts an
#         Laenge,   gibt die Laenge des Werts an
#         Wert, der Wert selbst, als hex-Array
#         InstID    welches Set?
# Rueckgabe : undef bei Erfolg, sonst Fehlercode
#################################################################
sub set
{
    # Uebernahme der Parameter
    local($InstID,$SetTyp,$SetLaenge,@SetWert)=@_;
    # Lokale Variablen
    local($Temp);
    &print_debug (2, "SkalarSet auf $InstID empfangen\n");

    if (!($SetTyp eq $MIB{$InstID})) {  # Typ ueberpruefen
        if ($$snmpversion == 1) {
            return ("03");              # SNMP_ERROR_badValue
        }
        else {
            return ("07");              # SNMP_ERROR_wrongType
        }
    }
    elsif ($SetTyp eq "02") {
        foreach (@SetWert)
            {
            $NewWert = unpack("A",pack("H2",$_));
            $Temp = $Temp.$NewWert;
            }
        $WertTemp{$InstID} = $Temp;
    $WertDef{$InstID} = 1;
    }
    elsif ($SetTyp eq "81") {
```

```
        $WertTemp{$InstID} = unpack("N",pack("H2H2H2H2",$SetWert[0],$SetWert[1],$SetWert[2],$SetWert[3]));
    $WertDef{$InstID} = 1;
    }
    &print_debug (3, "Wert vorlaeufig: $WertTemp{$InstID}\n");
    return(undef);
}



################################################################
# set_Tabelle (InstID,Typ,Laenge,StatusColumn,ColumnCount,Wert)
# Wie oben, aber fuer Tabellen (Handling der Status Column)
# Parameter : Typ,  gibt den Typ des zu setzenden Werts an
#         Laenge,   gibt die Laenge des Werts an
#         Wert, der Wert selbst, als hex-Array
#         InstID    welches Set?
#         StatusColumn Integer, der die Nummer der Statusspalte angibt
#                      (beginnt bei 1)
#         ColumnCount  Integer, gibt Anzahl der Tabellenspalten an
# Rueckgabe : Fehler: undef bei Erfolg, sonst Fehlercode
################################################################
sub set_Tabelle
{
    # Uebernahme der Parameter
    local($InstID,$SetTyp,$SetLaenge,$StatusColumn,$ColumnCount,@SetWert)=@_;
    # Lokale Variablen
    local($Temp,$ZeilenNummer,$SpaltenNummer,$Status,$i,$BasisOID);

    # Wenn Debug = 1 werden detailliertere Statsumeldungen ausgegeben
    $Debug = 1;

    # BasisOID/ZeilenNummer/SpaltenNummer des Eintrags holen
    ($BasisOID, $ZeilenNummer, $SpaltenNummer) = &split_InstID ($InstID);

    &print_debug (2, "TabellenSet auf $InstID empfangen.");
    &print_debug (3, "Zeile: $ZeilenNummer, Spalte: $SpaltenNummer, $ColumnCount Spalten, Statusspalte: $StatusColumn\n");

    if ($SpaltenNummer == $StatusColumn) {  # Status Column Set Request
        &print_debug (3, "Status Column Set Request");
    # Typ pruefen
        unless ($SetTyp eq "81") {          # Integer
            &print_debug (3, "Status Column Set falscher Typ");
            if ($$snmpversion == 1) {
                return ("03");              # SNMP_ERROR_badValue
            }
            else {
                return ("07");              # SNMP_ERROR_wrongType
```

