BAB IV

DESAIN DAN IMPLEMENTASI

4.1 Desain

Dalam desain aplikasi ini, penulis merujuk pada *package* yang disediakan oleh *AdventNet*. Package tersebut memiliki *class-class* yang memungkinkan aplikasi untuk berkomunikasi antara *SNMP entity*.

Untuk berkomunikasi, parameter yang digunakan untuk berkomunikasi antara *SNMP entities* adalah versi dari protocol SNMP yang digunakan antara entities, alamat tujuan (*IP Address/Host*), *port* yang digunakan untuk mengirimkan *request*, penghasil pesan yang unik (*trap*), dan request identifikasi untuk setiap pesan request. *Class SnmpApi* dan *class SnmpSession* sangat membantu yang benar-benar menyediakan fungsi-fungsi yang dibutuhkan. Dan *class SnmpPDU* mereferensikan data yang akan dikirim atau diterima menggunakan *SnmpSession*.

Sebagai awalan, untuk menyiapkan komunikasi antara *SNMP entities* adalah kita harus menginisialisasi *SnmpAPI*. Berikut bentuk penulisan yang digunakan:

```
SnmpAPI api = new SnmpAPI();
api.start();
```

SnmpAPI dijabarkan pada class *java.lang.Thread*, dan *api.start()* hanya menjalankan *thread SnmpAPI*. Thread ini memonitor seluruh session untuk timeout dan *retransmission*. Applikasi *instantiate SnmpSession* digunakan untuk berkomunikasi dengan *peer SNMP entities*. Api thread akan menambahkan

timeout dan retransmission ke dalam daftar. Juga, parameter session dari user seperti tanda debug (*api.getDebug()*), default remote port (*api.SNMP_PORT*) digunakan ketika mengirim request, referensi *SNMPv3 security* dan akses kontrol konfigurasi tabel. Sebagai catatan, satu session dapat digunakan untuk berkomunikasi dengan lebih dari satu SNMP peer. Sehingga system ini akan sangat bagus sekali untuk menggunakan satu *api thread monitoring request* pada seluruh session dalam applikasi. Penggunaan SnmpSession diperlihatkan pada bagian ini:

SnmpSession session = new SnmpSession(api);

session.setPeername(String);

Applikasi mengeset peer entity untuk mengirim request dengan menggunakan perintah *setPeername()*. Attribut *remoteHost* dari *SnmpPDU* secara otomatis akan menolak peername yang terdapat dalam SnmpSession. Maksudnya adalah, ketika *remoteHost* bernilai *null* pada *SnmpPDU*, pesan akan dikirim ke host, peername, yang berada pada session. Ketika *remoteHost* tidak benilai *null*, pesan akan dikirim ke *remoteHost*. Dan ini akan selalu menjadi ide yang baik untuk mengeset peername ke dalam SnmpSession (*host*). Keadaan awal dari peername selalu *null*, dengan maksud bahwa ketika peername dalam session dan remoteHost dalam PDU bernilai null, maka "***tidak ada remote IP Address yang didaftarkan***", akan mengembalikan nilai ke *SnmpException*.

session.setRemotePort(int port);

setRemotePort() memberikan nilai pada remote port dalam peer session yang akan digunakan dalam berkomunikasi. Nilai awal dari remote port adalah *SnmpAPI.SNMP_PORT*. Parameter remotePort dalam SnmpPDU akan

memberikan suatu nilai ke session. Akan sangat baik jika kita bisa menentukan port berapa yang akan kita gunakan untuk mengirim atau menerima pesan dalam berkomunikasi. Ketika remotePort dalam *SnmpPDU* bernilai 0, maka pesan akan dikirim ke *remotePort* yang terdaftar dalam session.

session.setVersion(int version);

Method *setVersion()* akan memberikan keadaan awal versi untuk pengiriman pesan yang digunakan oleh session. *Objek PDU* dan session dibentuk dengan sebuah kondisi nilai versi awal yaitu *SNMPv1* (*SnmpAPI.SNMP_VERSION_1*). Bila ditentukan versi dari SNMP yaitu *SNMPv3 API* untuk membangun aplikasi, maka seluruh pesan dari versi *SNMPv1*, *SNMPv2* dan *SNMPv3* akan bisa dikirim dan diterima menggunakan session yang sama (akan mengabaikan seluruh seluruh versi yang diset dalam session objek). Versi dari SNMP diset ke session dengan tujuan untuk menentukan sistem pengiriman dan penerimaan pesan dalam session, jika versi dari SNMP tidak ditentukan maka secara otomatis akan diberikan nilai versi *SNMPv1*.

Peringatan: ketika applikasi mengirim *SNMPv1 pdu* menggunakan session dimana versi yang diset adalah *SNMP_VERSION_3*, maka pesan *SNMPv3* lah yang akan kirimkan ke peer. Masalahnya muncul karena API menggunakan *SNMP_VERSION_1* sebagai keadaan awal *pdu version* dan ini tidak membedakan antara versi aplikasi pada pdu sebagai default dan setting yang ditentukan *SNMP_VERSION_1*. Untuk menghindari masalah ini, aplikasi bisa mengeset session versi *SNMP_VERSION_1* dan mengeset pdu versi ke *SNMP_VERSION_2C* atau *SNMP_VERSION_3* bila ingin berkomunikasi menggunakan *v2c* dan *v3 peer*.

session.setCommunity(String community);

session.setWriteCommunity(String write_community);

Method *setCommunity()* mengeset komuniti (*community*) string pada *SNMPv1* dan *SNMPv2c*. Community string pada pdu memberikan nilai community ke session. Ini berarti, hanya ketika community string pada pdu bernilai null, maka satu dari session sedang digunakan. *WriteCommunity* menggunakan *setWriteCommunity()* digunakan hanya untuk operasi *SET*. Ketika *writeCommunity* bernilai *null*, maka community itu sendiri digunakan juga untuk operasi *SET*. Sederhananya, community string, adalah nilai dari *writeCommunity* pada pdu yang memberikan satu nilai dalam session. Nilai awal community string adalah "*public*" dan nilai awal dari *writeCommunity* string adalah null. Sehingga, aplikasi harus mengeset *writeCommunity* sebelum menggunakan aplikasi untuk operasi *SET*.

session.setLocalPort(int port);

session.setLocalAddresses(String[] addresses);

Method setLocalPort() dan *setLocalAddresses()* digunakan untuk mengeset IP Address dan port terhadap session. Nilai awal yang diberikan adaalah localhost (127.0.0.1) untuk local address dan 0 untuk local port. Sebagai catatan bahwa *setLocalAddress* akan mengambil nilai string array sebagai argument.

session.setRetries(int retries);

session.setTimeout(int timeout);

Method setRetries() dan *setTimeout()* digunakan untuk mengeset nilai pengulangan (*retries*) dan waktu menunggu dalam mili-detik (*milli-seconds*) sebelum mencoba melakukan pengiriman atau penarikan ulang. Nilai timeout

bergerak exponensial setelah pengiriman ulang yang pertama. Sebagai contoh, jika timeout bernilai 5000 (berarti 5 detik) dan pengulangan di set 3, pengiriman ulang pertama akan terjadi setelah 5 detik, berikutnya pada 15 detik dan seterusnya. Seperti parameter session yaitu remote host, port dan sebagainya, pengulangan dan timeout pada pdu memberikan nilai pada session. Nilai awal adalah 0 untuk pengulangan (berarti tidak mengirim ulang jika timeout) dan 5000 untuk timeout (berarti menunggu hingga 5 detik baru timeout akan terjadi).

session.setUserName(byte[] name);

Method setUserName() sangat dibutuhkan untuk menggunakan *SNMPv2* message. Konfigurasi security dapat dipakai untuk digunakan dalam autentifikasi pesan sebelum dikirim ke SNMP peer. userName adalah tidak terlalu dibutuhkan pada komunikasi *SNMPv1* dan *SNMPv2c* menggunakan API. Nilai awal dari userName adalah "initial".

session.setTrapAuthEnable(Boolean isAuth);

setTrapAuthEnable() mengontrol apakah trap harus di autentifikasi bila pesan trap SNMPv3 diterima. Nilai awalnya adalah false, yang berarti tidak dibutuhkan autentifikasi trap dan notification.

session.setSocketParms(int socketTimeout, int socketDelay);

setSocketParms() mengeset parameter socket yang digunakan dalam berkomunikasi dengan session tersebut. Nilai awal dari *socketTimeout* adalah 250 ms, ini berarti *sock.receive()* memblok hanya untuk durasi *socketTimeout* dan akan mengaktifkan *java.io.InteruptedIOException*. Parameter *socketDelay* mengontrol waktu tunggu antara pemanggilan *sock.receive()* ketika *java.io.InteruptedIOException* aktif. Nilai awal dari *socketDelay* adalah 0 ms,

yang berarti penerimaan thread dalam SnmpSession akan mengembalikan penerimaan data tanpa menunggu antara pemanggilan interupsi *sock.receive()*.

```
try {
    session.open();
} catch (SnmpException e) {
    System.err.println("Error opening session: " + e.getMessage());
    System.exit(1);
}
```

Method open() akan membuka sebuah socket untuk berkomunikasi dengan SNMP entity yang lain dan mulai menerima thread untuk menerima dan mengolah pesan. Untuk mengaktifkan applet selalu melakukan *open(Applet applet)* untuk menggunakan session. Pada bagian ini kita tidak menggunakan dan tidak membahas *method open(Applet applet)*.

Sekali session telah dijalankan, aplikasi akan mengubah seluruh parameter komunikasi, termasuk yang telah dijelaskan di atas. Ketika lokal port, lokal address atau *socketParameters* dirubah, maka aplikasi harus menutup session dan membukanya kembali untuk mengaktifkan seluruh perubahan yang terjadi. Pengubahan terhadap parameter yang lain seperti *retries, timeout, community, writeCommunity, version* dan *remotePort* akan memberikan efek perubahan pada seluruh nilai dalam pengiriman pesan. Urutan perintah yang dibutuhkan ketika local host, port atau parameter socket berubah adalah sebagai berikut:

```
session.setSocketParms(int newSocketTimeout, int newSocketDelay);
session.setLocalPort(int newPort);
session.setLocalAddress(String[] newAddress);
```

```
session.close();
Try {
    session.open();
} catch (SnmpException e) {
    System.err.println("Error opening session: " + e.getMessage());
    System.exit(1);
}
```

4.1.1 Penanganan timeout dan pengiriman ulang

Ketika request telah diberikan ke session, pdu akan menambahkan ke daftar request (*requestList*) untuk menunggu ditanggapi. Ketika response diterima, daftar request tersebut dihapus dari *requestList* dan response akan ditambahkan ke daftar response (*responseList*). *RequestList* dan *responseList* keduanya akan berhubungan dengan session dan akan memperbaharui ketika mengirim dan menerima pesan menggunakan session. Disamping itu, *api thread* akan mengamati seluruh session terhadap timeout dan menambahkannya ke dalam daftar timeout (*timeoutList*) di session. Session ini akan menggunakan *timeoutList* untuk mengirim ulang pesan jika aplikasi akan mengirim ulang dimana pdu bernilai *non-zero*.

Aplikasi dapat mengontrol seluruh pengiriman ulang dan timeout menggunakan method *setTimeout()* dan *setRetries()* yang tersedia pada *SnmpSession class*. Aplikasi juga dapat memberikan suatu nilai pada timeout dan retries dengan menyeting setiap pdu yang akan dikirimkan.

Aplikasi juga dapat menangani timeout dan proses pengiriman ulang tanpa menggunakan servis dari API, jika dilakukan hal tersebut maka nilai awal yang

akan diberikan adalah 0. Ketika aplikasi menggunakan *multiple session* untuk berkomunikasi antara peer, aplikasi dapat menggunakan method untuk mengecek reaksi dan timeout. Proses tersebut adalah sebagai berikut.

api.checkResponse();

api.checkTimeout();

api.checkResponse() akan mengambil daftar response dalam session yang ada dalam daftar. Dan *api.checkTimeout()* akan mengambil daftar timeout dalam session yang ada dalam daftar dimana request yang mengalami timeout. Pada saat session mendapat response, aplikasi dapat menggunakan *session.checkResponse()* untuk mendapatkan response tersebut. Begitu pula *session.checkTimeout()* akan digunakan untuk mendapatkan request yang mengalami timeout.

