BGP Module Documentation for the PYenca Agent

Humberto Abdelnur, Radu State

BGP Module Documentation for the PYenca Agent

Release: BGP-PYenca-1.0.0

Humberto Abdelnur, Humberto.Abdenur@loria.fr
Radu STATE, state@loria.fr

Madynes Research team
LORIA-INRIA Lorraine
rue du jardin botanique
Villers-les-Nancy
FRANCE

Contents

List of Figures

Abstract

This documentation stand for the design, implementation ideas and some features and capabilities added to the NETCONF Protocol through the BGP implementation process. Also, it explains a resulting library created for this implementation to transforms from the data device to the XML design and backward, that could be use for the implementation of new modules.

1 Introduction

1.1 Information

1.1.1 BGP-PYenca's author information

Author names and organism attachment:
Humberto Abdelnur and Radu STATE
Madynes Research team
LORIA-INRIA Lorraine
rue du jardin botanique
Villers-les-Nancy
FRANCE

If you have any requests, comments, ideas for improvement or questions concerning Fuzzy_Packet, if you think you find a bug, please send emails to Humberto.Abdelnur@loria.fr.

1.1.2 Software information

Software name: BGP-PYenca
Version: 1.0.0
Programming language: Python
Operating System: Linux (tested in Gentoo)
License: GNU Lesser General Public License (LGPL)

1.1.3 Other information

This environment is strongly tied to the NetConf Configuration protocol. This protocol is based on the Manager/Agent paradigm, illustrated on Figure 1, which states that a manager (the client) sends some XML-based queries to the agent (the server) which, in turn, sends back, also, a XML-based reply for each query to the manager. A query will be one of these well-defined operations:

get, get-config allows to retrieve the state and configuration data of the device. Two methods may be used to fetch remote data: subtree filtering and XPath selection if that capability is supported by the agent,

edit-config allows to modify the remote configuration. The *operation* option can be set to *replace*, *merge*, *create* or *delete* some XML nodes,

copy-config, delete-config allows to push or to delete a target (running, candidate, startup) XML configuration of a device,

lock, unlock allows to lock and unlock some resources. For instance, locking the access to one or more resources so that no other modification can be performed on the device.

close-session, kill-session allows to close a session when needed and respectively to kill a session, if needed (for instance after a management fault).

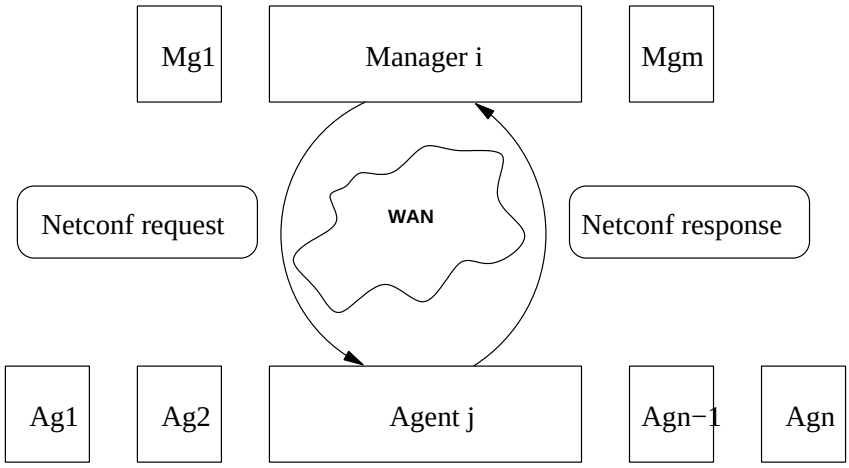

Figure 1: Manager/Agent paradigm with NetConf

In order for NETCONF to manage different devices, it allows to add new Plugins (or Modules) easily. Those modules should implement some default interface which will allow them to be added dynamicly to the Agent.

The BGP Network Management Framework is implemented as a module of the NetConf Agent. It must be registered as a module in the modules configuration file.

The BGP module is presented in detail in this document explaining its architecture, functional and static Design as well as the XML design specified to manage the device. Also, the implementation of this module made a library that could be use for the construction of other protocols, making their implementation much easier.

This is the first module implemented for the NetConf Agent, giving us a full test-bed for testing its ability to deal with complex and large-scale configuration. We guess that many other modules will follow a similar approach that can be defined over the following main steps:

- building a XML data model with XML Schema,
- building a parser to read proprietary configuration files,
- providing a mapping schema from one data model to the other.

