How to migrate from AGENT++ 4.2.x (or older) to 4.3.0 (or newer) to be able to use multiple agents in a single process?

In AGENT++ prior to 4.3.0, there were many MibEntry relations and the usage of the central Mib instance organised by static class members called "instance". Although this reduces code size and makes initialisation routines easier it makes it impossible to run more than one Mib instance in a process.

In AGENT++ 4.3.0 this has been refactored by lot of internal changes, which had two main goals:

1. Remove any Mib::instance usages that are not optional to the API user (i.e. backward compatibility can/should be provided)
2. Remove any v3MP::I usages that are not optional to the API user.

If you do not want to use the multi-agent support, you do not have to change anything in your code. In case, you want to clean-up the API usage and/or want to use the new features, please follow the steps below in your agent application:

AGENT++ agent intialisation (+new/-old)

```
        Snmpx snmp(status, inaddr);
-        mib = new Mib(persistentObjectsPath);
+        mib = new Mib(persistentObjectsPath, path(bootCounterFile));
-        reqList = new RequestList();
+        reqList = new RequestList(mib);
        reqList->set_snmp(&snmp);
        mib->set_request_list(reqList);

    v3MP *v3mp = new v3MP(engineId, snmpEngineBoots, stat);
+   snmp.set_mpv3(v3mp);

-        mib.add(new snmp_community_mib());
+        mib.add(new snmp_community_mib(&mib));

-        mib.add(new agentpp_config_mib());
+        mib.add(new agentpp_config_mib(&mib));

-        mib.add(new notification_log_mib());
+        mib.add(new notification_log_mib(&mib));

+   snmpCommunityEntry* communityEntry = snmpCommunityEntry::get_instance(mib);
+   if (communityEntry) {
            OctetStr co("public");
-                MibTableRow* row = snmpCommunityEntry::instance->add_row(Oidx::from_string(co, FALSE));
+           MibTableRow* row = communityEntry->add_row(Oidx::from_string(co, FALSE));
        OctetStr tag("v1v2cPermittedManagers");
-                snmpCommunityEntry::instance->set_row(row, co, co,
+       communityEntry->set_row(row, co, co, reqList->get_v3mp()->get_local_engine_id(),"", tag, 3, 1);
+   }

-        UsmUserTable *uut = new UsmUserTable();
+   v3MP* v3mp = mib.get_request_list()->get_v3mp();
+        UsmUserTable *uut = new UsmUserTable(v3mp);
         // add non persistent USM statistics
-        mib.add(new UsmStats());
+        mib.add(new UsmStats(v3mp));
         // add the USM MIB - usm_mib MibGroup is used to
         // make user added entries persistent
         mib.add(new usm_mib(uut));
         // add non persistent SNMPv3 engine object
-        mib.add(new V3SnmpEngine());
-        mib.add(new MPDGroup());
+        mib.add(new V3SnmpEngine(v3mp));
+        mib.add(new MPDGroup(v3mp));
```

A complete sample agent C++ code with multiple full featured agents in one process is show below.

It provides per **SNMP agent port** the following features:

- USM
- MPv3
- Request queue and processing
- MIB counters
- Independent MIB modules including persistency
- SNMPv3 context support
- An agent may contain the same or different MIB modules/objects

Multi-Agent Sample Code

```
/*_############################################################################
  _##
  _##  AGENT++ 4.0 - agent.cpp
  _##
  _##  Copyright (C) 2000-2020  Frank Fock and Jochen Katz (agentpp.com)
  _##
  _##  Licensed under the Apache License, Version 2.0 (the "License");
  _##  you may not use this file except in compliance with the License.
  _##  You may obtain a copy of the License at
  _##
  _##      http://www.apache.org/licenses/LICENSE-2.0
  _##
  _##  Unless required by applicable law or agreed to in writing, software
  _##  distributed under the License is distributed on an "AS IS" BASIS,
  _##  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  _##  See the License for the specific language governing permissions and
  _##  limitations under the License.