Aplikasi akan selalu menggunakan method *session.send(SnmpPDU)* untuk mengirim sebuat request. Ketika session.method *syncSend(SnmpPDU)* untuk mengirim dan menerima *synchronous PDU*, timeout akan ditangani secepatnya dalam API, dan *syncSend()* akan bernilai null jika timeout terjadi. *Method api.checkResponse()* dan *api.checkTimeout()* akan tidak berfungsi bila pesan dikirim atau diterima menggunakan *syncSend(SnmpPDU) (synchronously).*

4.1.2 Operasi pengambilan data SNMP yang digunakan

Class SnmpV3Message (API) berfungsi untuk pertukaran data antara *SNMP peer entities*. *SnmpV3Message* terdiri dari SnmpMessage (API) yang berfungsi untuk pemakaian *SNMPv1* dan *SNMPv2*. Class SnmpPDU berfungsi untuk melakukan pertukaran data antara SNMP entities. *SnmpPDU* dibungkus dalam sebuah pesan, dimana pesan dapat berasal dari *SNMPv1, SNMPv2c* atau *SNMPv3*. Aplikasi tidak akan memperhatikan pesan dari *class Snmp3Message* dan

SnmpMessage, dan aplikasi dapat bekerja sendiri dengan *SnmpPDU* untuk berinteraksi dengan *peer*. SnmpPDU menyediakan *method getMsg()* untuk mengakses *SnmpMessage*.

SnmpPDU sebagian besar menyediakan parameter komunikasi yang berhubungan dengan *SnmpSession*. Dimanapun nilai parameter yang diberikan pada pdu, ini akan memberikan nilai pada session. Beberapa hal yang akan dihasilkan oleh *SnmpPDU* adalah:

a. Method untuk bekerja dengan *variable bindings*.
b. Method yang berhubungan untuk seting PDU yang digunakan oleh operasi SNMP.
c. Method untuk mengaktifkan penggunaan objek *SnmpPDU* untuk *multiple request*.

Method – method yang ada pada *SnmpPDU* yang bekerja dengan variable bindings diantaranya adalah:

a. *addNull(SnmpOID oid);*
b. *addVariableBinding(int index, SnmpVarBind varbind)*
c. *void removeVariableBinding(int index)*
d. *addVariableBinding(SnmpVarBind varbind)*
e. *void removeVariableBinding(SnmpVarBind varbind)*
f. *SnmpOID getObjectID(int index)*
g. *SnmpVar getVariable(int index)*
h. *void setVariable(int index, SnmpVar var)*
i. *SnmpVarBind getVariableBinding(int index)*
j. *Vector getVariableBindings()*

k. *String printVarBinds()*

Method – method untuk menyeting yang berhubungan dengan SnmpPDU adalah:

a. *byte getCommand()*

b. *void setCommand(byte type)*

c. *int getReqid()*

d. *void setRegid(int id)*

Operasi *GetBulk* yang berhubungan dengan parameter v2c & v3:

a. *int getMaxRepetitions()*

b. *int getNotRepeaters()*

c. *void setMaxRepetitions(int max_rep)*

d. *void setReqid(int id)*

Model parameter akses kontrol yang berhubungan dengan SNMPv3:

a. *byte[ ] getContextID()*

b. *byte[ ] getContextName()*

c. *void setContextID(byte[ ] id)*

d. *void setContextName(byte[ ] name)*

Indikator *Exception* pada SnmpPDU:

a. *int getErrindex()*

b. *string getError()*

c. *int getErrstat()*

d. *void setErrindex(int index)*

e. *void setErrstat(int stat)*

Pada saat *SnmpPDU* di *setup* menggunakan method – method tersebut diatas, *SnmpPDU* akan di kirim melalui session ke *SNMP peer entity*. *SnmpSession* menyediakan method-method untuk berinteraksi dengan peer dengan urutan sebagai berikut:

a. *syncSend(pdu)* untuk mengirim *synchronous request* dan menerima response.
b. *send(pdu)* untuk mengirim request dan method *checkResponses(), checkTimeout(int reqid)* dan *receive(int reqID)* untuk menerima *asynchronous response*.
c. Gunakan *send(pdu)* untuk mengirim request dan gunakan method *callback* untuk mengambil dan memroses pesan.

4.1.3 Fasilitas sending/receiving TRAPS dan NOTIFICATIONS

Method – method trap dan notifikasi yang terdapat pada class SnmpPDU adalah:

a. *SnmpOID getEnterprise()*
b. *int getSpecificType()*
c. *int getTrapType()*
d. *void setEnterprise(SnmpOID oid)*
e. *void setSpecificType(int type)*
f. *void setTrapType(int type)*
g. *InetAddress getAgentAddress()*
h. *void setAgentAddress(InetAddress addr)*
i. *void setUpTime(long uptime)*
j. *long getUpTime()*

4.1.4 Pemrosessan pesan dan autentifikasi

Tatap muka *SnmpClient (API)* digunakan oleh aplikasi yang akan melakukan pengiriman dan penerima pesan *asynchronous*, atau aplikasi yang akan melakukan proses autentifikasi. Tatap muka SnmpClient menyediakan *method callback* untuk mengecek terhadap response. *Method callback()* secara otomatis dijalankan ketika response diterima. Aplikasi akan meminta session untuk menggirim pesan terhadap method *callback()*. Jika user mengeset *session.setCallbackThread(true)* maka callback method akan diberikan tanda terhadap *thread* yang berbeda ketika respon diterima. Jika *session.setCallbackThread(false)* (defaut), maka callback method akan diberi tanda pada thread yang sama dari thread yang diterima dan respon berurutan dapat diterima hanya jika user kembali dari callback method. Tetapi pemanggilan fungsi dari thread yang berbeda, kemampuan penerimaan thread pada penerimaan respon atau trap akan sedikit jelek. Untuk melakukan pemanggilan balik ketika respon diterima, aplikasi harus menggunakan SnmpClient dan meregisterkan dengan SnmpSession menggunakan *method addSnmpClient( )*.

4.1.5 Pemeliharaan sekuriti dan parameter akses kontrol

Setiap entiti *SNMPv3* akan mengatur konfigurasi informasi yang berhubungan untuk memulai kembali pengolahan data. Contoh untuk sebuah informasi yaitu *engineID*. Parameter *engineID* dan *engineBoots* menyimpan nilai (*count*), berapa kali *entity SNMPv3* diulang. Parameter *engineBoots* digunakan untuk menjalankan pengecekan berkala, untuk memastikan bahwa pesan telah sampai tanpa melakukan pengulangan. *SnmpAPI* menyediakan method untuk mendukung hal tersebut diatas.

a. *api.setSnmpEngineID(byte[ ] id);*

b. *api.setSnmpEngineBoots(int boots);*

c. *api.setSnmpEngineTime(int time);*

d. *api.setTimeWindow(int win);*

Method setTimeWindow() digunakan untuk mengeset *timewindow*, dimana waktu (*time*) dengan pesan harus diterima setelah pengiriman. Ini diperlukan untuk pertimbangan pada waktu *request/response*. Aktualisasi cek memaksa ketika dibutuhkan autentifikasi (*SnmpAPI.AUTH_NO_PRIV* atau *SnmpAPI.AUTH_PRIV*) untuk berkomunikasi dengan *SNMPv3 peer*.

api.setSerializeFileName(String name);

api.serialize();

Method setSerializeFileName() mengisi nama dari file dari seluruh konfirugasi yang disimpan. *Method serialize()* menyimpan konfigurasi saat ini dari *SNMPv3 entity.*

api.deSerialize();

Method deSerialize() menghasilkan konfigurasi yang ada dalam *serialize file* terhadap *SNMPv3 entity*. Tabel konfigurasi *USMUserTable (User based Security Model), VACM (View based Access Control Model)*, dan *engineBoots* dapat di *serialize* dan *de-serialize* oleh API.

Begitupula *class SnmpAPI* menyediakan method untuk mengakses tabel konfigurasi SNMPV3 yang berbeda seperti *USMUserTable, VacmAccessTable* dan sebagainya. Aplikasi dapat menambahkan atau memodifikasi entity yang ada ke dalam tabel. Aplikasi juga dapat membaca (*read*) konfigurasi melalui external datasource dan menambahkan entity ke dalam tabel. Dengan cara ini, aplikasi

dapat menyimpan seluruh konfigurasi *security* ke dalam database tersentral dan memegang tabel ketika aplikasi *starts-up*.

Aplikasi dapat menambahkan ke dalam *USMUserTable* sebagai berikut:

a. Mendapatkan referensi ke objek *USMUserTable*

USMUserTable usmtable = (USMUserTable)api.getSecurityProvider(). getTable(USE_SECURITY_MODEL);

b. Membentuk *USMUserEntry* baru dan menentukan parameter yang berbeda

USMUserEntry entry = new USMUserEntry(byte[] usr, byte[] id);

entry.setAuthProtocol(int protocol);

c. Tambahkan *USMUserEntry* ke *USMUserTable*

usmtable.addEntry(entry);

Aplikasi bisa mengikuti urutan perintah dibawah untuk memodifikasi *USMUserTable*.

a. Mendapatkan referensi ke objek *USMUserTable*

USMUserTable usmtable = (USMUserTable)api.getSecurityProvider(). getTable(USE_SECURITY_MODEL);

b. Mendapatkan referensi ke objek *USMUserEntry* yang dibutuhkan untuk modifikasi

USMUserEntry entry = usmtable.getEntry(byte[] name, byte[] id);

entry.setAuthProtocol(int protocol);

c. Kembalikan hasil perubahan ke *USMUserTable*.

usmtable.modifyEntry(entry);

4.1.6 Pengaturan berbagai macam counter yang dideskripsikan dalam group SNMP

API mengatur group SNMP counter, seperti yang dijabarkan pada *RFC1213-MIB*, untuk setiap entiti SNMP dalam aplikasi. Setiap entity SNMP diidentifikasikan dengan parameter local address dan local port. Aplikasi dapat mengakses group SNMP menggunakan:

api.getSnmpGroup(String local_address, int port);

Pada saat objek *SnmpGroup* diakses, objek tersebut dapat digunakan untuk mendapatkan nilai dari counter yang berbeda.

SnmpGroup groupCounters = api.getSnmpGroup(String local_address,
int port);
groupCounters.getSnmpInBadVersions( );

API tidak melakukan perubahan pada empat fariabel counter dibawah dan aplikasi yang akan melakukan penambahan secara otomatis.

snmpInBadCommunityNames

snmpInBadCommunityUses

snmpInTotalReqVars

snmpInTotalSetVars

Aplikasi dapat menggunakan method-method dibawah untuk melakukan update terhadap counters.

SnmpGroup groupCounters = api.getSnmpGoup(String local_address, int port);

gourpCounters.incrSnmpInBadCommunityNames( );

gourpCounters.incrSnmpInBadCommunityUses( );

gourpCounters.incrSnmpInTotalReqVars( );

groupCounters.incrSnmpInTotalSetVars( );

Dalam manajemen aplikasi yang sederhana, counter digunakan khusus untuk agent seperti *snmpInGetNexts*, *snmpInSetRequest* dan sebagainya yang memberikan nilai 0.

4.1.7 Beberapa definisi konstanta yang dibutuhkan aplikasi

Aplikasi menggunakan nilai konstanta yang dibutuhkan pada operasi-operasi yang berbeda dalam SNMP, nilai error-status pada pesan tanggapan, berbagai tipe objek dan sebagainya.

Tiga konstanta yang ada di *SnmpAPI class* adalah:

Tabel 4.1. Daftar konstanta SnmpAPI class

CLASS	KETERANGAN
SnmpAPI.SNMP_VERSION_1	Snmp versi 1
SnmpAPI.SNMP_VERSION_2	Snmp versi 2
SnmpAPI.SNMP_VERSION_2C	Snmp versi 2c
SnmpAPI.SNMP_VERSION_3	Snmp versi 3

Operasi SNMP yang berhubungan, yang berada pada *SnmpAPI* adalah:

Tabel 4.2. Daftar operasi SnmpAPI

CLASS	KETERANGAN
SnmpAPI.GET_REQ_MSG	Konstanta untuk GET Request PDU.
SnmpAPI.GET_RSP_MSG	Konstanta untuk GET Response PDU.