2 Installation

2.1 System Requirements

- **Python[7]**: 2.3.0 or newer. The whole implementation of the Module is done in Python so, check the appropriate Python package for your distribution or download the sources from www.python.org .
- **4Suite[1]**: 1.0 alpha 4 or newer. The XML processing is done entirely via this library. Check the package for your distribution or download the sources from it webpage. In the later case you should untar the sources and invoke "python setup.py install" in the top-level source directory.
- **Yapps[6]**: version 2 or newer. The parser of the data from the device is done by a class generated by this parser generator. Download the zip file from the YAPPS webpage. The files included in the zip don't need to be compiled. Unzip them in your library folder or any place you consider. Add the path of the folder you decided to the $PYTHONPATH environment variable.
- **Quagga[3] :** The BGP Module is responsible for configuring the device, it was fully tested with this routing software suite. It implementation is under the Unix Platform. Check the package for your distribution or download the sources from it site. Note that there is a bug in the versions prior to 0.98.3 that doesn't allow to execute the command "no router bgp AS-NUMBER" under certain circumstances (reffer to `http://lists.quagga.net/pipermail/quagga-users/2005-May/004606.html`), if the version you install also has this bug, you wont be able to execute NetConf operations as copy-conf, for example, because the Quagga soft will crash.
- **NetConf Agent[2]:** Check in our page the last release of it.

2.2 Installation

We provide the source code of the BGP Module, to install is quite easy, copy the main folder BGP_Module to the folder Module in the Server Agent and execute "python setup.py".

In order to the Agent to register this module, a node like the one show next should be added to the modules.xml file. For more information check the documentation of the Agent.

```
<module>
   <name>BGPModule</name>
   <mainFileName>BGP_Module</mainFileName>
   <className>BGP_Module</className>
   <xpath>/netconf/routing/bgp</xpath>
   <xsdfile></xsdfile>
   <parameters>
      <host>127.0.0.1</host>
      <port>2605</port>
      <password>pass</password>
      <enable_pass></enable_pass>
      <instance></instance>
      <allowxpath>true</allowxpath>
```

```
   </parameters>
</module>
```

Also it is necessary to configure the Quagga soft to allow VTY telnet connection, which can be done by adding the "password YOUR-PASSWORD" line to the bgpd.conf file (usually located in /etc/quagga).

2.3 Launching the Module

As the BGP implementation is a module of the NetConf agent yencap, it will automatically launch it self if the step specified in the installation were followed. Still, as this module is only a translator of protocols (NETCONT-CLI), it is necessary to launch the Quagga soft before commit any action with this module. In order to launch Quagga, you should have zebra running (by the command "zebra") and type "bgpd", check the help to see which options do you want to run.

2.4 Test Bed

Our testing and developing was carried through a test-bed that was made by the UML[4] (User Mode Linux Kernel) and managed by VNUML[5] (Virtual Network User Mode Linux). Also, we made our test-bed available for use through our page. Anyways, if you are interested in install them in your computer, we explain an easy way to do it. Even so, it is not explain here how to compile them from source, we recommend it to achieve better performance in the virtual computers.

Minimal requirements :

- Pentium III with 256 Mb
- Root access privileges
- Perl 5.6 or newer
- wget
- libxml2, expat2. Check the availability of these packages in your current distribution, more information could be get from http://xmlsoft.org/ and http://expat.sourceforge.net/
- Internet Connection for installing VNUML

Installing procedure.

- **UML**: A precompiled version of UML kernel can be found in http://jungla.dit.upm.es/~vnuml/#download . Also the root file system could be found there and it has already installed Quagga or could find it in our page [2] with the NetConf agent and BGP module installed on it. For more detailed information about how to compile the kernel, create the root file system, etc. check at the http://user-mode-linux.sourceforge.net .

  In order to add the Agent to the root file system, if not in it, or any other software, you should mount it in a regular folder as follow:

```
mount -o loop root_filesystem /mnt/loop/
chroot /mnt/loop /bin/bash
source /etc/profile
```

Then you can install any software in this terminal as if you were in the actual file system. Note that it has the structure of a regular filesystem. The root password for the file system is: xxxx, you could change it if you wish so. Once finishes exit and unmount the file system.

- **VNUML**: The VNUML releases can be found in http://jungla.dit.upm.es/~vnuml/#download . Once the file is downloaded and unzipped, you should proceed as follow.

```
./configure
make
make install
```

You could check that the installation was okay using:

```
vnulmparser.pl -V
```

2.4.1 Trying the Test-Bed

First you will need a sample network, there are some examples in http://jungla.dit.upm.es/~vnuml/#examples and also our page.

The sample will have a XML Document Topology, which specify how the scenario of the network will look like and some extra files that will be needed for configuring each of the UMLs. After you download you should modify the paths and version according to the ones specify in your computer.

