Overview

Part I

Introduction

1 Introduction

Smart-SNMPd Nagios-Plugins

Jens Rehsack

2011

Audience

Audience

- Developer who wants to create monitoring plugins (for Nagios) querying Smart-Snmpd
- Developer who wants to adapt/modify existing plugins querying Smart-Snmpd (eg. to add a smooth migration support for currently existing infrastructure)
- Developer who wants to understand the already existing nagios plugins querying smart-snmpd

Audience

Audience

- Developer who wants to create monitoring plugins (for Nagios) querying Smart-Snmpd
- Developer who wants to adapt/modify existing plugins querying Smart-Snmpd (eg. to add a smooth migration support for currently existing infrastructure)
- Developer who wants to understand the already existing nagios plugins querying smart-snmpd

Prerequisites of the Audience

Following knowledge is expected:

- advanced skills in at least one object oriented programming language
- more than one year practical experience in object oriented development
- advanced C++ (or expert C) knowledge
- experience in *Generative Programming* using the C++ feature *Templates*
- Experience with Unix or compatible operating systems

Nomenclature

C++-Sourcefile from the project

This is an excerpt from a c++ source file from the Smart-Snmpd-Nagios-Plugins project

Nomenclature

C++-Sourcefile from the project

This is an excerpt from a c++ source file from the Smart-Snmpd-Nagios-Plugins project

C++-Headerfile from the project

This is an excerpt from a c++ header file from the Smart-Snmpd-Nagios-Plugins project

Nomenclature

C++-Sourcefile from the project

This is an excerpt from a c++ source file from the Smart-Snmpd-Nagios-Plugins project

C++-Headerfile from the project

This is an excerpt from a c++ header file from the Smart-Snmpd-Nagios-Plugins project

C++-Headerfile from the boost-libraries

This is an excerpt from a c++ header file from the boost project
More about boost at `http://www.boost.org/`

Nomenclature

C++-Sourcefile from the project

This is an excerpt from a c++ source file from the Smart-Snmpd-Nagios-Plugins project

C++-Headerfile from the project

This is an excerpt from a c++ header file from the Smart-Snmpd-Nagios-Plugins project

C++-Headerfile from the boost-libraries

This is an excerpt from a c++ header file from the boost project
More about boost at `http://www.boost.org/`

C++-Headerfile aus der Standard-Template-Bibliothek

This is an excerpt from a c++ header file from the standard template library
Reference of the STL eg. at `http://www.cplusplus.com/reference/`

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

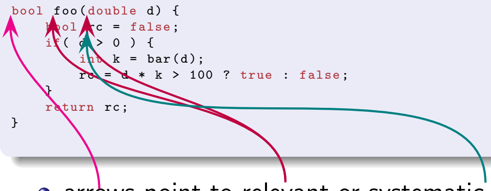
- arrows point to relevant

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction



```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to relevant or systematic important

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to relevant or systematic important code parts

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to relevant or systematic important code parts
- separate logic sections

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to relevant or systematic important code parts
- separate logic sections have own colors

Examples

Code-Style

- Sheets have limited room
- code examples are reduced to the max
- or concentrated to a square hunk
- original code is the authoritative reference, shown examples are for demonstration only

exampleconstruction