```
        }
    }
    # Wertebereich pruefen
    $Temp = unpack("N",pack("H2H2H2H2",$SetWert[0],$SetWert[1],$SetWert[2],$SetWert[3]));
    &print_debug (2, "Wert: $Temp");
unless (($Temp > 0 && $Temp < 7) && $Temp != $RS_NOTREADY) {
        &print_debug (3, "Status Column Set falscher Wert");
        if ($$snmpversion == 1) {
            return ("03");              # SNMP_ERROR_badValue
        }
        else {
       return ("0A");               # SNMP_ERROR_wrongValue
        }
    }
    # Auf Statusaenderung reagieren
    $Status = $Wert{$InstID};
    if ($Temp == $RS_CREATEANDGO) {     # createAndGo
        &print_debug (3, "createAndGo nicht moeglich");
        return ("0C");                  # inconsistentValue
    }
    elsif ($Temp == $RS_CREATEANDWAIT) { # createAndWait
        if (defined ($Status)) {        # Zeile gibt es schon
            &print_debug (3, "Debug: createAndWait auf bestehende Zeile versucht.");
            return ("0C");              # inconsistentValue
        }
        else {                          # Zeile erzeugen
    for ($i = 1; $i <= $ColumnCount; $i++) {
            #    print $BasisOID.'.'.$i.'.'.$ZeilenNummer.' wird '.$MIB{$BasisOID.'.'.$i.'.'."0"}."\n";
        $MIB{$BasisOID.'.'.$i.'.'.$ZeilenNummer} = $MIB{$BasisOID.'.'.$i.'.'."0"};
            }
            #@MIB = sort by_hierarchy keys(%MIB);
            $WertTemp{$InstID} = $RS_NOTREADY;
    $WertDef{$InstID} = 1;
        }

    }
    elsif ($Temp == $RS_ACTIVE) {       # active
        if ($Status == $RS_NOTINSERVICE || $Status == $RS_ACTIVE) {
            $WertTemp{$InstID} = $RS_ACTIVE;
        }
        else {
            print_debug (3,  "active auf nicht fertige Zeile (Status: $Status) versucht.");
            return ("0C");              # inconsistentValue
        }
    }
    elsif ($Temp == $RS_NOTINSERVICE) { # notInService
        if ($Status == $RS_NOTINSERVICE || $Status == $RS_ACTIVE) {
```

```
            $WertTemp{$InstID} = $RS_NOTINSERVICE;
        }
        else {
            &print_debug (3, "notInService auf nicht bestehende Zeile (Status: $Status) versucht.");
            return ("OC");              # inconsistentValue
        }
    }
    else {                              # destroy
        # Ein Destroy zerstoert eine Zeile unwiederbringlich,
        # d.h. es kann kein UNDO mehr ausgefuehrt werden. Da bei
        # einem Destroy auch kein Fehler auftreten kann, wird die
        # Ausfuehrung in die COMMIT Routine verlagert.
        &print_debug (3, "Destroy Request auf $InstID.");
        $WertTemp{$InstID} = $RS_DESTROY;
    }
}
else {                              # "Normale" Column Set Request
    if (!defined ($Wert{$BasisOID.'.'.$StatusColumn.'.'.$ZeilenNummer})) {
        # Error - Agent muss erst Status Column erzeugen
        &print_debug (3, "Fehler - Zeile $ZeilenNummer noch nicht erzeugt.");
    return ("OC");              # inconsistentValue
    }
    else {                          # Zeile bereits erzeugt
        $Temp = set ($InstID,$SetTyp,$SetLaenge,@SetWert);
        if (defined ($Temp)) {      # Fehler bei SET
           return ($Temp);          # Fehlercode zurueck
        }
        else {
            # Jetzt pruefen, ob alle Spalten mit Werten besetzt
            if ($Wert{$BasisOID.'.'.$StatusColumn.'.'.$ZeilenNummer} == $RS_NOTREADY) {
                &print_debug (3, "Entering Row Consistancy Check");
        $Temp = 1;
                for ($i = 1; $i <= $ColumnCount; $i++) {
                    #print $WertDef{$BasisOID.'.'.$i.'.'.$ZeilenNummer}."\n";
        unless (defined ($WertDef{$BasisOID.'.'.$i.'.'.$ZeilenNummer})) {
                        $Temp = 0;
                    }
                }
                if ($Temp) { # alles ok!
                    &print_debug (3, "Status von Zeile $ZeilenNummer von NOT_READY auf NOT_INSERVICE gaendert.");
        $TempStatusColumn = $RS_NOTINSERVICE;
                }
            }
        }
    }
}
return (undef);
```