SnmpAPI.GETBULK_REQ_MSG	Konstanta untuk GETBULK request PDU.
SnmpAPI.GETNEXT_REQ_MSG	Konstanta untuk GETNEXT request PDU.
SnmpAPI.INFORM_REQ_MSG	Konstanta untuk INFORM request PDU.
SnmpAPI.TRP_REQ_MSG	Konstanta untuk TRAP PDU.
SnmpAPI.TRP2_REQ_MSG	Konstanta untuk NOTIFICATION PDU.
SnmpAPI.SET_REQ_MSG	Konstanta untuk SET request PDU.
SnmpAPI.Standard_Prefix	Standar prefix digunakan jia OID tidak diberikan oleh root.

Tabel 4.3. Daftar variabel error SnmpAPI

CLASS
SnmpAPI.SNMP_ERR_NOERROR
SnmpAPI.SNMP_ERR_TOOBIG
SnmpAPI.SNMP_ERR_NOSUCHNAME
SnmpAPI.SNMP_ERR_BADVALUE
SnmpAPI.SNMP_ERR_READONLY
SnmpAPI.SNMP_ERR_GENERR
SnmpAPI.SNMP_ERR_AUTHORIZATIONERROR
SnmpAPI.SNMP_ERR_COMMITFAILED
SnmpAPI.SNMP_ERR_INCONSISTENTNAME
SnmpAPI.SNMP_ERR_INCONSISTENTVALUE
SnmpAPI.SNMP_ERR_NOACCESS
SnmpAPI.SNMP_ERR_NOCREATION
SnmpAPI.SNMP_ERR_NOTWRITABLE
SnmpAPI.SNMP_ERR_RESOURCEUNAVAILABLE
SnmpAPI.SNMP_ERR_UNDOFAILED
SnmpAPI.SNMP_ERR_WRONGENCODING
SnmpAPI.SNMP_ERR_WRONGLENGTH
SnmpAPI.SNMP_ERR_WRONGTYPE
SnmpAPI.SNMP_ERR_WRONGVALUE

4.1.8 Variabel class

Class SnmpVar ada dalam *AdventNet SNMP API package* yang merupakan base class dari seluruh variable class SNMP. *SnmpVar* adalah sebuah class yang menyediakan method-method yang akan digunakan aplikasi untuk bekerja dengan

fariabel SNMP. Class ini menyediakan method untuk mencetak (*print*), *ASN encoding*, *ASN decoding* dan sebagainya, tetapi tidak semuanya merupakan public. *Class SnmpVar* diklasifikasikan dalam sub class yaitu *sub class SnmpString* dan *sub class SnmpUnsignedInt*. Dibawah ini akan di jabarkan daftar class SnmpVar yang digunakan:

Tabel 4.4. Daftar direct sub-class dari SnmpVar

CLASS	KETERANGAN
SnmpInt	Digunakan untuk mewakili variabel Integer.
SnmpNull	Digunakan untuk mewakili variabel NULL.
SnmpOID	Identifikasi objek dalam SNMP. Class ini digunakan untuk berhubungan dengan MIB.
SnmpString	Digunakan untuk SNMP Oktet String.
SnmpUnsignInt	Merupakan super-class dari SNMP Aplikasi.
SnmpBitstring	Digunakan untuk mendefinisikan variabel SNMP BITSTRING. Juga untuk mendukung penggunaan Bitstring.
SnmpCounter64	Untuk tipe variabel SNMP Counter 64.

Tabel 4.5. Daftar SnmpString sub-class

CLASS	KETERANGAN
SnmpOpaque	-
SnmpIpAddress	Digunakan untuk tipe variabel SNMP IP Address.
SnmpNetworkAddress	Digunakan untuk tipe variabel Netword Address.
SnmpNsap	Digunakan untuk tipe variabel SNMP NSAP Address.
SnmpBits	Digunakan untuk membentuk objek dari SnmpString. SnmpBits juga memiliki method untuk mengambil nilai dari form yang berbeda. (spt, String, Byte).

Tabel 4.6. Daftar SnmpUnsignedInt sub-class

CLASS	KETERANGAN
SnmpCounter	Digunakan untuk tipe variabel SNMP Counter.

SnmpGauge	Digunakan untuk tipe variabel SNMP Gauge.
SnmpTimeticks	Digunakan untuk tipe variabel SNMP Timeticks.

Dibawah ini akan di jabarkan tentang method-method yang menyediakan tipe data dan sintak (tipe string) dari data.

Method getType() dan *getTypeString()* ada dalam SnmpVar yang menyediakan cara untuk mendapatkan tipe dan sintak data. Tipe di definisikan dalam SnmpAPI seperti yang dijabarkan diatas yaitu *SnmpAPI.INTEGER*, *SnmpAPI.STRING* dan sebagainya. *Method getTypeString()* mengembalikan nilai string seperti "*INTEGER*", "*BITSTRING*" dan sebagainya, yang didasarkan pada objek SnmpVar.

Class SnmpVar terdiri dari abstrak method yang digunakan terhadap seluruh variabel class. Berikut rinciannya.

A. Method toString( ) dan toTagString( )

Method toString() akan mengubah data kedalam bentuk *printable string*. Sebagai contoh, bila *method toString()* digunakan terhadap objek *SnmpIpAddress()*, maka akan dikembalikan nilai IP Address yang bernilai string ("127.0.0.1"), dan jika *method toString()* digunakan terhadap objek *SnmpCounter64*, maka akan dikembalikan nilai string "0x" yang diikuti data Counter64 dalam format hexadecimal. *Method toString()* biasanya digunakan oleh aplikasi untuk mencetak nilai dari objek SnmpVar. Berikut adalah contoh penggunaannya.

```
SnmpVar var = pdu.getVariable(0);
System.out.println("Type =" + var.getTypeString() + ":Value =" +
var.toString());
```

Potongan kode diatas akan menghasilkan output seberti ini, jika objek *SnmpIpAddress* bernilai lokal IP Address.

```
Type =IPADDRESS:Value =127.0.0.1
```

Potongan kode diatas akan menghasilkan tipe data Counter64 dan bernilai 255, jika method yang digunakan *toTagString()*.

```
Type =COUNTER64:Value =0x0255
```

B. Method toValue( ) dan toVarObject( )

Method toValue() dan *toVarObject()* dalam *class SnmpVar* memberikan nilai terhadap variabel dari masing-masing tipe Objek. Method ini melakukan hal yang sama terhadap seluruh class SnmpVar kecuali *SnmpIpAddress*, *SnmpNetworkAddress* dan *SnmpOID*. Berikut tabel yang menampilkan perbedaan antara *method toValue()* dan *getVarObject()* diantara ketiga class berikut.

Tabel 4.7. Daftar perbedaan method toValue( ) dan getVarObject( )

S.No	CLASS NAME	toValue()	getVarObject()
1	SnmpIpAddress	Menghasilkan String IP Address byte[ ] (7F000001)	Menghasilkan String IP Address dalam notasi titik. (127.0.0.1)
2	SnmpNetworkAddr	Sama seperti point 1.	Sama seperti point 1.
3	SnmpOID (untuk sysDesc dalam RFC1213-MIB)	Menghasilkan int[ ] dari nilai OID. (int[ ] = {1, 3, 6, 1, 2, 1, 1, 1})	Menghasilkan titik OID sebagai objek String. ("1, 3, 6, 1, 2, 1, 1, 1")

Untuk seluruh class selain dari ketiga class diatas (*SnmpIpAddress*, *SnmpNetworkAddress*, dan *SnmpOID*) akan menghasilkan nilai yang sama sebagaimana dijabarkan dalam tabel berikut.

Tabel 4.8. Daftar persamaan

S.No	CLASS NAME	toValue() / getVarObject()
1	SnmpBitString	Menghasilkan object String yang merepresentasikan nilai byte.
2	SnmpCounter64	Menghasilkan long[2] yang merepresentasikan nilai Counter64.
3	SnmpNull	Selalu menghasilkan nilai NULL.
4	SnmpInt	Menghasilkan object Integer.
5	SnmpUnsignedInt SnmpTimeticks SnmpCounter SnmpGauge	Menghasilkan object Long.
6	SnmpString SnmpNsap SnmpOpague SnmpBits	Menghasilkan object String.

C. Method toBytes()

Method toBytes() memberikan nilai sebagai bentuk baris-baris byte terhadap seluruh class SnmpVar. Seluruh sub-class *SnmpString, seperti*

SnmpNsap, SnmpOpaque, *SnmpIpAddress, SnmpNetworkAddress* dan *SnmpBits* akan memberikan nilai byte array.

Seluruh sub-class *SnmpUnsignedInt, seperti SnmpTimeticks, SnmpCounter* dan *SnmpGauge* menghasilkan byte array. Contoh, *SnmpUnsignedInt* bernilai 0xDEADBE EF, nilai byte array akan memiliki nilai:

Byte[0] = 0xDE, byte[1] = 0Xad, byte[2] = 0XBE, dan byte[3] = 0xEF

Method toBytes() dari *objek SnmpCounter64* menghasilkan byte array dengan nilai dari *objek Counter64*. Panjang data yang dihasilkan selalu berjumlah 8 dimana byte[0] akan menunjuk pada urutan yang terakhir sedangkan byte[7] akan menunjuk pada urutan pertama. Sebagai contoh, *objek Counter64* bernilai 0x0102030405060708, maka *method toBytes()* akan bernilai:

byte[ ] = {8, 7, 6, 5, 4, 3, 2, 1}

Method statik *createVariable()* tersedia dalam *class SnmpVar*, fungsinya membentuk objek variabel SNMP. Beberapa statik variabel yang disediakan adalah sebagai berikut.

Tabel 4.9. Daftar statik variabel

VARIABLE		
SnmpAPI.INTEGER	SnmpAPI.STRING	SnmpAPI.BITSTRING
SnmpAPI.OBJID	SnmpAPI.NULLOBJ	SnmpAPI.NETWORK ADDSS
SnmpAPI.COUNTER	SnmpAPI.GAUGE	SnmpAPI.UNSIGNED32
SnmpAPI.TIMETICKS	SnmpAPI.OPAQUE	SnmpAPI.UNITEGER32
SnmpAPI.NSAP	SnmpAPI.COUNTER64	

Jika variabel tidak dapat di inisialisasi, pesan kesalahan akan dilemparkan ke SnmpException.

4.2 Implementasi Aplikasi

Melanjutkan dari pembahasan sebelumnya mengenai desain, maka pada bab ini desain tersebut akan diimplementasikan dalam bentuk sebuah aplikasi. Aplikasi ini dibuat dengan menggunakan bahasa java Applet. Applet diperlukan untuk memberikan user interface yang mudah untuk di mengerti oleh user dan mudah dalam penggunaannya.

4.2.1 Persiapan Awal

Spesifikasi komputer yang diperlukan dalam mengoperasikan aplikasi monitoring ini adalah prosessor minimum Pentium 233 MHz, memori minimum 64MB RAM, dan disk space minimum 50 MB.

Sedangkan perangkat lunak (*software*) yang diperlukan adalah sebagai berikut.

a. *JDK 1.1.6* atau *JDK 1.3*. Untuk sistem operasi Windows 95/98/NT/XP/ Solaris bisa didapatkan dari web site *Sun's Javasoft* (http://www.javasoft.com). Sedangkan untuk sistem operasi Linux bisa didapatkan di http://www.blackdown.org. Atau bisa langsung menjalankan file jdk (j2sdk-1_3_1_06-windows-i586.exe) yang sudah disediakan pada CD Proyek Tugas Akhir dalam folder "j2se_131".

b. Web browser yang mendukung Java (Netscape 4.x / IE 4.x) dengan Sun's Java Plug-in, untuk menjalankan applet menggunakan web browser yang di buat mengguanakan SNMP API. Plug-in tersebut dibutuhkan untuk menjalankan applet yang mana komponennya di sediakan oleh komponen *swing* atau *JFC*.