```
bool foo(double d) {
    bool rc = false;
    if( d > 0 ) {
        int k = bar(d);
        rc = d * k > 100 ? true : false;
    }
    return rc;
}
```

- arrows point to relevant or systematic important code parts
- separate logic sections have own colors
- Finally the "typical" result stays flagged

Motivation

C++

- performance advantage compared to pluings in interpreted languages like Shell or the Perl Programming language
- Smart-Snmpd is written in C++, too - code sharing is possible

Motivation

C++

- performance advantage compared to pluings in interpreted languages like Shell or the Perl Programming language
- Smart-Snmpd is written in C++, too - code sharing is possible

Standard Template Library

- the STL is part of the language standard since C++98
- the amount of STL based projects is constantly growing
- advanced STL knowledge can implied meanwhile when hiring C++ developers
- STL is incredible fast

Motivation

C++

- performance advantage compared to pluings in interpreted languages like Shell or the Perl Programming language
- Smart-Snmpd is written in C++, too - code sharing is possible

Standard Template Library

- the STL is part of the language standard since C++98
- the amount of STL based projects is constantly growing
- advanced STL knowledge can implied meanwhile when hiring C++ developers
- STL is incredible fast

Boost

- *Boost* implements a lot of solutions which didn't made it into the C++98 standard, but are scheduled (TR1, TR2) for the upcoming ones (eg. `any`, `array` or `variant`)
- portability of a library is a prerequisite for it's acceptance into *Boost*

Overview

Part II

Step by Step to a working Plugin

2 CPU Load
- Includes
- Mibdata for Smart-Snmpd
- Mibdata for Net-Snmpd
- Plugin Application Controller
  - Initialization
  - Reporting
  - Fetching, Checking, ...
- Plugin's main routine

#includes

src/ check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

#includes

src/ check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)

#includes

src/ check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)
- persistent settings from configure

#includes

src/check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)
- persistent settings from configure
- Known oids

#includes

src/ check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)
- persistent settings from configure
- Known oids
- predefined checks (used to define the application class)

#includes

src/check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)
- persistent settings from configure
- Known oids
- predefined checks (used to define the application class)
- predefined check types (used to define check class)

#includes

src/check_proc_cnt_by_snmp.cpp

```
#include <smart-snmpd-nagios-plugins-build-defs.h>
#include <smart-snmpd-nagios-plugins/smart-snmpd-nagios-plugins.h>

#include <smart-snmpd-nagios-plugins/oids.h>
#include <smart-snmpd-nagios-plugins/snmp-check.h>
#include <smart-snmpd-nagios-plugins/snmp-check-types.h>
#include <smart-snmpd-nagios-plugins/snmp-check-appl.h>
```

- system include wrapper (resolves right includes as detected by configure)
- persistent settings from configure
- Known oids
- predefined checks (used to define the application class)
- predefined check types (used to define check class)
- skeleton for typical plugin application

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```cpp
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o 0) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define `convert`-method

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define `convert`-method with list of fetched *VarBind*s

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define `convert`-method with list of fetched *VarBind*s and the `datamap` to hold the converted data

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define convert-method with list of fetched *VarBind*s and the datamap to hold the converted data
- extract the fetched value into useful data type

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class` `SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define convert-method with list of fetched *VarBind*s and the `datamap` to hold the converted data
- extract the fetched value into useful data type
- insert extracted data into result `datamap`

SmartSnmpdProcessCountMibData

src/check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class` `SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define `convert`-method with list of fetched *VarBind*s and the `datamap` to hold the converted data
- extract the fetched value into useful data type
- insert extracted data into result `datamap` using the predefined key `ProveValueMapKey`

SmartSnmpdProcessCountMibData

src/ check_proc_cnt_by_snmp.cpp

```
static const Oid SmProcCntOids[] = { SM_PROCESS_TOTAL };
class SmartSnmpdProcessCountMibData : public SupportedMibData {
public:
    SmartSnmpdProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(SmProcCntOids)>( SmProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class` `SupportedMibData` specialization to fetch the total process count from a Smart-Snmpd
- define convert-method with list of fetched *VarBind*s and the `datamap` to hold the converted data
- extract the fetched value into useful data type
- insert extracted data into result datamap using the predefined key `ProveValueMapKey`
- throw an `snmp_bad_result` exception when extracting value fails

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.0) or corrupt" );
        }
    }
};
```

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.0) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.O) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class` `SupportedMibData` specialization to fetch the total process count from eg. a Net-Snmpd

HostResourcesProcessCountMibData

Repeat this for each supported MIB:

src/check_proc_cnt_by_snmp.cpp