  _##
  _############################################################################*/

#include <stdlib.h>
#include <signal.h>

#include <agent_pp/agent++.h>
#include <agent_pp/snmp_group.h>
#include <agent_pp/system_group.h>
#include <agent_pp/snmp_target_mib.h>
#include <agent_pp/snmp_notification_mib.h>
#include <agent_pp/snmp_community_mib.h>
#include <agent_pp/notification_originator.h>
#include <agent_pp/notification_log_mib.h>
#include <agent_pp/agentpp_simulation_mib.h>
#include <agent_pp/agentpp_config_mib.h>
#include <agent_pp/v3_mib.h>
#include <agent_pp/mib_policy.h>
#include <agent_pp/vacm.h>
#include <agent_pp/threads.h>
#include <agent_pp/tools.h>

#include <snmp_pp/oid_def.h>
#include <snmp_pp/mp_v3.h>
#include <snmp_pp/log.h>

#include "atm_mib.h"
#include "agentpp_notifytest_mib.h"
#include "agentpp_test_mib.h"


#ifdef SNMP_PP_NAMESPACE
using namespace Snmp_pp;
#endif

#ifdef AGENTPP_NAMESPACE
using namespace Agentpp;
#endif
```

```
static const char* loggerModuleName = "agent++.multi.agent";

// table size policies


// globals:
const unsigned int MAX_NUMBER_OF_AGENTS = 10;

bool go = TRUE;

unsigned short port[MAX_NUMBER_OF_AGENTS];
#ifdef _SNMPv3
const table_size_def table_sizes[4] =
{ table_size_def(oidUsmUserEntry, 30),
  table_size_def(oidVacmSecurityToGroupEntry, 30),
  table_size_def(oidVacmAccessEntry, 30),
  table_size_def(oidVacmViewTreeFamilyEntry, 30)
};
MibTableSizePolicy policy(table_sizes, 4, 5000);
#endif


static void sig(int signo)
{
        if ((signo == SIGTERM) || (signo == SIGINT) ||
            (signo == SIGSEGV)) {

                printf ("\n");

                switch (signo) {
                case SIGSEGV: {
                        printf ("Segmentation fault, aborting.\n");
                        exit(1);
                }
                case SIGTERM:
                case SIGINT: {
                        if (go) {
                                go = FALSE;
                                printf ("User abort\n");
                        }
                }
                }
        }
}


void init_signals()
{
        signal (SIGTERM, sig);
        signal (SIGINT, sig);
        signal (SIGSEGV, sig);
}



void init(Mib& mib, const NS_SNMP OctetStr& engineID, const UdpAddress& inaddr)
{
        OctetStr sysDescr("AGENT++v");
        sysDescr += AGENTPP_VERSION_STRING;
        sysDescr += " ATM Simulation Agent (";
        sysDescr += inaddr.get_printable();
        sysDescr += ")";
        mib.add(new sysGroup(sysDescr.get_printable(),
                             "1.3.6.1.4.1.4976", 10));
         mib.add(new snmpGroup());
        mib.add(new TestAndIncr(oidSnmpSetSerialNo));
        mib.add(new atm_mib());
        mib.add(new agentpp_simulation_mib());
        mib.add(new agentpp_notifytest_mib(&mib));
        mib.add(new agentpp_test_mib());
        mib.add(new snmp_target_mib());
```

```
#ifdef _SNMPv3
        mib.add(new snmp_community_mib(&mib));
#endif
        mib.add(new snmp_notification_mib());
#ifdef _SNMPv3
#ifdef _NO_THREADS
        mib.add(new agentpp_config_mib(&mib));
#else
        mib.add(new agentpp_config_mib(&mib));
#endif
        mib.add(new notification_log_mib(&mib));

        OctetStr nonDefaultContext("other");
        mib.add(nonDefaultContext, new atm_mib());

        v3MP* v3mp = mib.get_request_list()->get_v3mp();
        UsmUserTable *uut = new UsmUserTable(v3mp);

        uut->addNewRow("unsecureUser",
                       SNMP_AUTHPROTOCOL_NONE,
                       SNMP_PRIVPROTOCOL_NONE, "", "", engineID, false);

        uut->addNewRow("MD5",
                       SNMP_AUTHPROTOCOL_HMACMD5,
                       SNMP_PRIVPROTOCOL_NONE,
                       "MD5UserAuthPassword", "",  engineID, false);

        uut->addNewRow("SHA",
                       SNMP_AUTHPROTOCOL_HMACSHA,
                       SNMP_PRIVPROTOCOL_NONE,
                       "SHAUserAuthPassword", "",  engineID, false);

        uut->addNewRow("MD5DES",
                       SNMP_AUTHPROTOCOL_HMACMD5,