Java Plug-in ini bisa di dapatkan pada site http://java.sun.com/products/plugin/1.1.1 /index.html.

Perlu diperhatikan bahwa patch JDK 1.1 untuk Netscape 4.x harus di dibuang sebelum memasang Sun's Java Plug-in. Jika tidak dilakukan, maka Patch Netscape JDK 1.1 akan konflik dengan Java Plug-in dari Sun.

c. Source file aplikasi Agent yang juga sudah disertakan pada CD Proyek Tugas Akhir. Pada folder "agent" sub folder "win" merupakan file instalasi untuk Sistem Operasi Windows sedangkan pada sub folder "linux" merupakan file instalasi untuk Sistem Operasi linux.

4.2.2 Instalasi

Sebelum aplikasi digunakan, diperlukan persiapan dasar seperti melakukan instalasi JDK 1.3, registrasi library dan pengaktifan service SNMP Agent pada komputer lokal jika pengujian dilakukan pada komputer lokal. Untuk mempermudah, penulis akan memisahkan tahap-tahapnya mulai dari instalasi JDK 1.3, pengesetan CLASSPATH, dan instalasi aplikasi.

a. Jalankan file j2sdk-1_3_1_06-windows-i586.exe yang bisa di download pada web site http://java.sun.com/downloads/index.html atau pada CD Proyek Tugas Akhir. Ikuti petunjuk yang diperlihatkan pada saat instalasi dan tentukan letak instalasi pada drive file sistem.

b. Pada CD Proyek Tugas Akhir, sudah disiapkan satu file untuk mendaftarkan CLASSPATH dan mendaftarkan posisi java home yang nantinya diperlukan untuk menjalankan aplikasi. Dalam hal ini file yang sudah disiapkan adalah file *startMibBrowser.bat* dimana isi dari file tersebut adalah sebagai berikut.

set JAVA_HOME_DIR = c:\jdk1.3

set PATH = %JAVA_HOME_DIR%\bin;%PATH%

set CLASSPATH = .;..\classes;..\classes\lib\MibBrowser.jar

Pengguna bisa merubah isi dari file *startMibBrowser.bat* untuk menyesuaikan posisi java home. Dalam hal ini, dicontohkan posisi *java home* ada di *c:\jdk1.3*

c. Untuk menjalankan aplikasi MIB Browser, diperlukan *source file* MIB Browser yang ada pada CD Proyek Tugas Akhir. Seluruh file berada pada folder "src-app" sub folder "appMibBrowser". *Copy* seluruh file pada folder tersebut dan letakkan pada drive file sistem yang diinginkan, sebagai contoh di letakkan pada drive "C:" atau "D:". Kemudian jalankan file *startMibBrowser.bat.*

d. Untuk menjalankan aplikasi Agent, bisa langsung mengaktifkan service SNMP Agent (jika Sistem Operasi Win 2000/XP) yaitu dengan menambahkan komponen pada *Windows Setup*. Urutan penambahan service tersebut adalah sebagai berikut: *Start Menu* → *Program* → *Control Panel* → *Add/Remove Programs* → *Add/Remove Windows Components* → *Windows Components Wizard*, pada tab *Windows Components Wizard* ini, *double click* pada *Management and Monitoring Tools* → pilih *Simple Network Management Protocol*, tekan tombol *OK* hingga komponen berhasil terinstal. Sedangkan untuk Sistem Operasi Linux bisa langsung diaktifkan pada saat instalasi Sistem Operasi. Atau jika kondisi Sistem Operasi tidak terdapat komponen untuk menambahkan service SNMP Agent, jalankan file-file yang berada pada folder "agent" dalam CD Proyek Tugas Akhir (sesuaikan Sistem Operasi apa yang digunakan terhadap source di folder "agent" tersebut).

4.2.3 Struktur direktori dan file

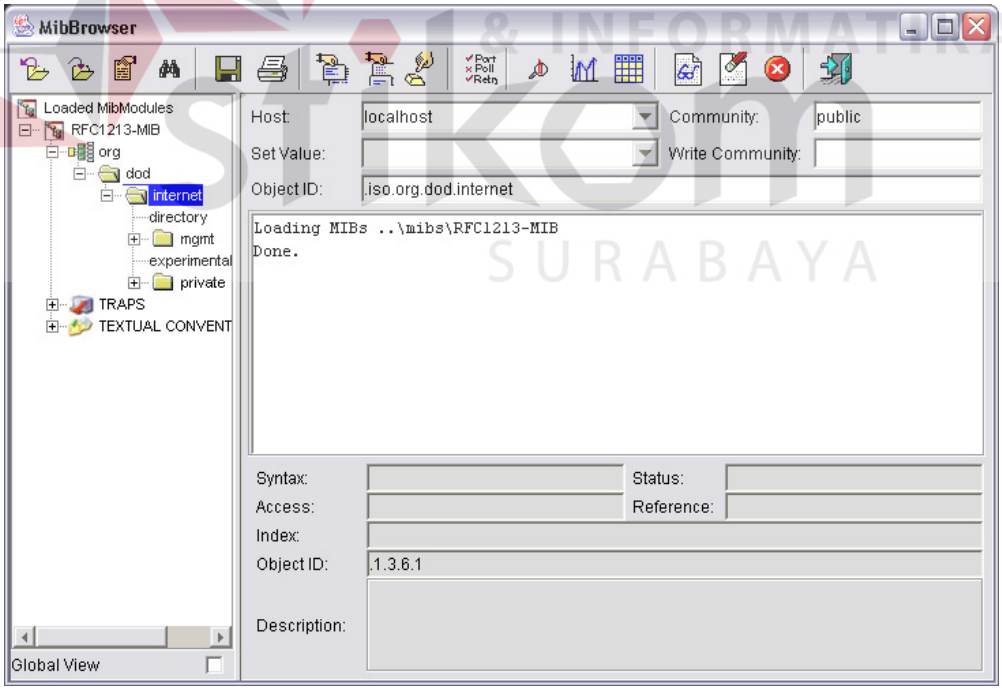
Struktur direktori aplikasi pada folder “src-app” (CD Proyek Tugas Akhir) dalam ruang lingkup aplikasi ini akan dijelaskan sebagai berikut.

Tabel 4.10. Daftar struktur file

DIREKTORI	KETERANGAN
classes	meyimpan seluruh file class dari aplikasi yang sudah di compile
lib	meyimpan seluruh library yang diperlukan aplikasi
src	meyimpan seluruh file java dari aplikasi
mibs	menyimpan file-file RFC

4.2.4 Pengoperasian aplikasi

File *startMibBrowser.bat* diperlukan untuk menjalankan aplikasi, dimana tampilan akan terbentuk berupa applet seperti yang diperlihatkan berikut ini.



Gambar 4.1. Tampilan awal

Gambar 4.1 diatas terbagi dalam 2 bagian (*frame*). Frame pertama memperlihatkan *MIB Tree* dimana file MIB ditampilkan. Sedangkan frame kedua digunakan untuk pengaturan (*setting options*) dan untuk menampilkan hasil proses. Pada frame ke dua ini terdiri dari bagian-bagian yaitu:

a. Nama atau alamat *SNMP agent*
b. Nama komuniti
c. Nilai *writeCommunity* digunakan untuk *set request*
d. Posisi aktif Node OID yang akan dioperasikan terhadap *MIB Module*
e. *Context Name* dan *Context Engine ID* untuk *SNMPV3 request*
f. Dan tampilan untuk menampilkan hasil *query*

Berikut akan dijabarkan bagian dan fungsi dari *Toolbar* dan *Menu* pada tampilan dari gambar 4.1 diatas.

A. Toolbar

Fungsi dari masing-masing icon toolbar tersebut adalah sebagai berikut.

a. Icon  digunakan untuk membuka/mengambil (*load*) file MIB ke dalam aplikasi. Memilih icon ini akan menampilkan dialog untuk memfasilitasi pengambilan file.
b. Icon  digunakan untuk membuang MIB dari aplikasi (*unload*).
c. Icon  digunakan untuk menampilkan informasi dari node yang dipilih.
d. Icon  digunakan untuk mencari node dalam *MIB Tree*. Akan ditampilkan dialog untuk mengisikan spesifikasi node yang akan dicari.

e. Icon  digunakan untuk menyimpan hasil query dari MIB ke dalam format file .txt.

f. Icon  digunakan untuk mencetak hasil query dari MIB. Akan ditampilkan dialog untuk setting printer.

g. Icon  digunakan untuk operasi *GET*. Operasi ini akan mengambil seluruh objek yang ada pada aktif objek MIB, atau spesifik objek jika objek yang dipilih adalah node MIB.

h. Icon  digunakan untuk operasi *GETNEXT*. Operasi ini akan mengambil objek berikutnya setelah objek yang dipilih, atau spesifik objek jik aobjek yang dipilih adalah node MIB.

i. Icon  digunakan untuk operasi *GETBULK*. Operasi ini akan mengambil sekuen dari objek berikutnya setelah objek yang dipilih.

j. Icon  digunakan untuk operasi *SET*. Operasi ini akan memberikan nilai kepada MIB dimana nilai bisa diisikan melalui ***Set Value***. Pengisian untuk tipe *Octet String* dalam format *hexadesimal* adalah dengan memisahkan setiap byte dengan titik dua (:). Sebagai contoh Octet String **0xff0a3212**, diisikan dengan format '**ff:0a:32:12**' dalam *Set Value*.

k. Icon 

digunakan untuk seting *Snmp Port*, *Snmp Version*, *Timeout*, *Retries* dan sebagainya. Nilai awal yang diberikan untuk port, version, timeout dan retries adalah 161, v1, 5 sec (detik) dan 0.

l. Icon  digunakan untuk menampilkan penerimaan *Trap* pada port yang diterpakai.

m. Icon  digunakan untuk menampilkan real-time grafik secara periodik sesuai dengan interval waktu yang ditentukan.

n. Icon  digunakan untuk menampilkan tabel SNMP.

o. Icon  digunakan untuk menampilkan hasil debug. Selama dialog debug terbuka, proses debug akan aktif dan proses debug akan dihentikan jika dialog di tutup.

p. Icon  digunakan untuk membersihkan layar hasil seluruh query.

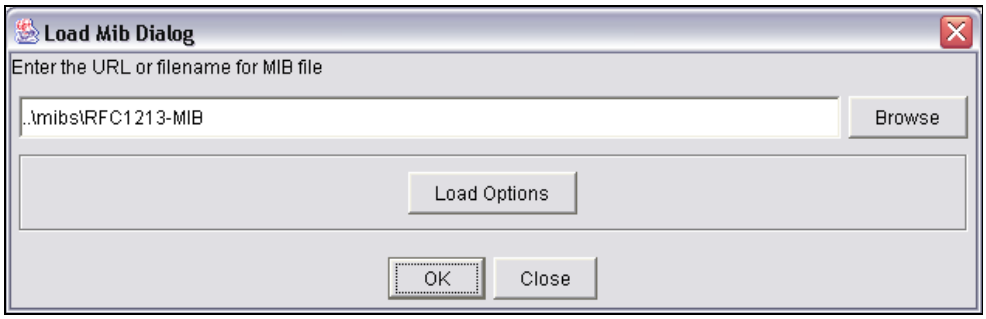
q. Icon  digunakan untuk menghentikan proses.

r. Icon  digunakan untuk menampilkan informasi tentang aplikasi.

s. Icon  digunakan untuk keluar dari aplikasi.