```
static const Oid HrProcCntOids[] = { HR_SYSTEM_PROCESSES ".0" };
class HostResourcesProcessCountMibData : public SupportedMibData {
public:
    HostResourcesProcessCountMibData()
        : SupportedMibData( make_vector<Oid, lengthof(HrProcCntOids)>( HrProcCntOids ) )
    {}

    virtual void convertSnmpData( vector<Vb> const &vblist, DataMapType &dataMap )
    {
        unsigned long proc_cnt;
        if( SnmpComm::extract_value( vblist[0], proc_cnt ) ) {
            AbsoluteThreshold data( proc_cnt );
            dataMap.insert( make_pair( ProveValueMapKey, data ) );
        }
        else {
            throw snmp_bad_result( "Process count incomplete (o.0) or corrupt" );
        }
    }
};
```

- define array of oids containing the identifiers of the objects to fetch
- Implement a `class SupportedMibData` specialization to fetch the total process count from eg. a Net-Snmpd
- . . .

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl`

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd*

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd*

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd* to the plugin application by pushing their identifiers into `mSupportedSnmpDaemons` (order matters)

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd* to the plugin application by pushing their identifiers into `mSupportedSnmpDaemons` (order matters)
- provide the `SupportedMibData` specialization

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd* to the plugin application by pushing their identifiers into `mSupportedSnmpDaemons` (order matters)
- provide the `SupportedMibData` specialization according to

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```cpp
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd* to the plugin application by pushing their identifiers into `mSupportedSnmpDaemons` (order matters)
- provide the `SupportedMibData` specialization according to the identified daemon

Plugin Application Controller (I)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    virtual void initSupportedSnmpDaemons() {
        mSupportedSnmpDaemons.push_back( IdentifySmartSnmpdMib );
        mSupportedSnmpDaemons.push_back( IdentifyNetSnmpd );
    }

    virtual SupportedMibDataType * getMibData( SnmpDaemonIdentifier const &identifiedDaemon ) {
        if( identifiedDaemon.getName() == IdentifySmartSnmpdMib.getName() )
            return new SmartSnmpdProcessCountMibData();
        else if( identifiedDaemon.getName() == IdentifyNetSnmpd.getName() )
            return new HostResourcesProcessCountMibData();

        throw unknown_daemon();
    }
};
```

- define Application class derived from `CheckPluginAppl` fetching static objects and have separate warn and crit threshold
- introduce the daemons *Smart-Snmpd* and *Net-Snmpd* to the plugin application by pushing their identifiers into `mSupportedSnmpDaemons` (order matters)
- provide the `SupportedMibData` specialization according to the identified daemon
- throw `unknown_daemon` when unknown daemon has been identified (shouldn't happen, but better safe than sorry)

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects , SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status
- provide check name for generating status message

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status
- provide check name for generating status message
- provide application name,

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status
- provide check name for generating status message
- provide application name, version string

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status
- provide check name for generating status message
- provide application name, version string and application description

Plugin Application Controller (II)

src/check_proc_cnt_by_snmp.cpp

```
class SnmpProcessCountCheckAppl
    : public CheckPluginAppl< FetchStaticObjects, SnmpWarnCritCheck< AbsoluteThreshold > > {
public:
    string createResultMessage( DataMapType const &dataMap ) const {
        string msg = to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + " procs currently running";
        return msg;
    }

    string createPerformanceMessage( DataMapType const &dataMap ) const {
        string msg = string("procs=") + to_string(dataMap[ProveValueMapKey].as<AbsoluteThreshold>()) + ";";
        return msg;
    }

protected:
    virtual string const getCheckName() const { return "PROCS"; }
    virtual string const getApplName() const { return "check_proc_cnt_by_snmp"; }
    virtual string const getApplVersion() const { return SSNC_VERSION_STRING; }
    virtual string const getApplDescription() const {
        return "Check count of running processes via Simple Network Management Protocol";
    }
};
```

- create result and performance messages to report status
- provide check name for generating status message
- provide application name, version string and application description to generate help output

Plugin Application Controller (III)

Missing something?