Also, as VNUML uses SSH to interact with the UMLs, it is suggest to generate a public-key to avoid been asking for the password every time you run the Test-Bed. To create such a key run :

```
ssh-keygen -t rsa1
```

The first thing to do will be to boot the UML's and create the topology network, were topology.xml is the XML Document Topology.

```
vnumlparser.pl -t topology.xml -vB
```

This operation will take a while, consider that you are booting as many machine as you specify in the XML Document Topology. After the UMLs where created, you should start them by.

```
vnumlparser.pl -s topology.xml -vB
```

And at this point, if you success, you have the Test-Bed running at your machine. To stop the test-bed but keeping the UMLs alive run:

```
vnumlparser.pl -p topology.xml -vB
```

Finally, to shutdown the UMLs

```
vnumlparser.pl -d topology.xml -vB}
```

3 Lessons Learned

3.1 Semantic differences and side effects

The design of a XML based agent for BGP management raised several interesting problems related to side effects due to structural differences between XML document driven configuration and CLI oriented management. We will illustrate such a case, and for clarity sake, keep the example as simple as possible. Assume that a management request should delete a node in a given configuration document. According to the NetConf protocol, this node and only this one must be deleted. However, consequently to the constraints of the device, some operations performed on it could also delete other data and the configuration could become inconsistent. The problem is that the resulting XML document does not reflect the expected configuration because some other data in the configuration tree might also have been deleted. This is the case, for instance, when a node is deleted and other nodes reference it, or when the update of a value invalidates some other nodes.

The constraint required to be addressed is that the result should remain consistent with the device. The configuration of the BGP plane with CLI might sometimes also delete other data. We identified two solutions for this problem:

- Accept that the XML result will be different than the expected XML configuration document result, but it will be consistent with the device and semantically acceptable,
- Cancel the operation if the XML result is not acceptable and send back an error.

Since each moderately complex protocol defines such constraints, this problem is recurrent not only for the BGP configuration plane, but also on a more larger scale. In our work, we solved the problem by domain specific knowledge and exception handling, but a more general solution would be needed to efficiently drive the global XML to CLI translation. Suppose the BGP configuration task shown in Figure 2.

The request (Figure 2 *step 2*), requires to bind the "Fg" group to every neighbor.

Note that the resulting configuration (Figure 2 *step 3*) does not include information about the **next-hop-self** property, and in the original request this property was not supposed to be deleted (Figure 2 *step 1*). However, the final configuration does not include any more reference to this property, which is silently removed. The reason for this is that in configuring BGP, if a group is set to a neighbor, the latter can not specify individual properties. This is side effect due to BGP domain specifics. Note that this problem arises in the XML as well as in the CLI configuration.

4 Functionality Overview

We will describe in this section the fundamentals underlying of the design and implementation of a BGP module for PYenca.

The major requirement we had to address consisted in an efficient XML to CLI transparent translation, allowing an easy adaptation to other vendor

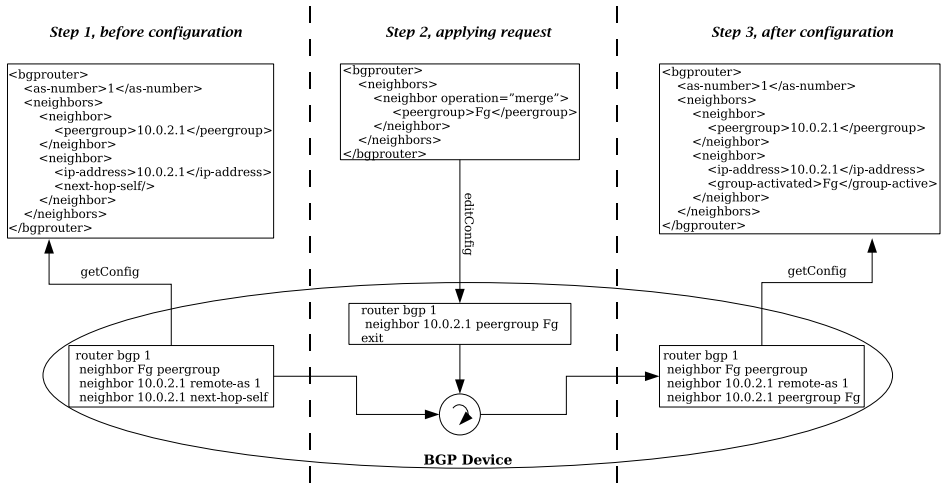

Figure 2: BGP Configuration Constraints

specific command line interfaces. The overall module architecture comprises three major building blocks.

- A complete language designed to drive the XML to CLI bidirectional translation, allowing to adapt the module to any vendor specific CLI.
- An interface with the PYenca agent
- An interface with the device via CLI.