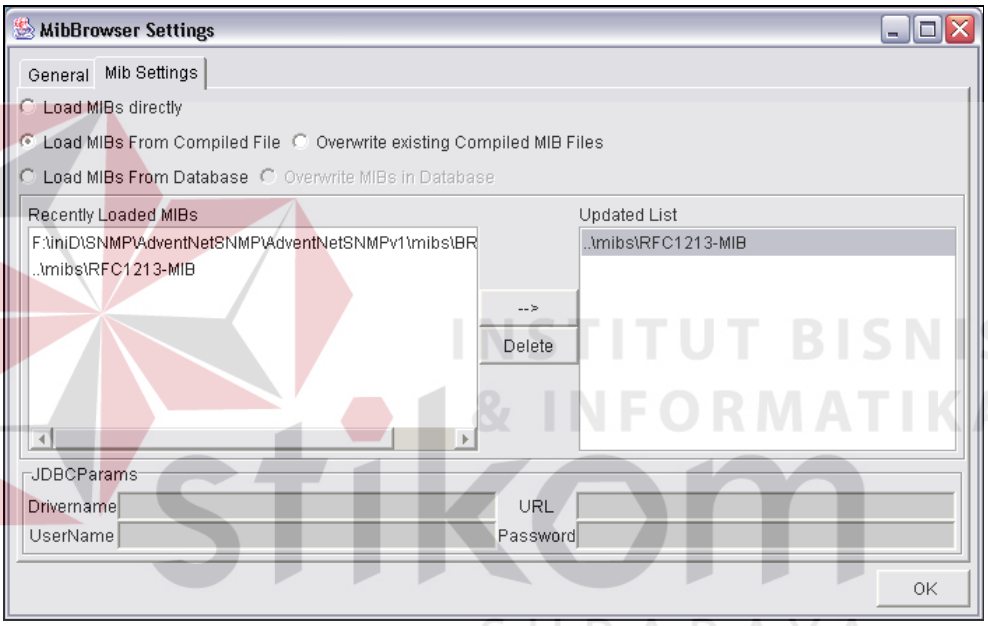
B. Dialog Pengambilan File MIB

Untuk menampilkan infomasi MIB, diperlukan satu file yang mendefinisikan spesifik dari MIB. Aplikasi ini menyediakan bagian untuk memudahkan pengambilan file MIB dimana digambarkan dalam bentuk sebagai berikut.



Gambar 4.2. Load MIB Dialog

Browse digunakan untuk mengarahkan ke direktori mana file yang akan digunakan, pada contoh ini file MIB diambil pada direktori *c:\AdventNetSnmpV3\mibs* sedangkan *Load Option* akan mengarahkan ke pengaturan MIB. Tombol *OK* akan memberikan jawaban bahwa file tersebut akan digunakan.



Gambar 4.3. Setting MIB

Gambar 4.3 memperlihatkan setting dari MIB dimana file-file yang di load dapat di ambil berdasarkan dari file sistem, file hasil kompilasi atau dari database. Kondisi awal pengaturan berada pada *Load MIBs From Compiled File* dan kondisi ini bisa dirubah sesuai kebutuhan yang diperlukan oleh user, sebagai contoh pilihan *Load MIBs From Database*, kondisi ini akan mengaktifkan

pengisian pada blok bagian paling bawah yaitu kelompok *JDBC Params*. User akan disediakan isian mulai dari *Driver Name, User Name, URL* dan *Password*.

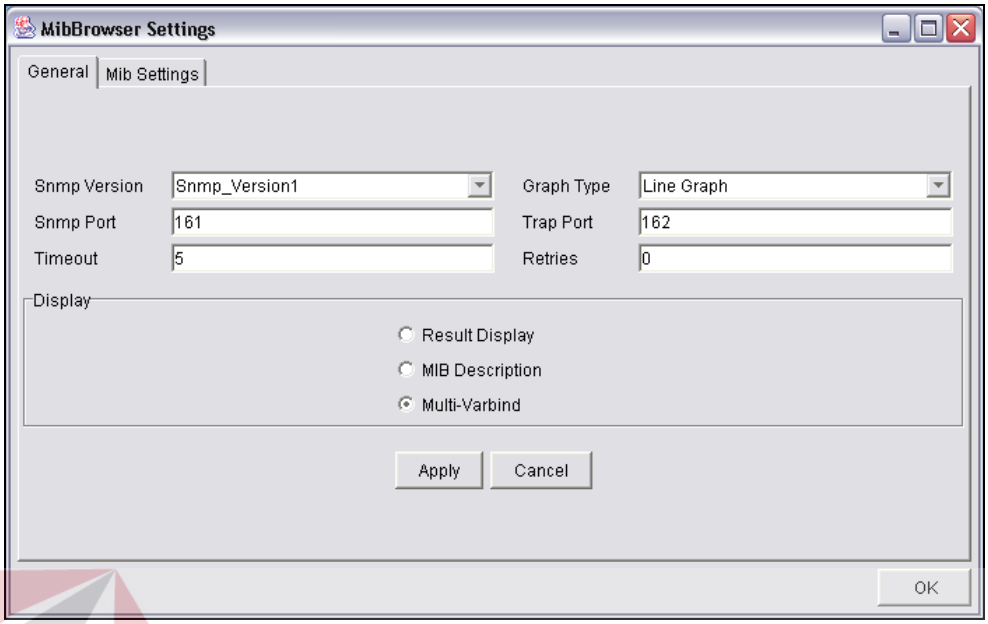
Kelompok *Recently Loaded MIBs* merupakan kumpulan file MIB yang sudah di load dalam sistem aplikasi, sedangkan kelompok *Update List* merupakan kumpulan file MIB yang akan di perbaruhi dalam pengambilan data.

C. Pengaturan MibBrowser

Untuk menampilkan dialog pengaturan (diperlihatkan pada *gambar 4.4*) bisa melalui icon pengaturan. Pada pengaturan ini akan ditampilkan kondisi awal dari data yang diperlukan aplikasi. Berikut daftar kondisi awal dari dialog pengaturan tersebut.

Tabel 4.11. Daftar kondisi awal pengaturan

PILIHAN	KONDISI AWAL	PILIHAN LAIN
Snmp Version	V1	V2c atau V3
Snmp Port	161	Bebas
Time out	5 detik	Bebas
Max Repetitions	50	Bebas
Graph Type	Line Graph	Bar Chart
Trap Port	162	Bebas
Retries	0	Bebas
Non-Repeaters	0	Bebas

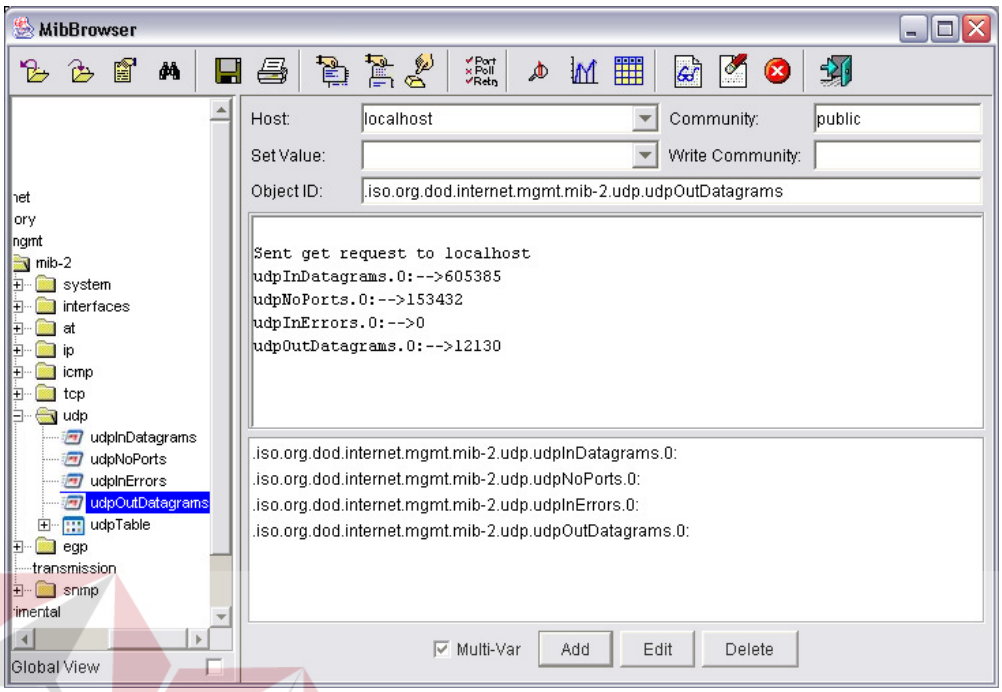


Gambar 4.4. Settings

D. Operasi SNMP

Aplikasi mendukung operasi dasar *GET, GETNEXT, GETBULK* dan *SET*. Pada kondisi file MIB sudah di load, gunakan icon 'get' untuk mendapatkan data berdasarkan node yang dipilih. Untuk mendapatkan data yang lengkap dari agent, dibutuhkan *ObjectID* dan *instant*. Melalui MIB, kita dapatkan ObjectID yang diperlukan.

Pada *gambar 4.5* dalam kelompok *Display* diperlihatkan 3 pilihat yang salah satunya adalah *Multi-Varbind*. Multi-Varbind ini berfungsi untuk memberikan fasilitas kepada user untuk mendaftarkan setiap titik node ke dalam daftar (*list*) dengan nilai yang bisa diisi pada bagian *Set Value*.



Gambar 4.5. Multi-Varbind

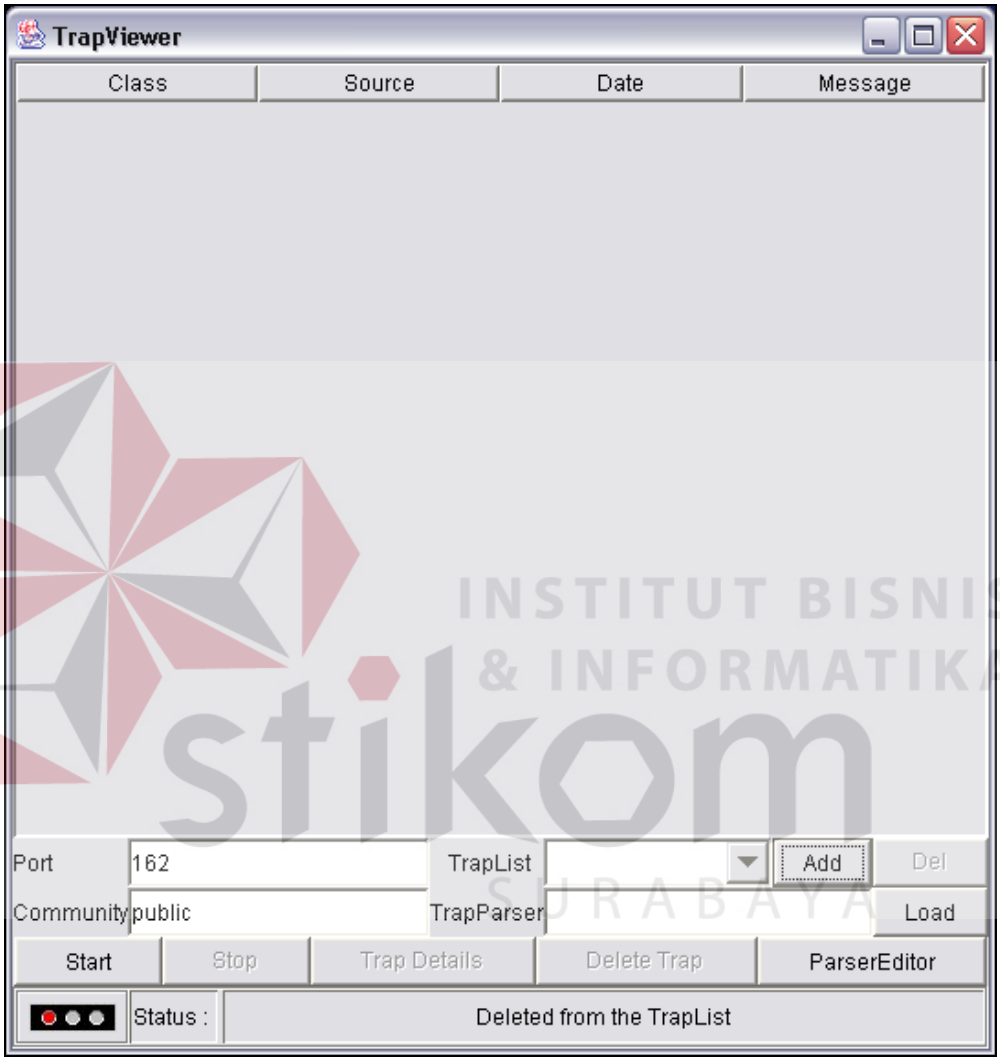
Tombol *Add* pada *gambar 4.5* berfungsi untuk melakukan proses *SET* dengan memilih node (*ObjectID*) pada *MIB Tree* dan nilai pada *Set Value*. Jika bagian *Set Value* tidak di isi, maka nilai *NULL* akan diberikan pada variabel. Pada contoh dalam gambar diatas diperlihatkan nilai yang diberikan adalah 34.