                       SNMP_PRIVPROTOCOL_DES,
                       "MD5DESUserAuthPassword",
                       "MD5DESUserPrivPassword",  engineID, false);

        uut->addNewRow("SHADES",
                       SNMP_AUTHPROTOCOL_HMACSHA,
                       SNMP_PRIVPROTOCOL_DES,
                       "SHADESUserAuthPassword",
                       "SHADESUserPrivPassword",  engineID, false);

        uut->addNewRow("MD53DES",
                       SNMP_AUTHPROTOCOL_HMACMD5,
                       SNMP_PRIVPROTOCOL_3DESEDE,
                       "MD53DESUserAuthPassword",
                       "MD53DESUserPrivPassword",  engineID, false);

        uut->addNewRow("SHA3DES",
                       SNMP_AUTHPROTOCOL_HMACSHA,
                       SNMP_PRIVPROTOCOL_3DESEDE,
                       "SHA3DESUserAuthPassword",
                       "SHA3DESUserPrivPassword",  engineID, false);

        uut->addNewRow("MD5IDEA",
                       SNMP_AUTHPROTOCOL_HMACMD5,
                       SNMP_PRIVPROTOCOL_IDEA,
                       "MD5IDEAUserAuthPassword",
                       "MD5IDEAUserPrivPassword",  engineID, false);

        uut->addNewRow("SHAIDEA",
                       SNMP_AUTHPROTOCOL_HMACSHA,
                       SNMP_PRIVPROTOCOL_IDEA,
                       "SHAIDEAUserAuthPassword",
                       "SHAIDEAUserPrivPassword",  engineID, false);

        uut->addNewRow("MD5AES128",
                       SNMP_AUTHPROTOCOL_HMACMD5,
                       SNMP_PRIVPROTOCOL_AES128,
```

```
                        "MD5AES128UserAuthPassword",
                        "MD5AES128UserPrivPassword",  engineID, false);

        MibTableRow* r = uut->addNewRow("SHAAES128",
                        SNMP_AUTHPROTOCOL_HMACSHA,
                        SNMP_PRIVPROTOCOL_AES128,
                        "SHAAES128UserAuthPassword",
                        "SHAAES128UserPrivPassword",  engineID, false);
    if (r) { uut->set_storage_type(r, storageType_permanent); }

        uut->addNewRow("MD5AES192",
                        SNMP_AUTHPROTOCOL_HMACMD5,
                        SNMP_PRIVPROTOCOL_AES192,
                        "MD5AES192UserAuthPassword",
                        "MD5AES192UserPrivPassword",  engineID, false);

        uut->addNewRow("SHAAES192",
                        SNMP_AUTHPROTOCOL_HMACSHA,
                        SNMP_PRIVPROTOCOL_AES192,
                        "SHAAES192UserAuthPassword",
                        "SHAAES192UserPrivPassword", engineID, false);

        r = uut->addNewRow("MD5AES256",
                        SNMP_AUTHPROTOCOL_HMACMD5,
                        SNMP_PRIVPROTOCOL_AES256,
                        "MD5AES256UserAuthPassword",
                        "MD5AES256UserPrivPassword", engineID, false);
    if (r) { uut->set_storage_type(r, storageType_readOnly); }

        uut->addNewRow("SHAAES256",
                        SNMP_AUTHPROTOCOL_HMACSHA,
                        SNMP_PRIVPROTOCOL_AES256,
                        "SHAAES256UserAuthPassword",
                        "SHAAES256UserPrivPassword", engineID, false);

        uut->addNewRow("SHA512AES256",
                        SNMP_AUTHPROTOCOL_HMAC384SHA512,
                        SNMP_PRIVPROTOCOL_AES256,
                        "SHA512AES256UserAuthPassword",
                        "SHA512AES256UserPrivPassword", engineID, false);

        uut->addNewRow("SHA384AES128",
                        SNMP_AUTHPROTOCOL_HMAC256SHA384,
                        SNMP_PRIVPROTOCOL_AES256,
                        "SHA384AES128UserAuthPassword",
                        "SHA384AES128UserPrivPassword", engineID, false);

        uut->addNewRow("SHA256AES128",
                        SNMP_AUTHPROTOCOL_HMAC192SHA256,
                        SNMP_PRIVPROTOCOL_AES128,
                        "SHA256AES128UserAuthPassword",
                        "SHA256AES128UserPrivPassword", engineID, false);

        uut->addNewRow("SHA224AES128",
                        SNMP_AUTHPROTOCOL_HMAC128SHA224,