Plugin Application Controller (III)

Missing something?
Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

Plugin Application Controller (III)

Missing something?
Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

- command line parameters are defined as `SnmpCheckAppl` is composed

Plugin Application Controller (III)

Missing something?
Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

- command line parameters are defined as `SnmpCheckAppl` is composed
- objects from *SNMP daemon* are fetched as described in provided `SupportedMibDataType`

Plugin Application Controller (III)

Missing something?
Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

- command line parameters are defined as `SnmpCheckAppl` is composed
- objects from *SNMP daemon* are fetched as described in provided `SupportedMibDataType`
- comparing simple data types is built-in

Plugin Application Controller (III)

Missing something?
Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

- command line parameters are defined as `SnmpCheckAppl` is composed
- objects from *SNMP daemon* are fetched as described in provided `SupportedMibDataType`
- comparing simple data types is built-in
- ...

Plugin Application Controller (III)

Missing something?

Remember, this is a framework to develop snmp check plugins. That means, a lot of things are done automatically unless developer intervenes:

- command line parameters are defined as `SnmpCheckAppl` is composed
- objects from *SNMP daemon* are fetched as described in provided `SupportedMibDataType`
- comparing simple data types is built-in
- . . .

More complicated examples follow in later parts.

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application

Plugin's main routine

src/ check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line,

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know ...

Plugin's main routine

src/ check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know . . .
- . . . the objects to fetch

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know ...
- ... the objects to fetch for proving

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know . . .
- . . . the objects to fetch for proving and reporting

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know ...
- ... the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared ...)

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know . . .
- . . . the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared . . .)
- catch

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know . . .
- . . . the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared . . .)
- catch and handle different exceptions,

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know ...
- ... the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared ...)
- catch and handle different exceptions, set error message

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know . . .
- . . . the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared . . .)
- catch and handle different exceptions, set error message
- report status to standard output

Plugin's main routine

src/check_proc_cnt_by_snmp.cpp

```
int main(int argc, char *argv[]) {
    int rc = STATE_EXCEPTION;
    SnmpProcessCountCheckAppl checkAppl;
    string msg;

    try {
        checkAppl.setupFromCommandLine(argc, argv); checkAppl.configure();
        checkAppl.identifyDaemon();

        checkAppl.fetchData(); checkAppl.convert();

        checkAppl.createMessages();
        rc = checkAppl.prove< std::greater_equal<AbsoluteThreshold> >();
    }
    catch(alarm_timeout_reached &a) { rc = STATE_UNKNOWN; msg = a.what(); }
    catch(snmp_error &s) { rc = STATE_UNKNOWN; msg = s.what(); }
    catch(std::exception& e) { cerr << (msg = e.what()) << endl; }

    return checkAppl.report(rc, msg);
}
```

- Setting up the plugin application from command line, configure it upon the received parameters and identify the daemon we're checking to know ...
- ... the objects to fetch for proving and reporting
- convert fetched data (data aren't everytime in suitable format when coming from *SNMP daemon* and different daemons usually deliver different data which must be normalized to get compared ...)
- catch and handle different exceptions, set error message
- report status to standard output and return status code (`CheckPluginAppl::report()` returns given rc)

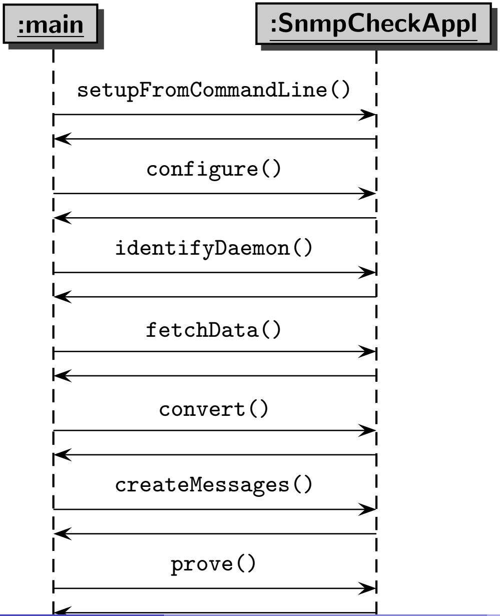
Overview

Part III

Course of a Plugin

3 Structure of a Plugin
- Application Flow

Application Flow


:main
:SnmpCheckAppl
setupFromCommandLine()
configure()
identifyDaemon()
fetchData()
convert()
createMessages()
prove()