```
}

1;
```

B.7 sub_coms.pl

```
################################################################
# com fuehrt das COMMIT, also das eigentliche Setzen der
# Variablen aus. Dazu wird WertTemp endgueltig nach Wert kopiert.
#
# Tabellenzeilen werden besonders behandelt:
# - handelt es sich um ein COMMIT eines Destroy Requests auf eine
#   Status Column, so wird die gesamte Tabellenzeile zerstoert.
# - handelt es sich um ein COMMIT auf einen 'normalen' Tabellen-
#   eintrag, der die Tabellenzeile komplettiert, so wird der
#   Statuswert der Zeile von notReady auf notInService geandert.
#   Vgl. dazu die Variable $TempStatusColumn in set_Tabelle in
#   sub_sets.pl
#
# Parameter : InstID        - Object Identifier
#             StatusColumn  - Nummer der Status Column, bei
#                             Commit in Tabellenzeile, undef sonst
#             ColumnCount   - Anzahl der Tabellenspalten, bei
#                             Tabellenzugriff, undef sonst
#
# Rueckgabe : undef bei Erfolg, sonst Fehlercode
################################################################
sub com
{
    local ($InstID,$StatusColumn,$ColumnCount) = @_;
    local ($BasisOID,$ZeilenNummer,$SpaltenNummer,$i);
    if (defined ($StatusColumn)) { # Ja, Zugriff auf Tabelle
    ($BasisOID, $ZeilenNummer, $SpaltenNummer) = &split_InstID ($InstID);
        &print_debug (3, "TableCommit $InstID auf Zeile $ZeilenNummer, Spalte $SpaltenNummer empfangen");
        if ($SpaltenNummer == $StatusColumn && $WertTemp{$InstID} == $RS_DESTROY) {
            # COMMIT auf destroy Request - Zeile loeschen
            for ($i = 1; $i <= $ColumnCount; $i++) {
                delete $Wert{$BasisOID.'.'.$i.'.'.$ZeilenNummer};
                delete $WertTemp{$BasisOID.'.'.$i.'.'.$ZeilenNummer};
                delete $WertDef{$BasisOID.'.'.$i.'.'.$ZeilenNummer};
                if ($ZeilenNummer != 0) {
            # Nur zusaetzlich angelegte Zeilen aus %MIB
            # loeschen, Zeile 0 muss wegen korrekter
            # Typpruefung erhalten bleiben
            delete $MIB{$BasisOID.'.'.$i.'.'.$ZeilenNummer};
        }
```

```
            }
            @MIB = sort by_hierarchy keys(%MIB);
        }
    elsif ($SpaltenNummer == $StatusColumn && $WertTemp{$InstID} == $RS_ACTIVE) {
        # ACTIVE auf Zeile, jetzt sichtbar machen
            $Wert{$InstID} = $WertTemp{$InstID};
        @MIB = sort by_hierarchy keys(%MIB);
        }
        else {
            # Erstmal Wert uebernehmen...
        $Wert{$InstID} = $WertTemp{$InstID};
        # ... und pruefen, ob COMMIT den Status der ganzen Zeile von
            # notReady auf notInService aendert
            if (defined ($TempStatusColumn)) {  # Ja!
                $Wert{$BasisOID.'.'.$StatusColumn.'.'.$ZeilenNummer} = $TempStatusColumn;
                $WertTemp{$BasisOID.'.'.$StatusColumn.'.'.$ZeilenNummer} = $TempStatusColumn;
                $TempStatusColumn = undef;
            }
        }
    }
    else {  # Ganz normales Commit
        &print_debug (3, "Skalar Commit auf $InstID empfangen");
    $Wert{$InstID} = $WertTemp{$InstID};
    }
    return(undef);
}

1;
```

B.8 sub_undos.pl

```
#################################################################
# undo Funktion zum Abbrechen einer Variablen-Aenderung
# Parameter : Instance-ID zum Identifizieren der Variablen
# Rueckgabe : undef bei Erfolg, sonst Fehlercode
#################################################################
sub undo
    {
    # Uebernahme der Parameter
    local($InstID)=@_;
    &print_debug (1, "Undo auf $InstID empfangen");
    $WertTemp{$InstID} = $Wert{$InstID};
    return(undef);
    }

1;
```

B.9 handle_syslog.pl

```
#
# This module provides the actual functions that
# read events from the /dev/log pipe or socket
# and check if a trap needs to be triggered.
#

use Socket;