                        SNMP_PRIVPROTOCOL_AES128,
                        "SHA224AES128UserAuthPassword",
                        "SHA224AES128UserPrivPassword", engineID, false);

        // add non persistent USM statistics
        mib.add(new UsmStats(v3mp));
        // add the USM MIB - usm_mib MibGroup is used to
        // make user added entries persistent
        mib.add(new usm_mib(uut));
        // add non persistent SNMPv3 engine object
        mib.add(new V3SnmpEngine(v3mp));
        mib.add(new MPDGroup(v3mp));
#endif
}

class SnmpAgent: public Runnable {
```

```
public:
    SnmpAgent(const UdpAddress& address): Runnable() {
        inaddr = address;
    }
    virtual ~SnmpAgent() { }
    virtual void run();

protected:
    UdpAddress inaddr;
};

OctetStr& path(OctetStr& path)
{
    for (int i=0; i<path.len(); i++) {
        if (path[i] == '/') {
            path[i] = '_';
        }
    }
    return path;
}

void SnmpAgent::run() {
        Mib* mib;
        RequestList* reqList;
        int status;
        // bind localhost only -> agent will not be reachable from
        // outside
        // UdpAddress inaddr("127.0.0.1");
        Snmpx snmp(status, inaddr);

        if (status == SNMP_CLASS_SUCCESS) {

                LOG_BEGIN(loggerModuleName, EVENT_LOG | 1);
                LOG("main: SNMP listen address");
                LOG(inaddr.get_printable());
                LOG_END;
        }
        else {
                LOG_BEGIN(loggerModuleName, ERROR_LOG | 0);
                LOG("main: SNMP port init failed");
                LOG(status);
                LOG(Snmp::error_msg(status));
                LOG_END;
                exit(1);
        }

        OctetStr persistentObjectsPath(".config/");
        OctetStr bootCounterFile(".config_bc_");
        persistentObjectsPath += inaddr.get_printable();
        bootCounterFile += inaddr.get_printable();
        persistentObjectsPath += "/";
        // Make sure persistent objects path exists:
        if (!AgentTools::make_path(persistentObjectsPath.get_printable())) {
            LOG_BEGIN(loggerModuleName, WARNING_LOG | 1);
            LOG("Directory for storing persistent data could not be created (dir)");
            LOG(persistentObjectsPath.get_printable());
            LOG_END;
        }
        mib = new Mib(persistentObjectsPath, path(bootCounterFile));

#ifdef _SNMPv3
        unsigned int snmpEngineBoots = 0;
        OctetStr engineId(SnmpEngineID::create_engine_id(inaddr.get_port()));

        // you may use your own methods to load/store this counter
        status = mib->get_boot_counter(engineId, snmpEngineBoots);
        if ((status != SNMPv3_OK) && (status < SNMPv3_FILEOPEN_ERROR)) {
                LOG_BEGIN(loggerModuleName, ERROR_LOG | 0);
                LOG("main: Error loading snmpEngineBoots counter (status)");
                LOG(status);
```

```
                LOG_END;
                exit(1);
        }

        snmpEngineBoots++;
        status = mib->set_boot_counter(engineId, snmpEngineBoots);
        if (status != SNMPv3_OK) {
                LOG_BEGIN(loggerModuleName, ERROR_LOG | 0);
                LOG("main: Error saving snmpEngineBoots counter (status)");
                LOG(status);
                LOG_END;
                exit(1);
        }

        int stat;
        v3MP *v3mp = new v3MP(engineId, snmpEngineBoots, stat);
        snmp.set_mpv3(v3mp);
#endif
        reqList = new RequestList(mib);
#ifdef _SNMPv3
        // register v3MP
        reqList->set_v3mp(v3mp);
#endif
        // register requestList for outgoing requests
        mib->set_request_list(reqList);

        // add supported objects
        init(*mib, engineId, inaddr);

        reqList->set_snmp(&snmp);

#ifdef _SNMPv3
        // register VACM
        Vacm* vacm = new Vacm(*mib);
        reqList->set_vacm(vacm);

        // initialize security information
        vacm->addNewContext("");
        vacm->addNewContext("other");

        // Add new entries to the SecurityToGroupTable.