Untuk menghapus *varbind* dari daftar, bisa dilakukan dengan memilih salah satu *varbind* dan menekan tombol *Delete*. Sedangkan untuk merubah *varbind* dalam daftar, bisa dilakukan dengan menekan tombol *Edit*.

E. Trap Viewer dan Trap Parser Editor

Trap Viewer digunakan untuk menerima *traps*. Menggunakan *Trap Viewer* user dapat melihat trap yang masuk pada port yang ditentukan. Trap bisa dikirim melalui host mana saja. Nomer port (*Port Number*) dan nama komuniti

(*Community Name*) bisa diisi melalui inputan pada dialog *Trap Viewer*. *Gambar 4.6* memperlihatkan dialog *Trap Viewer*.



Gambar 4.6. Trap Viewer

Komponen – komponen dari *Trap Viewer* adalah sebagai berikut.

a. *Trap Table*, menampung seluruh trap yang diterima.

b. *Port*, menentukan port berapa yang digunakan untuk menerima trap.

c. *Community*, menentukan spesifikasi komuniti dalam menerima trap.

d. *Trap List*, adalah daftar yang berisikan port dan community.

e. *Trap Parser.*

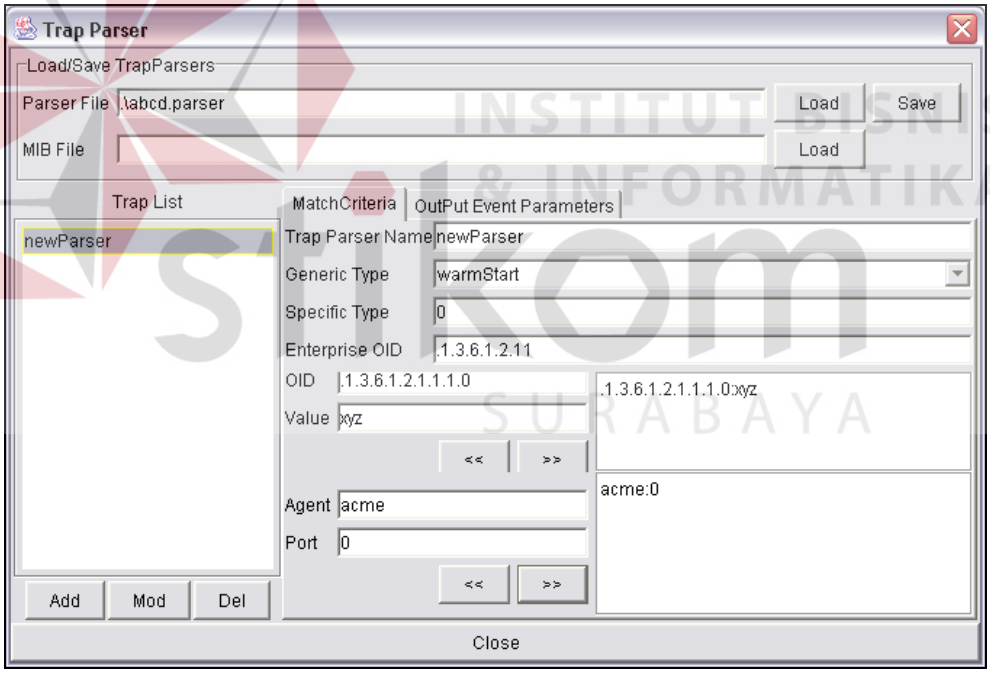
f. Tombol *Start* dan *Stop.*

g. Tombol *Trap Detail*, menampilkan detail trap.

h. Tombol *Delete Trap*, menghapus trap listener.

i. Tombol *Parser Editor*, menampilkan dialog *trap parser.*

Trap Parser, dapat memfilter setiap trap yang diterima dimana trap yang cocok akan disesuaikan dengan kriteria dari trap yang di definisikan pada *Parser Criteria.* Pada *gambar 4.7* memperlihatkan dialog tatap muka dari *Parser Editor*.



Gambar 4.7. Trap Parser

Trap Parser Editor merupakan *UI Tool* untuk menghasilkan *file trap parser*. *Trap Parser Editor* digunakan untuk konfigurasi dan untuk menguraikan

(*parse*) kejadian. Trap terkadang mengandung informasi yang kurang jelas sehingga mempersulit user untuk mendefinisikan, dan dengan menggunakan *Trap Parser* akan memberikan kemudahan kepada user untuk mendefinisikan informasi.

Kriteria yang benar adalah sangat dibutuhkan terhadap kebutuhan setiap trap yang diterima untuk di tampung ke dalam *Trap Table*.

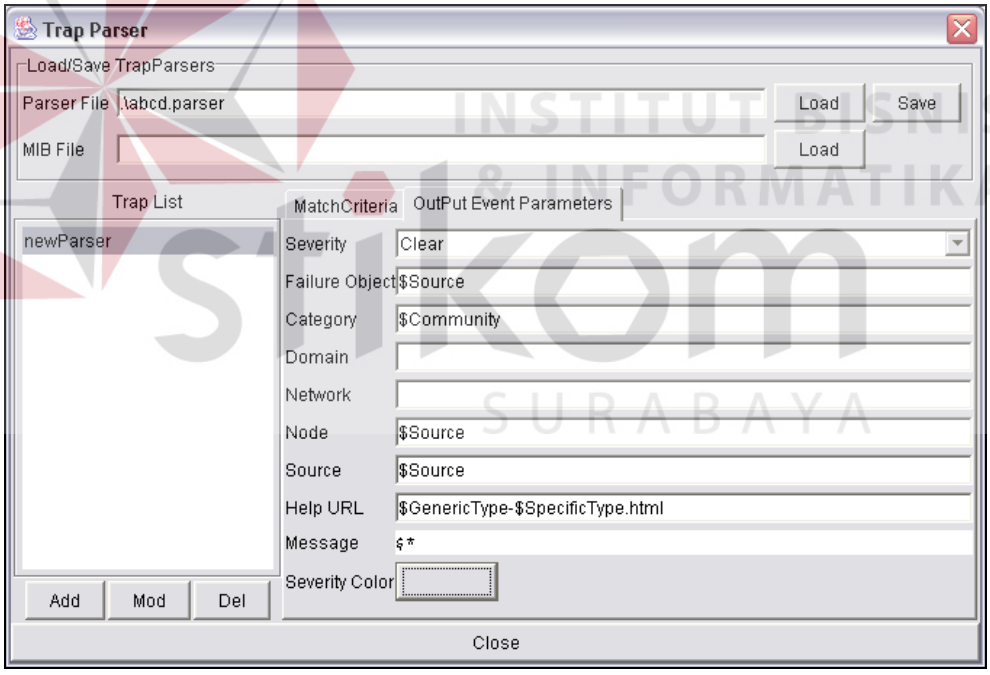
Suatu nilai dapat ditambahkan pada kriteria yang diinginkan ke dalam satu file dengan nama yang berbeda, sehingga *Trap Viewer* akan mencocokkan kriteria tersebuh secara beraturan (*sequence*). Jika trap sama/cocok maka akan di tampung ke dalam *Trap Table*.

Selama trap dalam kondisi siap, hanya ada satu *file parser* yang di *load* oleh *Trap Viewer*. Dan selama itu, *Trap Viewer* akan mencocokkan semua kriteria yang terpasang pada *file parser* hingga trap di matikan. Berikut adalah kriteria-kriteria yang disiapkan.

a. *Generic Type* : setiap trap memiliki nomer *generic type*. Nomor ini harus unik untuk *trap parser*. Nilai yang disediakan adalah *ColdStart, WarmStart, Linkdown, Linkup, Authentication Failure, egpNeighbourloss, enterprise Specific.*
b. *Spesific Type* : bagian ini berisi nilai mulai dari 0 – 64k. Bagian ini di khususkan untuk *generic type* yang bertipe *enterprise Specific*.
c. *Enterprise OID* : bagian ini berisikan *SNMP enterprise identifier* pada sebuah trap, yang digunakan untuk identiti unik dari aplikasi.
d. *OID and Value* : bagian ini harus sesuai untuk seluruh *OID:Value* dalam *trap PDU*.

e. *Agent and Port* : bagian ini merupakan tambahan untuk menyesuaikan terhadap kriteria yang ada dimana trap harus dikirimkan oleh *Agent* pada port tertentu (*Agent:Port*). Jika port berisi nilai 0, maka aplikasi bisa mengirim trap melalui port apa saja.

Setiap *Match Criteria* memiliki *Output Event Parameter* yang mendefinisikan kejadian apa saja yang akan di tangkap oleh *Trap Table*. *Output Event Parameter* ini akan ditampilkan pada *Trap Details* yang memberikan data lengkap tentang informasi yang ditangkap. *Gambar 4.8* memperlihatkan bagian dari parameter *Output Event Parameter*.



Gambar 4.8. Trap Parser

Kondisi awal dari setiap parameter *Event Event Parameters* (dalam hal ini disebut dengan *parser variable*) biasanya diawali dengan karakter "**$**". Variabel

tersebut mendefinisikan karakteristik dari *parser* dalam *Trap Details*. Berikut adalah *variabel parser* yang digunakan.

a. *$Community* : Tanda ini (*token*) akan diisi dengan komuniti *String*.

b. *$Source* : Tanda ini (*token*) akan diisi dengan nama/alamat.

c. *$Enterprise* : Tanda ini (*token*) akan diisi dengan *enterprise id*.

d. *$Agent* : Tanda ini (*token*) akan diisi dengan alamat *Agent*.

e. *$SpecificType* : Tanda ini (*token*) akan diisi dengan tipe spesifik dari trap yang diterima.

f. *$GenericType* : Tanda ini (*token*) akan diisi dengan tipe generik dari trap yang diterima.

g. *$Uptime* : Tanda ini (*token*) akan diisi dengan nilai *uptime*.

h. *$** : Tanda ini (*token*) akan diisi dengan gabungan variabel OID dan nilai variabel dari setiap variabel yang digabungkan.

i. *$#* : Tanda ini (*token*) akan diisi dengan seluruh nilai variabel SNMP pada variabel gabungan dari trap yang diterima.

j. *$N* : Tanda ini (*token*) akan diisi dengan nilai (N-1)th SNMP pada variabel gabungan dari trap yang diterima.

k. @* : Tanda ini (*token*) akan diisi dengan seluruh nilai OID pada variabel gabungan dari trap yang diterima.

l. *@N* : Tanda ini (*token*) akan diisi dengan nilai (N-1)th OID pada variabel gabungan dari trap yang diterima.

Berikut adalah aturan dalam membentuk sebuah *file parser*.

a. Jalankan *Trap Parser Editor* melalui *Trap Viewer*.

b. Isi group *Match Criteria* yang terdiri dari *Generic Type, Spesific Type, Enterprise OID, OID* dan *Value Pair (bisa dikosongkan, pilihan)* dan *Port Pair (bisa dikosongkan, pilihan).*

c. Biarkan nilai pada group *Output Event Parameters* dengan nilai standar

- *Severity* : "-"
- *Category* : "$Source"
- *Node* : "$Source"
- *Source* : "$Source"
- *HelpURL* : "$GenericType-$SpesificType.html"
- *Message* : "$*"

d. Nilai yang baru akan diberikan pada dialog *Trap Details* terhadap *field* yang berhubungan.

e. Isikan nama trap.

f. Tambahkan *Trap Parser* ke dalam *Trap List*.

g. Ulang mulai dari point pertama untuk menambahkan lebih dari satu *Match Creteria*.

h. Simpan *parser* yang sudah dibuat ke dalam *parser file* dengan menggunakan tombol *Save*.

i. Setelah di simpan, *parser file* akan ditampilkan pada bagian *Parser File text field.*

j. Tutup *Trap Parser Editor* menggunakan tombol *Close*.

k. Dan sekarang, *parser file* telah selesai dibentuk.