A high level functional architecture of the BGP module is shown in Figure 3.

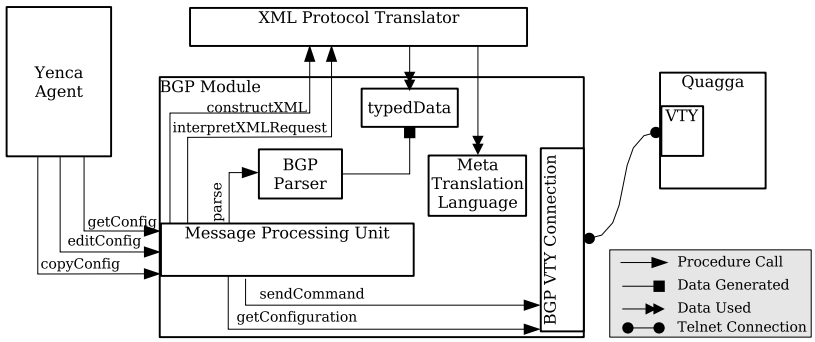

Figure 3: BGP Module's Functional Architecture

4.1 Message Processing Unit

The interactions with the managed device and the PYenca agent is performed by the Message Processing Unit (MPU). This entity is able to 1) perform run

time subscription on the PYenca agent, 2) to receive, process and send XML requests from the agent and 3) interact via CLI with the Quagga router.

The main actions related to processing XML requests are:

- **getconfig request:** It requests the **BGP CLI Connection** to get the configuration from the device and next it generates a structure of primitives types **typedData**, using the **BGP Parser**. Once the structure is ready, the construction of the XML document is done by the **XML Protocol Translator** library and the **constructXML** function. This step is based on the usage of the generated **typedData** and the **Meta Translation Language**.
- **editconfig request:** Received request concerning edit-config are translated to CLI commands. The components used by this activity are done using the **Meta Translation Language** structure, and the **XML Protocol Translator**. The generated commands are sent to the device through the **BGP CLI Connection**.
- **copyconfig request:** this function is just a particular case of the edit-config activity.

4.2 BGP CLI Connection

Its function is to communicate with the real device, in this case is through a telnet connection to the VTY -Virtual Terminal Interface- implemented by the Quagga software. In our implementation we relied on the telnetlib library provided by the default Python installation.

4.3 BGP Parser

This entity generates the **typedData** structure by parsing the raw data obtained from the CLI connection to the device. The parser is based on YAPPS2 [6], an outstanding parser generator for Python. The syntax defined by YAPPS includes rules triggered by the matching of regular expression. A rule is associated to an action which consists in a series of Python language statements. To illustrate this process, let us consider an example XML configuration document (Figure 4) and a small subset of the parser rules (Figure 5).

```
router bgp 1
 neighbor 10.0.2.1 remote-as 1
 neighbor 10.0.2.1 activate
 neighbor 10.0.3.1 remote-as 2
 no neighbor 10.0.3.1 activate
 neighbor 10.0.4.1 remote-as 2
 neighbor 10.0.4.1 activate
 neighbor 10.0.5.1 remote-as 3
 neighbor 10.0.5.1 activate
```

Figure 4: BGP CLI Configuration example

The main rule, "Parser", is triggered which generates a new dictionary, **bgpconf**. Next, it will try to match the regular expression, beginning with "router

```
#########################
## Aditional Python Function     ##
#########################

def getNeighborFrom(neighbor,dataconf):

        if (not dataconf.has_key("neighbors")):
                dataconf["neighbors"] = []
                dataconf["neighbors"].append(neighbor)
        else:
                for x in dataconf["neighbors"] :
                        if (x["ip-address"] == neighbor["ip-address"]):
                                neighbor = x
                                break
                else :
                        dataconf["neighbors"].append(neighbor)

        return neighbor

############################
## Data to generate the Parser File ##
############################
%%
parser BGPParser:
   option:      "context-insensitive-scanner"

   ignore       : "[ \t\r]*"
   token END: "[ \t\r]*\n"
   token STR : "[\.:\-a-zA-Z0-9]*"

   rule Parser:  {{ bgpconf = {} }}
         ("router bgp" STR END              {{ bgpconf["as-number"]  = STR }})
                  Neighbor<<bgpconf>>*
                                            {{ return {"bgprouter":bgpconf} }}

   rule Neighbor<<dataconf>>:               {{ neighbor = {} }}
        (("neighbor" STR                    {{ neighbor["ip-address"] = STR }}
                                            {{ neighbor =
                                                getNeighborFrom(neighbor,dataconf) }}
                ("remote-as" STR            {{ neighbor["remote-as"]  = STR}}
                |"passive"                  {{ neighbor["passive"]    = True }}
                |"activate"                 {{ neighbor["activate"]   = True }}))
        | ("no neighbor" STR                {{ neighbor["ip-address"] = STR }}
                                            {{ neighbor =
                                                getNeighborFrom(neighbor,dataconf) }}
                "activate"                  {{ neighbor["activate"]   = False }}))
        END
%%
```