        // Used to determine the group a given SecurityName belongs to.
        // User "new" of the USM belongs to newGroup

        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "new",
                          "newGroup", storageType_nonVolatile);

        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "test",
                          "testGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_V2, "public",
                          "v1v2group", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_V1, "public",
                          "v1v2group", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "initial",
                          "initial", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "unsecureUser",
                          "newGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5",
                          "testNoPrivGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA",
                          "testNoPrivGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5DES",
                          "testGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHADES",
                          "testGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD53DES",
                          "testGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA3DES",
                          "testGroup", storageType_nonVolatile);
        vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5IDEA",
                          "testGroup", storageType_nonVolatile);
```

```
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHAIDEA",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5AES128",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHAAES128",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5AES192",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHAAES192",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "MD5AES256",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHAAES256",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA512AES256",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA384AES128",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA256AES128",
                  "testGroup", storageType_nonVolatile);
vacm->addNewGroup(SNMP_SECURITY_MODEL_USM, "SHA224AES128",
                  "testGroup", storageType_nonVolatile);

// remove a group with:
//vacm->deleteGroup(SNMP_SECURITY_MODEL_USM, "neu");

// Set access rights of groups.
// The group "newGroup" (when using the USM with a security
// level >= noAuthNoPriv within context "") would have full access
// (read, write, notify) to all objects in view "newView".
vacm->addNewAccessEntry("newGroup",
                        "other",        // context
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_exact,  // context must mach exactly
                        // alternatively: match_prefix
                        "newView", // readView
                        "newView", // writeView
                        "newView", // notifyView
                        storageType_nonVolatile);
vacm->addNewAccessEntry("testGroup", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_AUTH_PRIV,
                        match_prefix,
                        "testView", "testView",
                        "testView", storageType_nonVolatile);
vacm->addNewAccessEntry("testNoPrivGroup", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_AUTH_NOPRIV,
                        match_prefix,
                        "testView", "testView",
                        "testView", storageType_nonVolatile);
vacm->addNewAccessEntry("testNoPrivGroup", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_prefix,
                        "testView", "testView",
                        "testView", storageType_nonVolatile);
vacm->addNewAccessEntry("testGroup", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_prefix,
                        "testView", "testView",
                        "testView", storageType_nonVolatile);
vacm->addNewAccessEntry("v1v2group", "",
                        SNMP_SECURITY_MODEL_V2,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_exact,
                        "v1ReadView", "v1WriteView",
                        "v1NotifyView", storageType_nonVolatile);
vacm->addNewAccessEntry("v1v2group", "",
```

```
                        SNMP_SECURITY_MODEL_V1,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_exact,
                        "v1ReadView", "v1WriteView",
                        "v1NotifyView", storageType_nonVolatile);
vacm->addNewAccessEntry("initial", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV,
                        match_exact,
                        "restricted", "",
                        "restricted", storageType_nonVolatile);
vacm->addNewAccessEntry("initial", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_AUTH_NOPRIV,
                        match_exact,
                        "internet", "internet",
                        "internet", storageType_nonVolatile);
vacm->addNewAccessEntry("initial", "",
                        SNMP_SECURITY_MODEL_USM,
                        SNMP_SECURITY_LEVEL_AUTH_PRIV,
                        match_exact,
                        "internet", "internet",
                        "internet", storageType_nonVolatile);

// remove an AccessEntry with:
// vacm->deleteAccessEntry("newGroup",
//                              "",
//                        SNMP_SECURITY_MODEL_USM,
//                        SNMP_SECURITY_LEVEL_NOAUTH_NOPRIV);


// Defining Views
// View "v1ReadView" includes all objects starting with "1.3".
// If the ith bit of the mask is not set (0), then also all objects
// which have a different subid at position i are included in the
// view.
// For example: Oid "6.5.4.3.2.1", Mask(binary) 110111
//              Then all objects with Oid with "6.5.<?>.3.2.1"
//              are included in the view, whereas <?> may be any
//              natural number.

