CcspSnmpPa

Introduction

SNMP Protocol Agent (PA) provides the solution for SNMP Protocol to access CCSP Data Model.

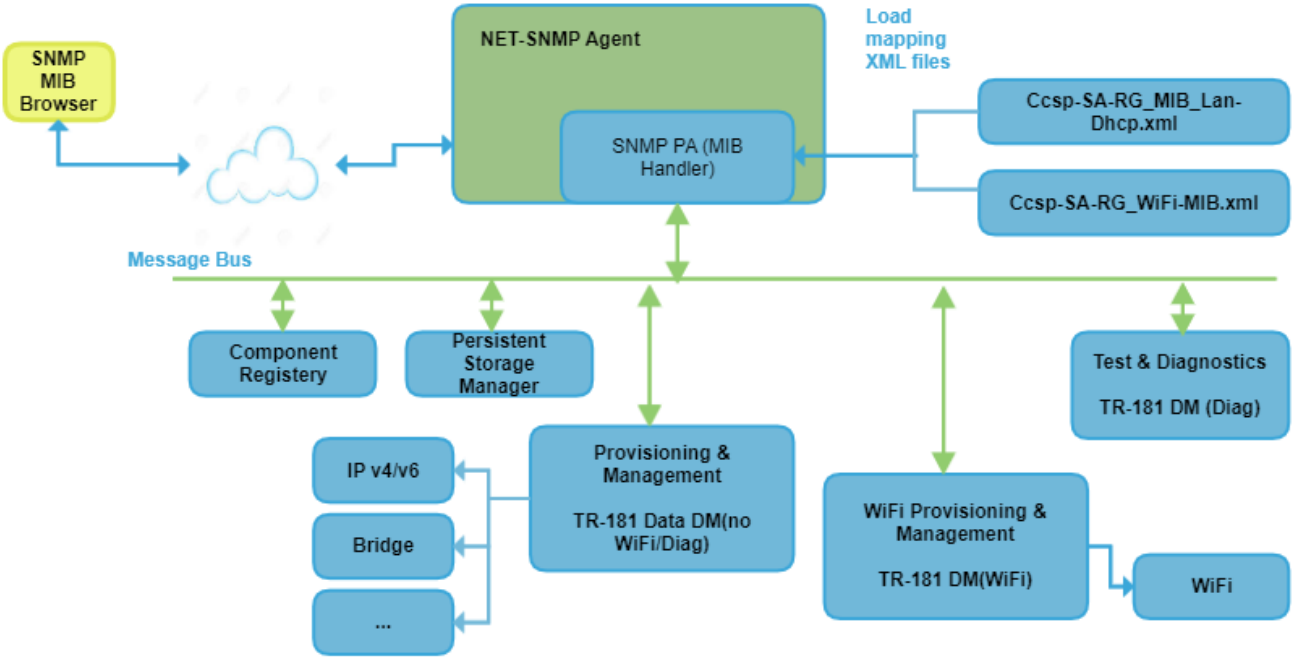
Externally, SNMP protocol agent works like a regular SNMP agent. It takes coming SNMP requests and sends back the corresponding results or errors.

Internally, SNMP protocol agent processes the SNMP requests and translates those into CCSP Messages. Those messages go on to CCSP Message Bus and come back with response messages. SNMP protocol agent processes the response messages and response to SNMP requests accordingly.

In order to translate between SNMP requests and CCSP Messages, SNMP protocol agent loads the XML files describing how MIB objects are mapped to Data Model objects.

Architecture

This is the architecture diagram of SNMP Protocol Agent



CCSP SNMP Protocol Agent is implemented as a NET-SNMP MIB handler. During startup, it loads all the XML files defining the following:

- MIB object – Data Model object mapping
- Custom callback APIs

After it is up, CCSP SNMP Protocol Agent processes the SNMP requests handed to it by NET- SNMP agent, send the translated CCSP Messages to the destination components, processes the responses and provide the result back to SNMP agent.

MIB to Data Model Mapping XML Schema

Root Node

The root node is named as “mib2DM”. It is constructed with a set of scalar mib groups, mib tables and some informational nodes.

MIB Custom Callback APIs

If special logic is required for certain MIB objects beyond MIB – Data Model mapping can support, custom callback functions can be utilized.