Figure 5: Parser File

bgp". Once matched, it will try to read the reg. exp. specified by STR, and next the END reg. exp. The resulting data that match the STR will be added to the dictionary **bgpconf** with the key **as-number**. Next, it will try to parse the rest, following the "Neighbor" rule. Note that when it calls it, it does with an argument which is the bgpconf dictionary itself. In the case of the Neighbor, there is one particularity: in order to group all the properties of a neighbor, it creates a dictionary for each of it and appends it to a list, associated to the key **neighbors**. For every command that starts with "neighbor A.B.C.D", it calls the additional function declared in the top, "getNeighborFrom". This is done to find the dictionary representing that neighbor. If this key is not found, it will be created. Finally, it will return a new dictionary with the bgpconf data and with "bgprouter" as its key value. The **typedData** structure generated by the above example will look like Figure 6.

```
TypedData =
            {"bgprouter": {  "as-number":"1",
                             "neighbors":[
                                          {"ip-address":"10.0.2.1",
                                           "remote-as":"1",
                                           "activate":True},
                                          {"ip-address":"10.0.3.1",
                                           "remote-as":"2",
                                           "activate":False},
                                          {"ip-address":"10.0.4.1",
                                           "remote-as":"2",
                                           "activate":True},
                                          {"ip-address":"10.0.5.1",
                                           "remote-as":"3",
                                           "activate":True}
                                          ]
                           }
            }
```

Figure 6: Typed Data example

4.4 Meta Translation Language

The most complex functionality performed by the module is the bidirectional mapping from configuration represented under the form of a XML document to a series of CLI commands. The difficulties arise from the two conceptually different semantics and from our objective to provide a module easily adaptable to any type of command line interface. We designed a full-fledged language, called Meta Translation Language. This language is expressed in XML and provides a series of constructs capable to drive the XML to CLI mapping. We are not presenting at this section the complete language, but rather restrict to the basics needed to understand the examples, for more information check section 5. The reader should note that the Meta Translation Language has a full support for conditional expressions, loops and dynamic types. Its main purpose is to drive the translation from XML to CLI and vice-versa.

For illustration purposes, consider the structure shown in Figure 12.

Note that main node and all the nodes that belonging to the `<children>` nodes stands for the real structure of the desire XML document. Where `<children>`

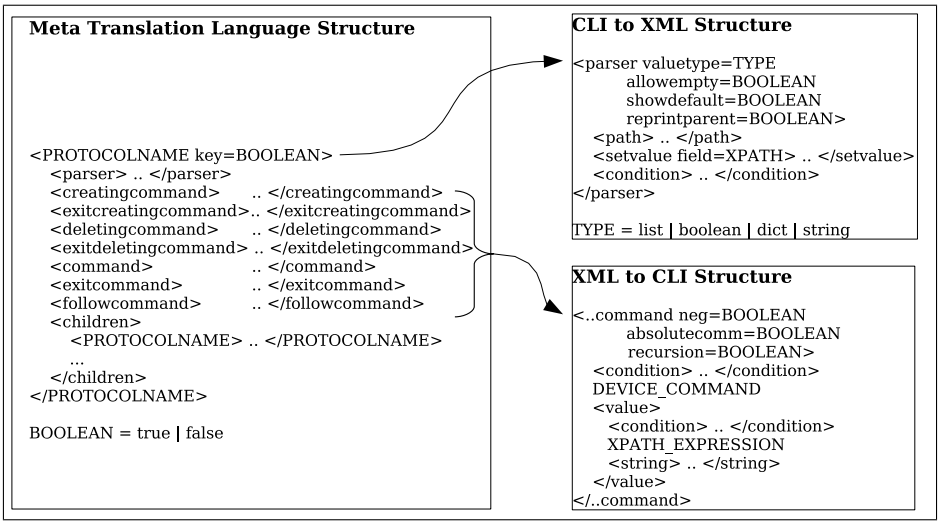

Figure 7: Meta Translation Language

avoid ambiguity with the desire XML nodes with the *instruction* of the XML Meta Translation Language nodes (i.e. `<command>`, `<parser>`,etc).

Every *instruction* node could contain a `<condition>` node which will be used in the translation process depending in its boolean value.

Tags can have a "key" attribute which allows an edit-config operation with an "operation" attribute in it if is set to false.

In the following, we will describe the language elements required for this mapping.