vacm->addNewView("v1ReadView",
                 "1.3",
                 "",             // Mask "" is same as 0xFFFFFFFFFF...
                 view_included,  // alternatively: view_excluded
                 storageType_nonVolatile);

vacm->addNewView("v1WriteView",
                 "1.3",
                 "",             // Mask "" is same as 0xFFFFFFFFFF...
                 view_included,  // alternatively: view_excluded
                 storageType_nonVolatile);

vacm->addNewView("v1NotifyView",
                 "1.3",
                 "",             // Mask "" is same as 0xFFFFFFFFFF...
                 view_included,  // alternatively: view_excluded
                 storageType_nonVolatile);

vacm->addNewView("newView", "1.3", "",
                 view_included, storageType_nonVolatile);
vacm->addNewView("testView", "1.3.6", "",
                 view_included, storageType_nonVolatile);
vacm->addNewView("internet", "1.3.6.1","",
                 view_included, storageType_nonVolatile);
vacm->addNewView("restricted", "1.3.6.1.2.1.1","",
                 view_included, storageType_nonVolatile);
vacm->addNewView("restricted", "1.3.6.1.2.1.11","",
                 view_included, storageType_nonVolatile);
vacm->addNewView("restricted", "1.3.6.1.6.3.10.2.1","",
                 view_included, storageType_nonVolatile);
```

```
        vacm->addNewView("restricted", "1.3.6.1.6.3.11.2.1","",
                         view_included, storageType_nonVolatile);
        vacm->addNewView("restricted", "1.3.6.1.6.3.15.1.1","",
                         view_included, storageType_nonVolatile);

        // add SNMPv1/v2c community to v3 security name mapping
        snmpCommunityEntry* communityEntry =
                snmpCommunityEntry::get_instance(mib);
        if (communityEntry) {
            OctetStr co("public");
            MibTableRow* row = communityEntry->add_row(Oidx::from_string(co, FALSE));
            OctetStr tag("v1v2cPermittedManagers");
            communityEntry->set_row(row, co, co,
                        reqList->get_v3mp()->get_local_engine_id(),
                        "", tag, 3, 1);
        }

#endif

#ifdef _SNMPv3
        // register table size policies
        MibTableSizePolicy::register_policy(mib->get_default_context(),
                                            &policy);
#endif
        // load persistent objects from disk
        mib->init();


        Vbx* vbs = 0;
        coldStartOid coldOid;
        NotificationOriginator no(mib);
        // add an example destination
        UdpAddress dest("127.0.0.1/162");
        no.add_v1_trap_destination(dest, "defaultV1Trap", "v1trap", "public");
        // send the notification
        mib->notify("", coldOid, vbs, 0);

        Request* req;
        while (go) {
                req = reqList->receive(2);
                if (req) {
                    mib->process_request(req);
                }
                else {
                    mib->cleanup();
                }
        }
        delete reqList;
        delete mib;
#ifdef _SNMPv3
        delete vacm;
        delete v3mp;
#endif
}

int main (int argc, char* argv[])
{
    int num_agents = 0;
    while (num_agents < MAX_NUMBER_OF_AGENTS && num_agents < argc - 1) {
         port[num_agents] = atoi(argv[1+num_agents]);
         num_agents++;
    }
        if (num_agents == 0) {
         // minimum of two agents if not specified explicitly:
         port[0] = 4700;
         port[1] = 4701;
         num_agents = 2;
    }

#ifndef _NO_LOGGING
        DefaultLog::log()->set_filter(ERROR_LOG, 5);
```

```
        DefaultLog::log()->set_filter(WARNING_LOG, 5);
        DefaultLog::log()->set_filter(EVENT_LOG, 5);
        DefaultLog::log()->set_filter(INFO_LOG, 5);
        DefaultLog::log()->set_filter(DEBUG_LOG, 1);
#endif
        Snmp::socket_startup();  // Initialize socket subsystem

        init_signals();

    ThreadPool agentPool(num_agents);
    agentPool.set_one_time_execution(TRUE);
    SnmpAgent* agents[num_agents];
    for (int i=0; i<num_agents; i++) {
        UdpAddress inaddr("0.0.0.0");
        inaddr.set_port(port[i]);
        agents[i] = new SnmpAgent(inaddr);
        agentPool.execute(agents[i]);
    }
    agentPool.join();
        Snmp::socket_cleanup();  // Shut down socket subsystem
        return 0;
}
```