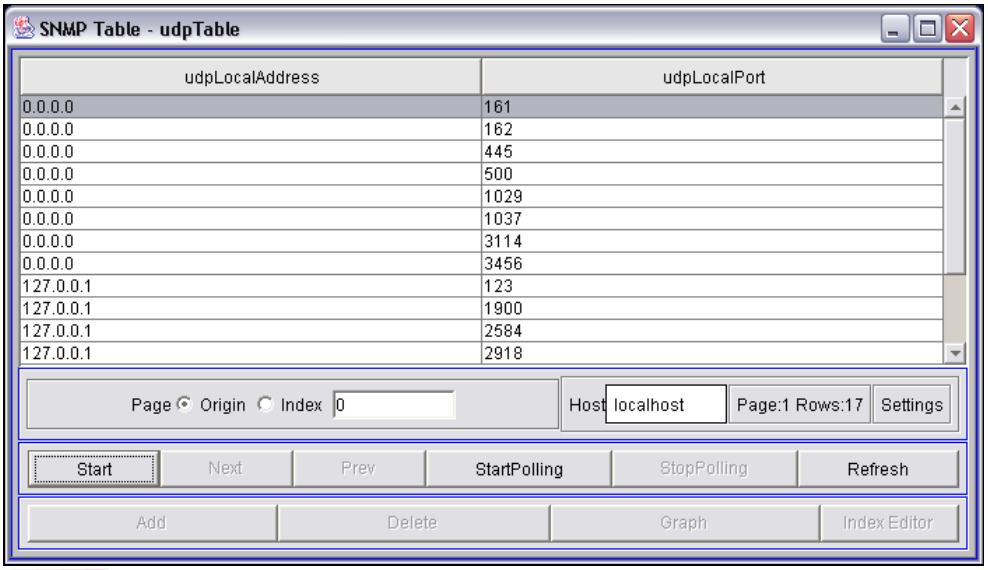
Untuk menambahkan *Trap* dari standar MIB, maka dilakukan proses sebagai berikut.

a. Pilih tombol *Load* pada bagian *MIB File* dan pilih file MIB yang akan dibuat *parser-nya*.
b. Setelah file MIB di load, secara otomatis variabel pengisian akan terisi sesuai dengan data yang ada dalam *MIB File* yang ditampilkan pada bagian *Match Criteria* dan *Output Event Parameter*.

F. Table Operations

Aplikasi mendukung *SNMP Table* dimana operasi ini dapat menampilkan lebih dari satu dialog yang kita disebut *SNMP Table Panel*. *SNMP Table Panel* menyediakan beberapa fungsi seperti penambahan baris pada table yang sudah ada, penampilan grafik, dan sebagainya. Langkah-langkah berikut merupakan tahapan untuk menampilkan dialog *SNMP Table*.

a. Tentukan nama atau *IP Address* dari *Agent*.
b. Pilih OID yang benar, user bisa mengambil melalu *SNMP Tree,* sebagai contoh OID : .iso.org.dod.internet.mgmt.mib-2.udp.udpTable
c. Pilih icon *View SNMP Table* atau melalui menu *View* → *SNMP Table*.
d. Dialog *SNMP Table* diperlihatkan pada *Gambar 4.9.*
e. Untuk menjalankan, tekan tombol *Start*.



Gambar 4.9 SNMP Table

Pilihan yang disediakan pada dialog *SNMP Table* ini adalah

a. *Page* : Bagian ini ada 2 pilihan, *origin* atau *index*. Jika pilihan pada *origin*, data akan diambil dari *Agent* apa adanya, sedangkan *index*, user dapat memilih nilai index dengan sistem pendekatan dari penerimaan data di tabel.

b. *Host* : nama host harus diisikan untuk menentukan dari mana data akan diambil, pada contoh ini diisi dengan "*localhost*".

c. *Settings* : bagian ini berguna untuk tambahan pilihan untuk mendukung penerimaan data, seting tambahan tersebut adalah

- *Polling Interva* : nilai interval yang diperlukan untuk mengambil data.
- *Page Size (rows)* : jumlah baris yang ditampilkan dalam setiap layar.
- *No of ColumnView* : jumlah kolom yang ditampilkan dalam setiap layar, nilai awalnya adalah 5.
- *Port No* : nomer port yang digunakan untuk menerima.
- ***Start*** : tombol *start* untuk memulai proses.

- *Next, Prev* : tombol *next, prev* sebagai navigasi dalam menampilkan sejumlah data pada satu layar.
- *StartPolling* : digunakan untuk memulai proses pengambilan data secara simultan pada interval waktu yang ditentukan.
- *EndPolling* : digunakan untuk menghentikan proses yang diawali dengan tombol *StartPolling*.
- *Refresh* : digunakan untuk merefresh tabel secara periodik. Ini berguna jika hasil proses ingin diperbaharui.
- *Add* : digunakan untuk menambahkan baris pada tabel. Akan muncul dialog baru untuk mengisikan data-data yang diperlukan untuk ditambahkan ke baris tabel.
- *Delete* : digunakan untuk menghapus baris dalam tabel.
- *Graph* : digunakan untuk menampilkan grafik.
- *Index Editor* : digunakan untuk mengedit dan mengeset nilai index dalam tabel.

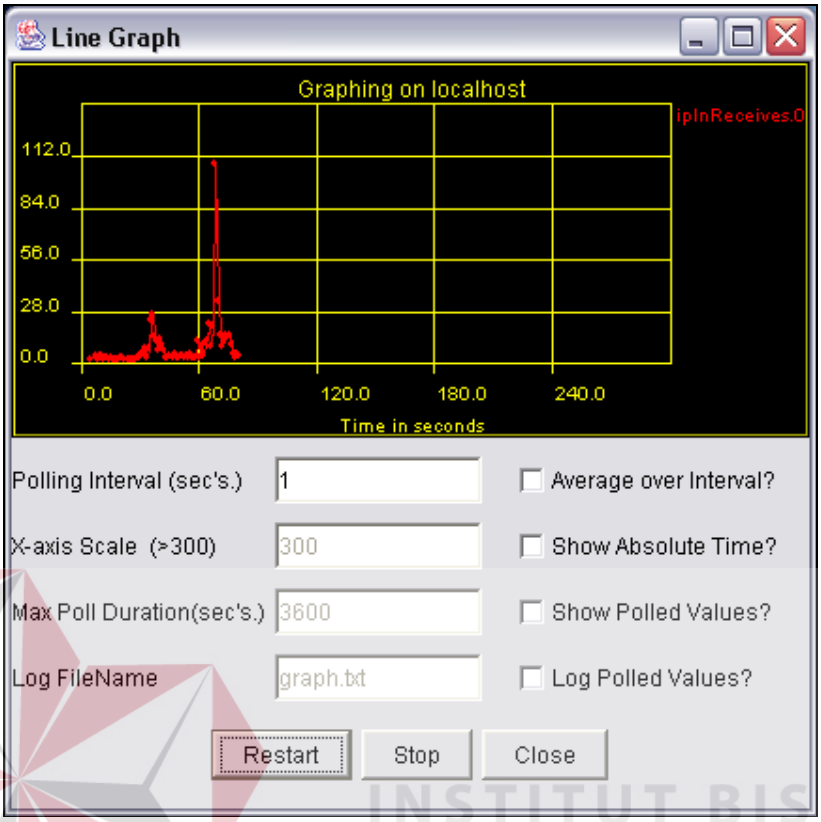
Terlepas dari pilihan diatas, jika di *klik kanan* pada judul tabel, akan muncul menu yang terdiri dari:

a. *Edit the header name of the selected colomn*
b. *View Graph for selected cells*
c. *Add a new row to the table*
d. *Delete the selected rows from the table*
e. *View the non-accessible index*

G. Grafik (Graphs)

Aplikasi mendukung fungsi untuk menampilkan data secara realtime dalam bentuk grafik, baik grafik garis atau grafik bar. Data SNMP akan dirubah ke dalam tipe Integer atau Unsigned Integer untuk mereprestasikan ke dalam grafik. Sederhananya, nilai yang diberikan akan bertipe *Counter*, *Gauge* atau *Timeticks*. Urutan untuk menampilkan dialog grafik adalah sebagai berikut:

a. Tentukan nama atau IP Address sebagai host yang diperlukan untuk sumber pengambilan data.
b. *Load* file MIB ke dalam aplikasi.
c. Berikan spesifik variabel yang diperlukan (bisa diambil melalui *MIB Tree*). Contoh: *.iso.org.dod.internet.mgmt.mib-2.ip.ipInReceives.0*
d. Pilih icon *Graph* untuk menampilkan grafik, diperlihatkan pada *gambar 4.10*.



Gambar 4.10. Line Graph

Berikut pilihan yang terdapat pada *Line Graph*.

a. *Polling Interval* : kondisi awal bernilai 5 detik (*sec's*), digunakan untuk interval waktu.

b. *X-axis Scale* : nilai minimum 300 detik (*sec's*), memungkinkan untuk menentukan skala *X-axis*.

c. *Max Poll Duration* : kondisi awal bernilai 3600 detik (*sec's*), digunakan untuk menentukan waktu maksimum yang dapat ditampung.

d. *Log File Name* : digunakan untuk menyimpan data ke dalam file .txt.

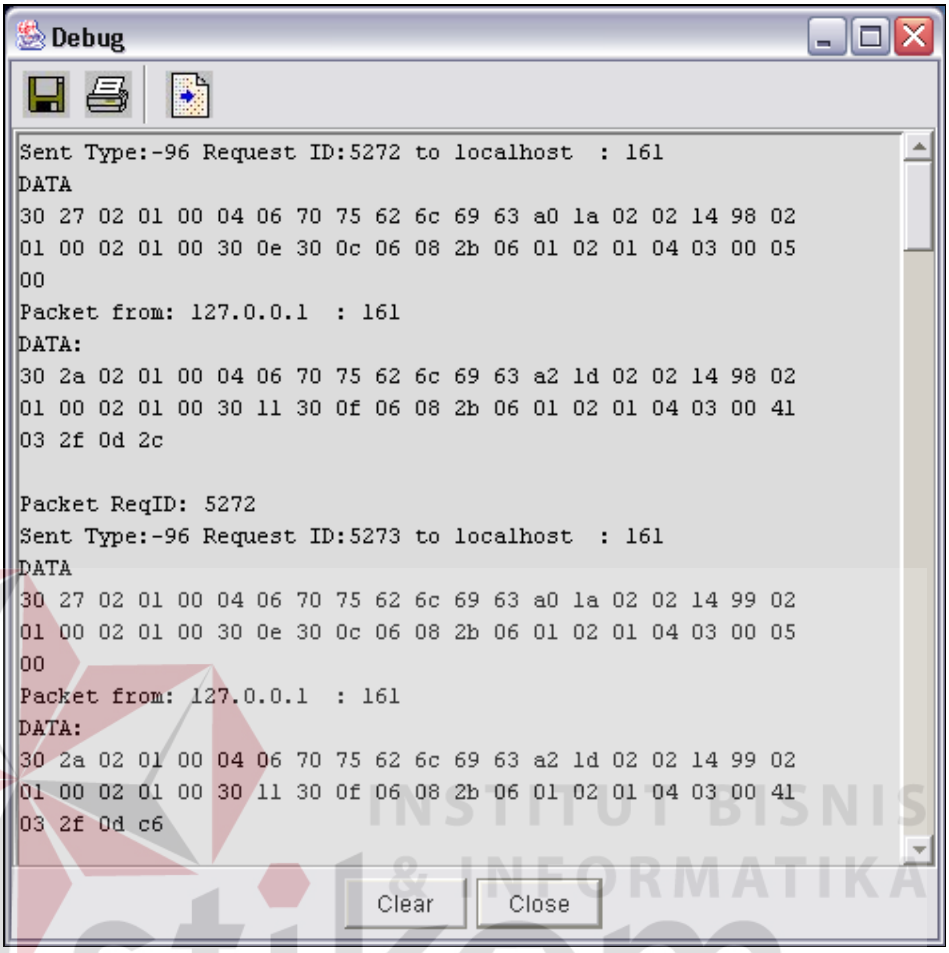
   - *Average over Interval* : jika dipilih, akan dihasilkan nilai rata-rata dari nilai sebenarnya.