4.4.1 Language Elements for the CLI-to-XML mapping

The `<parser>` node (Figure 12 *CLI to XML Structure*) is used to map the **typedData** structure with the target XML document. Its structure can be specified as many time as needed in each "PROTOCOLNAME" node.

- **valuetype** attribute: is used to specify which is the type expected in the **typedData**. The default value is "dict" for the nodes that have a `<children>` node, otherwise is "string".
- `<condition>` node: a boolean value used in conditional expressions.

4.4.2 Language Elements for the XML-to-CLI mapping

The commands nodes are the essential Meta Translation language elements to create the CLI command from a XML request.

- `<device-command>` node : this stands for a string that is the actual CLI command to generate. In order to add dynamic data to it, this string follows the definition of the String Formatting Operations from the Python [7] language. The values passed to this string will be the ones obtained

from the `<value>` node. To adapt the module to another vendor specific command line interface is easily possible by changing values of this node.

- `<value>` node: generates a string that will be passed to the `<device-command>` node. The value should be a XPath expression scoped over the target XML document.

The overall evaluation of a **command** node is show in a Chart Flow at Figure 8.

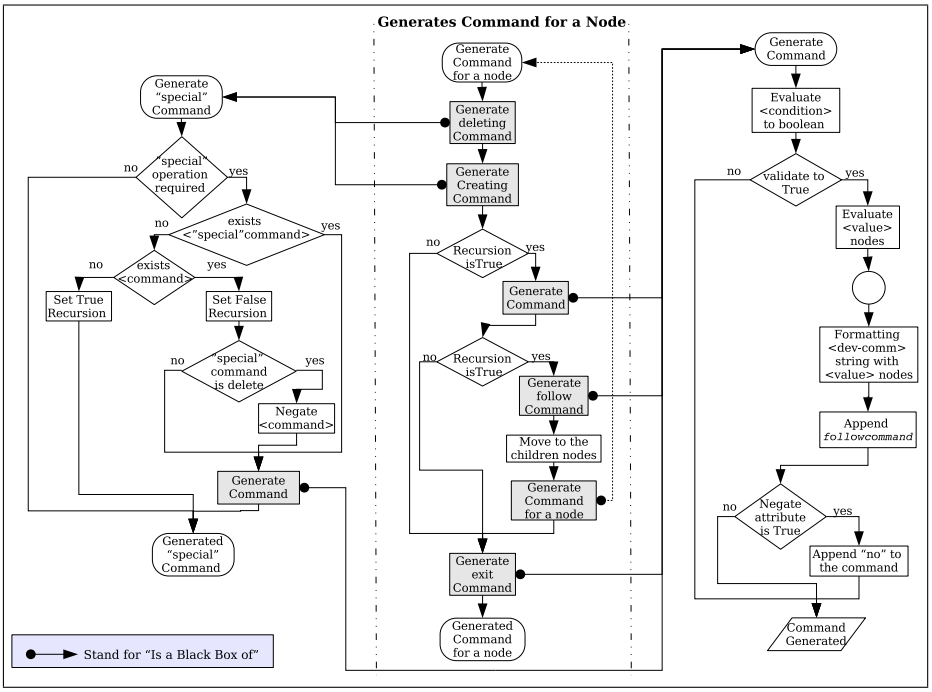

Figure 8: Command Generation process for a node

The main features of this processing structure are .

- The possibility to generates commands **recursively**, which allows the creation or deletion of an XML node extracting information of its scope to process the necessary request. To illustrate this, consider two different request, 1) operates over a neighbor and 2) delete the router configuration. For the first request, as we pointed in the `<command>` node explanation, it is necessary to creates the *envelop* of the `<bgprouter>` node, then move inside its children, continue the process until the *envelop* is done and at that point generates the command for the request. In the second request, as deleting the router configuration takes no more than a **no router bgp *A*** command, it is not necessary to parse into `<bgprouter>` scope, so the recursively is specify to stop at that point.
- The **conditional statement** provided by the `<condition>` nodes, which allow the generation or not of a command depending in the context values of the existing device configuration

- the easily **adaptation** of the module to another vendor specific command line interface by changing the text values of the `<dev-comm>` nodes.

There are several types of command language elements. To illustrate the main concepts of them, consider the follow CLI Commands' example, Figure 9, which attempts to replace all the bgp configuration for one that specify a neighbor and set as passive.