The callback prototype is:

```
typedef int (*handleRequestsProc)

(

netsnmp_mib_handler *handler,

netsnmp_handler_registration *reginfo,

netsnmp_agent_request_info *reqinfo,

netsnmp_request_info *requests

);
```

This callback will be called before SNMP Protocol Agent MIB handler processing the requests. If the callback takes care of certain request and it doesn’t want to be processed anymore, the “requests->processed” should be set to 1. In this case, the MIB handler will ignore this request and move on.

The XML schema for custom callbacks is:

<!---- Definition of callbackInfo ---->

```
<xs:complexType name="callbackInfo">

    <xs:sequence>

        <!----  The prototype of the “handleRequests”:

            typedef int

            (*handleRequestsProc)

                (

                    netsnmp_mib_handler *handler,

                    netsnmp_handler_registration *reginfo,

                    netsnmp_agent_request_info *reqinfo,

                    netsnmp_request_info *requests

                );

            This callback will be called before Ccsp generic MIB handler.

            Whenever a request was handled in this callback, you may set “request-->processed” to 1.

        ---->

        <xs:element name="handleRequests" type="xs:string" minOccurs=”0” />

        <!----  The prototype of the “refreshCache”:

            typedef int

                (*refreshCacheProc)

                (

                    netsnmp_tdata* table

                );

            This callback will be called first in RefreshCache of Ccsp generic MIB handler.

        ---->

        <xs:element name="refreshCache" type="xs:string" minOccurs=”0” />

    </xs:sequence>

</xs:complexType>
```

Node of “mapping”

Mapping is consisted of two parts,

- MIB object, identified by the OID
- Data Model object/parameter, identified by the Data Model object/parameter name.

The schema is as follows:

<!---- Definition of mibInfo ---->

```
<xs:complexType name="mibInfo">
    <xs:sequence>
        <xs:element name="lastOid" type="xs:positiveInteger" />
        <!-- The “name” will not be used by SNMP agent, but it will be helpful when editing the mapping file.
It’s optional. ---->

        <xs:element name="name" type="xs:string" minOccurs=”0” />
        <xs:element name="access" type="mibAccess" />
        <!— The data type of this mib ---->

        <xs:element name="dataType" type="xs:string" />
        <xs:element name="range" type="rangeInfo" minOccurs=”0”/>
    </xs:sequence>
</xs:complexType>
```

<!---- Definition of dmInfo ---->

```
<xs:complexType name="dmInfo">
    <xs:sequence>
        <!-- It defines the mapped Data Model parameter name. It could be under an object or a table.  ---->

        <!-- For instance, “Device.DeviceInfo.SoftwareVersion” or “Device.Hosts.Host.%d.Name”. ---->

        <xs:element name="paramName" type="xs:string" />
        <xs:element name="dataType" type="trDataType " />
        <!-- Enumeration is an integer in MIB while a string value in TR data model. We put one single string
including the whole mapping. ---- >

        <!-- For instance “None(0), Requested(1), Complete(2), Error(3)”. The maps are separated by a ‘,’.---->

        <!----  The bitmask type is also supported. It’s a choice of data type extension here ---->

        <xs:choice minOccurs=”0”>
            <xs:element name="enumeration" type="xs:string" />
            <xs:element name="bitmask" type="xs:string" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

<!---- Definition of mappingInfo ---->

```
<xs:complexType name="mappingInfo">
    <xs:sequence>
        <xs:element name="mib" type="mibInfo" />
            <!—Certain mibs may not need a mapping in DM, such as “StorageType”. Hence it’s optional. ---->

        <xs:element name="dm" type="dmInfo" minOccurs=”0” />
    </xs:sequence>
</xs:complexType>
```

Node of “scalarGroup”

This node defines how scalar MIB objects are mapped to Data Model objects:

<!---- Definition of a scalarGroupInfo ---->

```
<xs:complexType    name="scalarGroupInfo">
    <xs:sequence>
        <!----  Unique name of group name. It could be the name of baseOid. It will be used for the hook/callback
registration. ---->

        <xs:element name="name" type="xs:string"/>
        <!----  “baseOid” is the base OID string for this group of scalar mibs. All the mibs have to be in the
same OID layer. ---->