################################################
# Check whether /dev/log is pipe or socket and
# open it accordingly
################################################
sub open_logger
{
    # Check if named pipe is used (HP-UX) and open it
    if (-p $SYSLOGPIPESOCK) {
        &print_debug (2, "Opening $SYSLOGPIPESOCK as named pipe");
        open (SL_HANDLE, $SYSLOGPIPESOCK)
            or die "open: $!";
    }
    # If no pipe, check if socket already installed and abort
    elsif (-S $SYSLOGPIPESOCK) {
        die "$SYSLOGPIPESOCK already installed by another process!";
    }
    # If /dev/log exists but is neither socket nor pipe
    # something weird is going on
    elsif (-e $SYSLOGPIPESOCK) {
        die "$SYSLOGPIPESOCK is neither socket nor named pipe!";
    }
    # Install my own socket to read from
    else {
        socket (SL_HANDLE, PF_UNIX, SOCK_STREAM, 0)     or die "socket: $!";
        unlink ($SYSLOGPIPESOCK);
        bind (SL_HANDLE, sockaddr_un ($SYSLOGPIPESOCK)) or die "bind; $!";
        listen (SL_HANDLE, 5);
    }
}


################################################
# If there is a syslog message, read it, decode
# it and trigger a trap (if configured)
################################################
sub read_logger {
    my ($facility, $severity, $act_string);
```

```
    my ($source_id, @dest_ids, $i, $trap_source_inst);
    if (-p $SYSLOGPIPESOCK) {
        $buf = <SL_HANDLE>;
        if (!$buf) {
            &print_debug (3, "Buf ist leer!");
            return ();
        }
    }
    else {
        accept (Client, SL_HANDLE);
        $buf = <Client>;
        close Client;
        if (defined ($SYSLOGSOCKNEW)) {
                # Pass msg along to real syslogd
            socket (SN_HANDLE, PF_UNIX, SOCK_STREAM, 0) or die "socket: $!";
                connect (SN_HANDLE, sockaddr_un ($SYSLOGSOCKNEW))
                        or die "connect: $!";
            print SN_HANDLE $buf;
            close SN_HANDLE;
            }
    }

    ($facility, $severity, $act_string) = decode_string ($buf);
    &print_debug (3, "Syslog string decoded to [$facility][$severity][$act_string]");
    unless ($facility) {
        return ();
    }

    ($source_id, $trap_source_inst) = scan_source_table ($facility, $severity, $act_string);
    if (defined ($source_id)) {
    @dest_ids = scan_dest_table ($source_id);
        for ($i = 0; $i < @dest_ids; $i++) {
            # Trigger the trap!
            &SendTrap (6, 0, "1.3.6.1.3.100.8.1", '1.3.6.1.3.100.8.1.', "1.5.0", "02", "[$source_id] $act_string");

            # Insert into history!
            &insert_into_history ($source_id);
        &print_debug (1, "Triggering trap destination $dest_ids[$i]\n");
        }
    }
}


################################################
# Insert a triggered trap into the
# history table
################################################
```

```
sub insert_into_history {
    my $trapid = @_;

    $Wert{'2.3.1.1.'.$anz_traps_triggered} = $anz_traps_triggered;
    $Wert{'2.3.1.2.'.$anz_traps_triggered} = $trapid;
    $Wert{'2.3.1.3.'.$anz_traps_triggered} = localtime(time);
    # $MIB{'2.3.1.1.'.$anz_traps_triggered} = $MIB{'2.3.1.1.0'};
    # $MIB{'2.3.1.2.'.$anz_traps_triggered} = $MIB{'2.3.1.2.0'};
    # $MIB{'2.3.1.3.'.$anz_traps_triggered} = $MIB{'2.3.1.3.0'};
    $WertDef{'2.3.1.1.'.$anz_traps_triggered} = 1;
    $WertDef{'2.3.1.1.'.$anz_traps_triggered} = 1;
    $WertDef{'2.3.1.1.'.$anz_traps_triggered} = 1;

    $anz_traps_triggered += 1;
}


################################################
# Scan the trapSourceTable for a match
# and return trapSourceID or undef if no match
################################################
sub scan_source_table {
    my ($facility, $severity, $logstring) = @_;
    my ($i, $trapid);

    # $Wert{'1.2.0'} entspricht trapSourceTableNextInstance
    for ($i = 0; $i < $Wert{'1.2.0'}; $i++) {
        if ($facility =~ /$Wert{'1.1.1.3.'.$i}/ and
            $severity =~ /$Wert{'1.1.1.4.'.$i}/ and
            $logstring =~ /$Wert{'1.1.1.5.'.$i}/ and
            $Wert{'1.1.1.2.'.$i} == $RS_ACTIVE) {