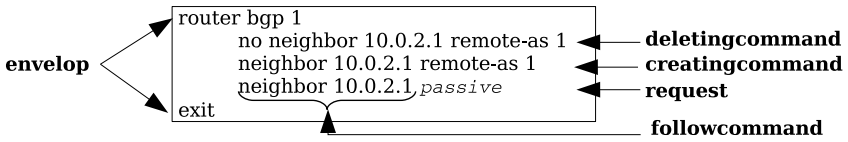

Figure 9: CLI Commands' Request

- **envelop**: used to reach the context place of the XML document. It is done by the `<command>` and `<exitcommand>` nodes, an example of them is show in the *Meta Language Translation* Figure 11, from line 2 to 5 and 6 to 8, respectively.

- **deletingcommand (creatingcommand)** : used to generate the command when edit-config requests to delete (add) a node in the configuration document. Consider the `<deletingcommand>` and `<creatingcommand>` that belongs to the `<neighbor>` node from line 14 to 20 and 21 to 27, in the *Meta Translation Language* Figure 11.

  The request issues to delete the neighbor for which a command like **no neighbor A.B.C.D remote-as E** should be generated. The process, at the `<deletingcommand>` node, will extracts the string value from its `<dev-comm>` node and formats it with the xpath expression values taken by its `<value>` nodes. A "no" will be pre-appended to the command because the *negate* attribute is set at the node.

  Next, it request to creates a new neighbor, by using the `<creatingcommand>` node, in the same way as above.

- **followcommand**: generates a CLI command that will be passed to each of the children nodes of the current node. These children nodes are responsible to append it to the beginning of their commands. Consider the example from the *Meta Translation Language*, that also belongs to the `<neighbor>` node from line 21 to 24 which will generate a command like **neighbor A.B.C.D**.

- **request** : Finally the `<passive>` node is reach in the *Meta Translation Language* Figure 11 at line 28. It process its commands, which will be **passive** and append it to the *followcommand* received which is **neighbor 10.0.2.1** and that way generates the desired request command.

For illustration purposes, the language could be separated in two 1) that provides the information to construct the XML document from the *typedData* (show in *Meta Translation Language* at Figure 10) and 2) the information need

to interpret the XML request and creates the CLI commands (show in *Meta Translation Language* at Figure 11)

4.5 XML Protocol Translator

This entity performs the transformation from BGP CLI commands into the XML document and vise-versa. In order to do that it provides two functions:

- **constructXML:** Once a CLI command series is parsed to a **typedData** structure, the **Meta Transformation Language** is used to create the XML configuration document. In order to do it, it considers all the nodes from the **Meta Translation Language** aligned with the equivalent records in the **typedData**. The alignment is done using the `<parser>` node. A node will be added just if the associated condition is evaluated to true.
- **interpretXMLRequest:** XML encoded NetConf requests (for instance "edit-config") and the **Meta Translation Language** are used to generate the corresponding CLI commands. A search in the XML configuration document is performed to identify the place where the request will be applied. Once the search is done, the command envelope and the request commands are generated as follows.
  - **generating the commands envelope:** The identified XML place and the underlying **Meta Translation Language** structure are used to reach the location within the configuration required by the request. In order to do this, the nodes from the identified XML place, are explored and processed with the `<command>`, `<followcommand>` and `<exitcommand>` defined by the **Meta Translation Language** structure.
  - **applying the request:** this process is similar to the **command envelope** creation, but the exploration goes through the request action node rather than the identified place and the `<creatingcommand>` and/or `<deletingcommand>` are also processed if the specified operation requires it. By default, the `<command>` is considered as explained in subsection 5.2.

4.5.1 CLI to XML example

To illustrate the translation process, let us consider the **typedData** and the **Meta Translation Language** (shown in Figure 10).

Since the `<bgprouter>` doesn't have a `<parser>` node associated, the default operation is to check if the dictionary, **typedData**, has "bgprouter" as key, and then to 1) proceed to the creation of the empty node, 2) change the relative position of the dictionary to the value of "bgprouter" and 3) move the `<children>` node of the Meta Translation Language. Next, it performs the same as above for all children of the node. In case of the `<as-number>` node, it adds the value of the key element "as-number", "1", because the "valuetype" is by default "string" (*Step 1* at Figure 10)..