        <!----  Otherwise, put them in another scalarGroup. ---->

        <xs:element name="baseOid" type="xs:string"/>
        <!----  “enabled” is defined in case this group is not ready at back--end yet or deprecated ---- >

        <!----  If it’s not enabled, the MIB handler will not load it. It’s enabled if the node doesn’t present
(by default) ---->

        <xs:element name="enabled" type="xs:boolean" minOccurs="0"/>
        <!----  Cache is always recommended for performance concerns. The default is 30 seconds ---->

        <xs:element name="cacheTimeout" type="xs:nonNegativeInteger" minOccurs="0" />
        <!----  In some special case, the scalar mibs may map to an entry in Data Model Table. ---->

        <!----  This optional node provides conditional mapping. For instance, “Wifi.Radio.%d.Frequency=2.4” ----

        <xs:element name="mapToEntry" type="xs:string" minOccurs="0" />
        <!----  If callback/hook functions are needed for this group, add it here. ---->

        <xs:element name="callbacks" type="callbackInfo " minOccurs="0" />
        <!----  a set of mibs ---->

        <xs:element name="mapping" type="mappingInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
Node of “mibTable”
```

Because table objects covers more than parameters, more work and functionalities are included:

- Index mapping – for MIB and Data Model table objects not sharing the same index scheme
- Filtering – Data Model table may contains more instances than SNMP wants. Filtering criteria can be given to pick only the entries matching the criteria. The criteria can be on parameter level, e.g., “Device.Routing.Router.1.IPv4Forwarding.%d.StaticRoute = true”.

<!---- Definition of indexInfo ---->

```
<xs:complexType name="indexInfo">
    <xs:sequence>
        <xs:element name="mib" type="mibInfo" />
        <xs:choice>
            <!-- The mib may be mapped to the instance number of data model entry. It’s false by default. ---->

            <xs:element name="mapToInsNumber" type="type:mapToInsInfo" />
            <!-- If it’s not mapped to instance number, it will be mapped to a data model parameter. ---->

            <xs:element name="dm" type="dmInfo" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

<!---- Definition of a mibTableInfo ---->

```
<xs:complexType name="mibTableInfo">
    <xs:sequence>
        <!----  Mib table name. It will be used for the hook/callback registration. ---->

        <xs:element name="name" type="xs:string"/>
        <!----  The fully OID of the mib table (not entry). For instance “1,3,6,1,4,8072,2,2,2“.---->

        <!----  Otherwise, put them in another scalarGroup. ---->

        <xs:element name="tableOid" type="xs:string"/>
        <!----  “enabled” is defined in case this group is not ready at back--end yet or deprecated ---- >

        <!----  If it’s not enabled, the MIB handler will not load it. It’s enabled if the node doesn’t present
(by default) ---->

        <xs:element name="enabled" type="xs:boolean" minOccurs="0"/>
        <!----  Whether the table is writable or not. It’s not by default. ---->

        <xs:element name="writable" type="xs:boolean" minOccurs="0"/>
        <!----  Specify the maxi entries in the table. Default is 16. ---->

        <xs:element name="maxEntries" type="xs:positiveInteger" minOccurs="0"/>
        <!----  Cache is always recommended for performance concerns. The default is 30 seconds ---->

        <xs:element name="cacheTimeout" type="xs:nonNegativeInteger" minOccurs="0" />
        <!----  In some special case, the mib table maps to some entries in a DM table. ---->

        <!----  The filter may be added here to achieve this goal. For instance, “Routing.%d.type = static” ---->

        <xs:element name="mapToEntries" type="xs:string" minOccurs="0" />
        <!----  If callback/hook functions are needed for this group, add it here. ---->

        <xs:element name="callbacks" type="callbackInfo " minOccurs="0" />
        <!----  a set of indexs ---->

        <xs:element name="index" type="indexInfo" minOccurs="1" maxOccurs="8"/>
        <!----  a set of mibs ---->

        <xs:element name="mapping" type="mappingInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

Complete Schema

Find a sample Complete_Schema.xml .

Mapping XML File Examples

Please find a sample mapping file : Ccsp_RDKB-RG-WiFi-MIB.xml .