            # return trapSourceID
            return ($Wert{'1.1.1.1.'.$i}, '1.1.1.1.'.$i);
        }
    }

    return undef;
}

################################################
# Scan the trapDestTable for matching trap
# destinations and return a list of line numbers
# or an empty array if none found
################################################
sub scan_dest_table {
    my ($trapid) = @_;
```

```
    my ($i, $j) = (0, 0);
    my (@result);

    # $Wert{'1.4.0'} entspricht trapDestTableNextInstance
    for ($i = 0; $i < $Wert{'1.4.0'}; $i++) {
        if ($trapid == $Wert{'1.3.1.2.'.$i} and
            $Wert{'1.3.1.4.'.$i} == $RS_ACTIVE) {
        $result[$j++] = $i;
        }
    }

    return (@result);
}

################################################
# Decode the string read from socket/pipe
# and return facility/severity/logstring
# as an array
################################################
sub decode_string {
    my ($instring) = @_;
    my ($facility, $severity);
    my ($before, $pri, $remainder);

    # See syslog.h for following arrays
    my @severitynames = ("emerg", "alert", "crit", "err", "warning",
                         "notice", "info", "debug");
    my @facilitynames = ("kern", "user", "mail", "daemon", "auth",
                         "syslog", "lpr", "news", "uucp", "cron",
                         "authpriv");

    ($before, $pri, $remainder) = split /<|>/, $instring, 3;
    unless (defined $remainder) {
        # Instring is broken (no fac/sev)
        &print_debug (3, "Broken Syslog String (binar. trash?");
        return ();
    }

    $facility = ($pri & 0x3f8) >> 3;    # See syslog.h
    $severity = $pri & 0x07;            # See syslog.h

    # Remove trailing trash (HP-UX Pipe)
    ($remainder) = split /[^\- :\w\[\]\/\*]/, $remainder, 2;
    return ($facilitynames[$facility], $severitynames[$severity], $remainder);
}

1;
```

B.10 support.pl

```
# Dieses Modul enthaelt Unterstuetzungsfunktionen
#
# - Debug Routine
# - Socket Dispatcher
# - Signal Handler/Config Dump & Restore
# - /syslog.conf rausschreiben

##################################################
# Debug
##################################################
sub print_debug
{
    my ($severity,@logstring) = @_;

    # Check if running in background.
    # If so, don't output anything
    if ($dbforeground) {
        if ($severity <= $DEBUGLEVEL) {
            foreach $string (@logstring) {
                print $string;
            }
            print "\n";
        }
    }
}


#################################################
# Nach Aenderung syslog.conf rausschreiben
#################################################
sub dump_syslog_conf {
    unless (defined $SyslogConfPfadWrite) {
        return ();
    }

    unless (open (SYSLOGDUMP, ">".$SyslogConfPfadWrite)) {
        &print_debug (1, "Could NOT write $SyslogConfPfadWrite\n");
    } else {
        foreach $key (sort by_hierarchy keys(%Wert)) {
            if ($key =~ /^4\.1\.1\.2\./) {
                print SYSLOGDUMP "$Wert{$key}\n";
            }
        }
        close (SYSLOGDUMP);
    }
```

```
}

################################################
# Dump Config Hashes to file
################################################
sub dump_config_hashes
{
    # Dump the config to files
        unless (open (MIBDUMP, ">mib_dump.txt") and
        open (WERTDUMP, ">wert_dump.txt") and
        open (WERTDEF, ">wert_def.txt")) {
        &print_debug (1, "Could not open Dump Files: $!\n");
        &print_debug (1, "Config NOT saved!\n");
    } else {
        foreach $key (sort by_hierarchy keys(%MIB)) {
            print MIBDUMP "$key/$MIB{$key}\n";

        }
        foreach $key (sort by_hierarchy keys(%Wert)) {
            unless ($key =~ /^3\.1\./ or
                    $key =~ /^3\.2\./ or
                $key =~ /^4\./) {
                print WERTDUMP "$key/$Wert{$key}\n";
            }
        }
        foreach $key (sort by_hierarchy keys(%WertDef)) {
            unless ($key =~ /^3\.1\./ or
                    $key =~ /^3\.2\./ or
                $key =~ /^4\./) {
                print WERTDEF "$key/$WertDef{$key}\n";
            }
        }
        close (MIBDUMP);
        close (WERTDUMP);
        close (WERTDEF);
    }
}