- *Show Absolute Time* : jika dipilih, akan memiliki kondisi *hrs:secs (jam:detik)*
- *Show Polled Value* : jika dipilih, memungkinkan untuk menampilkan seluruh data dalam periode waktu (ketelitian).
- *Log Polled Values* : jika dipilih, akan mencatat hasil data ke dalam file .txt.

Tombol *Stop* yang disediakan, berfungsi untuk menghentikan proses, tombol *Restart* berfunsi untuk memulai ulang perhitungan berdasarkan data variabel yang diberikan, dan tombol *Close* untuk keluar dari dialog grafik.

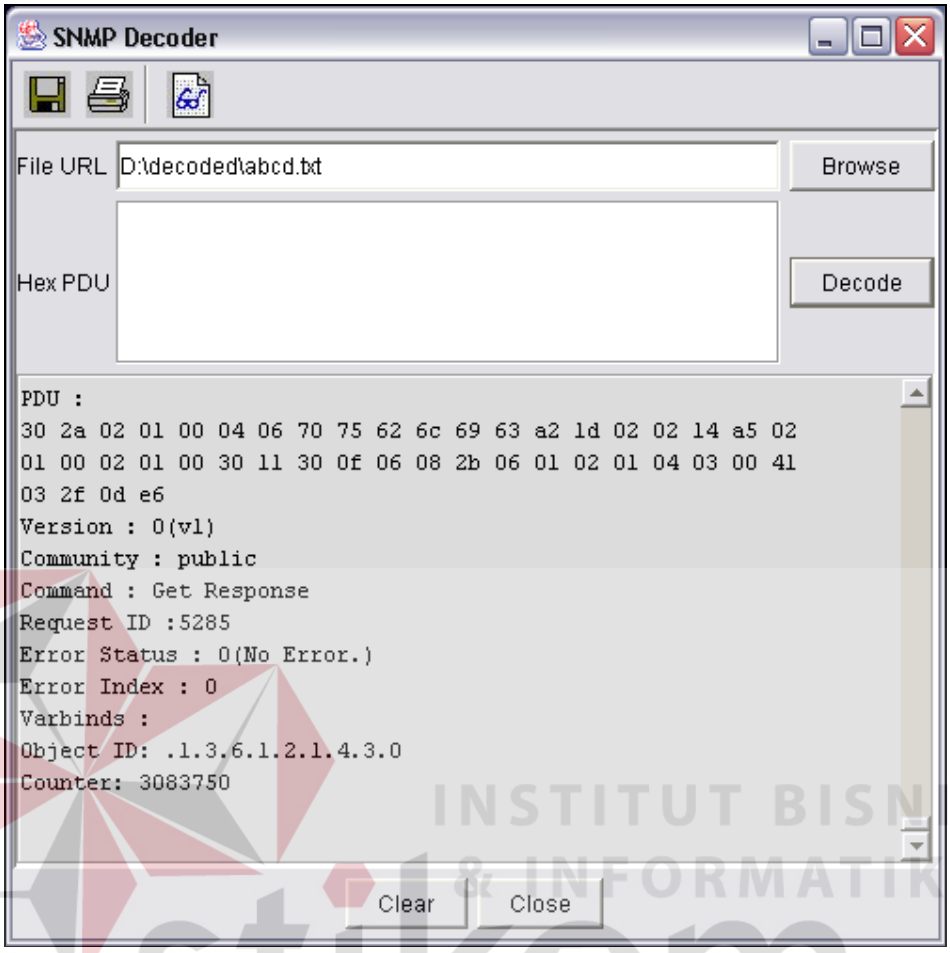
H. Debug dan Decoding

Aplikasi juga mendukung fasilitas untuk *debug* dan *decode output*. Dialog dapat ditampilkan melalui icon *debug*. Selama dialog ini aktif, proses debug akan terus aktif untuk setiap proses yang ada. Gambar 4.11 memperlihatkan dialog debug.



Gambar 4.11. Debug

Tombol-tombol yang disediakan diatas terdiri dari *save, print, decode, clear* dan *close*. *Decode* ini digunakan untuk melakukan pengembalian kode dari *SNMP Debug*. Gambar 4.12 memperlihatkan proses *decoded* dari file abcd.txt yang ada.



Gambar 4.12. Decoded

Prosed *decoded* ini bisa dilakukan dalam 2 cara yaitu:

Metode 1:

a. Klik icon *decoder*.

b. Copy informasi debug ke dalam area "*Hex PDU*".

c. Klik tombol decode.

d. Hasil decode akan ditampilkan pada bagian bawan dialog.

Metode 2:

a. Klik icon decoder.

b. Ambil file yang mengandung informasi debug, ini bisa didapat melalui copy informasi debug dan kita simpan ke dalam satu file, contoh file "*debug.txt*".
c. Klik tombol decode.
d. Hasil decode akan ditampilkan pada bagian bawan dialog.

4.3 Implementasi pada Jaringan

Dengan melakukan percobaan pada satu jaringan dimana sysAdmin akan melakukan proses request terhadap salah satu server dan melihat grafik trafik data yang masuk atau keluar pada server dengan IP Address 172.16.1.167.

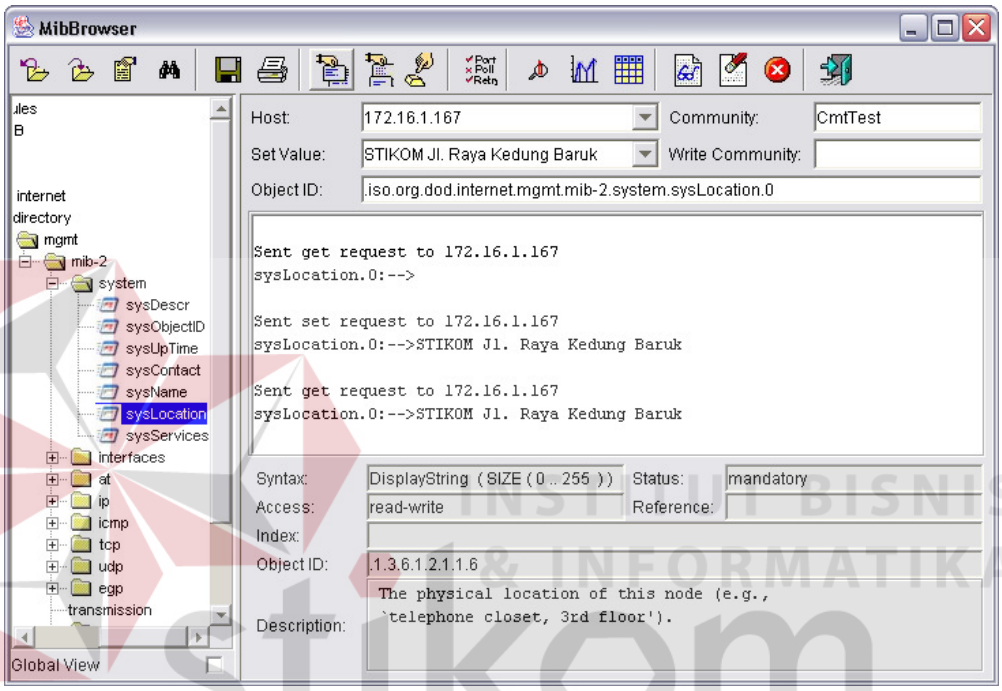
4.3.1 Request dan Update

Dalam keadaan aplikasi jalan dan MIB Tree sudah menampilkan object ID yang benar, maka aplikasi sudah siap melakukan request.

Dengan menggunakan tools menu yang tersedia, pilih terlebih dahulu objek pada MIB Tree yang ingin diambil datanya. Kemudian tentukan alamat tujuan yang akan menuju ke device seperti diperlihatkan pada gambar 4.1. Untuk *community* secara otomatis akan bernilai "public".

Jika pengaturan dasar ini sudah dipenuhi, maka proses request bisa dilakukan yaitu dengan memilih icon pada menu yang tersedia. Untuk melakukan proses "set", nilai yang bisa diberikan dapat diisikan pada inputan "Set Value". Pada percobaan ini, dilakukan proses "set" dengan nilai "STIKOM Jl. Raya Kedung Baruk" dan nama community yang digunakan adalah "CmtTest" ("CmtTest" di set pada Agent – *Control Panel* → *SNMP Service Properties*). Hasil percobaan bisa dilihat pada gambar 4.13. Pada layar diperlihatkan bahwa proses "get" yang pertama nilai dari object *sysLocation* adalah kosong, kemudian

pada proses berikutnya yaitu proses “set”, nilai diberikan ke object *sysLocation* adalah “STIKOM Jl. Raya Kedung Baruk”. Dan terakhir dilakukan proses “get” dimana nilai yang didapatkan sudah tidak lagi kosong tapi sama dengan nilai yang sudah diberikan sebelumnya.



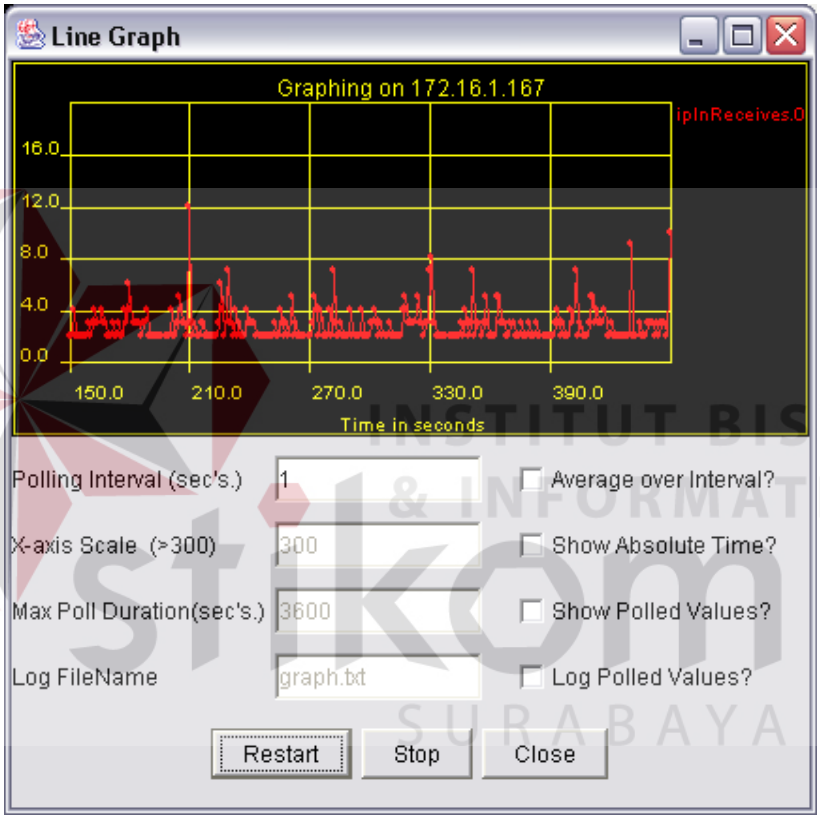
Gambar 4.13. Hasil Uji Coba Proses “get/set”

4.3.2 Menampilkan grafik

Mengambil data yang kemudian ditampilkan ke dalam grafik merupakan bagian penting dalam aplikasi ini, fungsinya lebih memberikan kemudahan dalam membaca informasi yang diambil sehingga mudah dalam menganalisa atau mengontrol.

Untuk itu telah disediakan fungsi untuk menampilkan dialog grafik seperti yang terlihat pada gambar 4.10.

Dengan memilih salah satu object pada MIB Tree, selanjutnya memilih icon untuk menampilkan grafik. Pada dialog tersebut terdapat isian yang dapat disesuaikan dengan kebutuhan hingga dapat menghasilkan file log dalam format txt. Pada percobaan ini dilakukan terhadap IP Address 172.16.1.167 dengan object *ipInReceive* dan hasil ditunjukan pada gambar 4.14.



Gambar 4.14. Hasil Uji Coba Grafik