################################################
# Catch INT and TERM signals and save internal
# configuration
################################################
sub catch_term_int
{
    my $signame = shift;
    my $key;
```

```
    # Restore original signal handler
    # (in case installed signal handler hangs)
    $SIG{INT} = 'DEFAULT';
    $SIG{TERM} = 'DEFAULT';

    &print_debug(1, "\nCaught SIG$signame, saving config...");
    &dump_config_hashes();

    # Shut down and die
    &DPI_UnRegister;
    &DPI_Close;
    close (DPI_Anschluss);
    die "Terminating.\n\n";
}


#################################################
# Socket Dispatcher Funktionen
#################################################

# Variablen

# Timeout bei Socket Wait
$s_timeout = 120;

# Default Funktion
$s_deffunc = undef;

# Hash nimmt Filehandle/Funktionen Paare auf
undef(%s_funcs);

# Bitstring, der die zu pruefenden Filehandles
# aufnimmt
$s_checkstring = "";

################################################
# Funktion zum Setzen des Timeouts
################################################
sub dispatch_timeout
{
    local ($new_timeout) = @_;
    $s_timeout = $new_timeout;
}

################################################
# Funktion zum Setzen der Default Funktion
# Die hier uebergebene Funktion wird aufgerufen,
```

```
# wenn select() ein Ereignis meldet, dem File-
# handle jedoch keine Funktiion zugeordnet ist
################################################
sub dipatch_setdefault
{
    local ($func_address) = @_;
    $s_deffunc = $func_address;
}

################################################
# Funktion zum Verknuepfen einer Funktion mit
# einem Filehandle. Schlaegt select() an, so
# wird die uebergebene Funktion ausgefuehrt.
# Wird nur ein Handle, aber keine Funktion ueber-
# geben, so wird bei Anliegen von Daten auf dem
# Handle die Default Funktion ausgefuehrt (soweit
# definiert).
################################################
sub dispatch_addfunc
{
    local ($fileno, $func_address) = @_;

    # Fileno in Checkstring fuer select()
    # aufnehmen
    vec ($s_checkstring, $fileno, 1) = 1;

    # Und, falls auszufuehrende Funktion
    # uebergeben, diese in Hash aufnehmen
    if ($func_address) {
        $s_funcs{$fileno} = $func_address;
    }
}

################################################
# Funktion zum Loeschen eines Filehandles und
# der entsprechenden Funktion
################################################
sub dispatch_delfunc
{
    local ($fileno, $func_address) = @_;

    # Filehandle nicht mehr pruefen
    vec ($s_checkstring, $fileno, 1) = 0;

    # ...und Funktion aus Hash loeschen
    delete $s_funcs{$fileno};
}
```

```
################################################
# Der eigentliche Dispatcher, der select()
# aufruft und die, dem Ereignis entsprechende,
# Funktion aufruft.
################################################
sub dispatch_now
{
    local ($dispatch_timeout) = @_;
    my $i = 0;
    my $rout = "";

    unless (defined ($dispatch_timeout)) {
        $dispatch_timeout = $s_timeout;
    }

    &print_debug (3, "Calling select with timeout $dispatch_timeout");
    $isready = select ($rout = $s_checkstring, undef, undef, $dispatch_timeout);
    # Event auswerten
    &print_debug (3, "Select Call returned $isready handles with data");
    while ($isready > 0) {
        if (vec ($rout, $i, 1) == 1) {
            # Handle mit Daten gefunden!
            $isready--;
            if (defined ($s_funcs{$i})) {
                # und dazugehoerige Funktion
                &print_debug (3, "Called function for handle $i");
                &{ $s_funcs{$i} }($i);  # Aufruf
            }
            else {
                # Ohne zugehoerige Funktion
                if (defined ($s_deffunc)) {
                    # Aber mit Default
                    &print_debug (3, "Called def. function for handle $i");
                    &$s_deffunc($i);
            }
                else {
                    # Keine Funktion, kein Default
                    &print_debug (2, "Warning, uncatched event");
                }
            }
        }
        $i++;
    }
}

1;
```