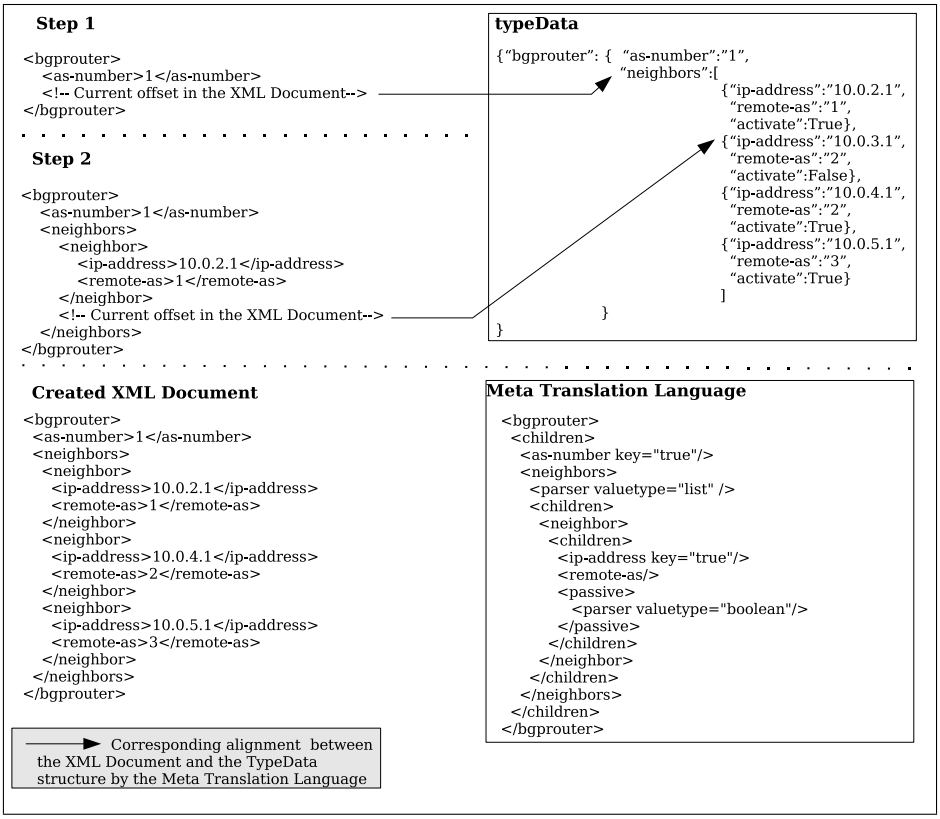

Figure 10: CLI to XML process construction

When the process reaches the `<neighbors>` node, it creates it, and because its "valuetype" is "list", for each value in the list associated with the key "neighbors", it will do as follows: because `<neighbor>` has a `<condition>` node in the `<parser>` node, it checks the boolean value in the dictionary with key "activate", if this key is set to true, it creates the `<neighbor>` node. Finally, it creates the nodes `<ip-address>` and `<remote-as>` with the respective values taken from the dictionary passed by "neighbors". Note that for the case of a node of "valuetype" equal to "boolean" it creates the node just if the value is set to true. (*Step 2* at Figure 10)..

For the rest of the elements in the list it proceeds in the same way, such that the resulting XML document looks like the one illustrated in Figure 10 (*Created XML Document*).

4.5.2 XML to CLI Command example

Supposing the device is configured as the *Existing XML Document* and the edit-config request is as shown in *XML Request* in Figure 11.

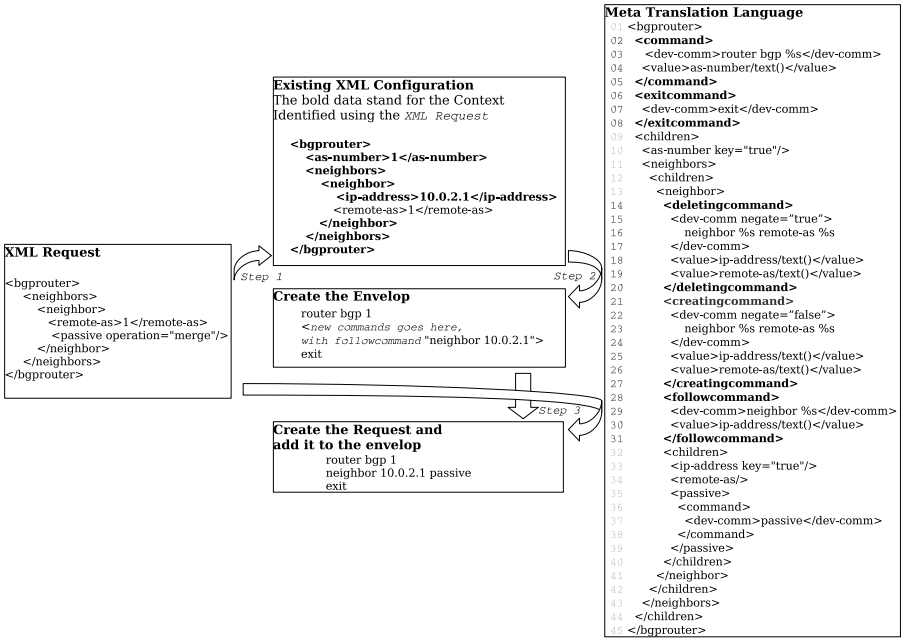

Figure 11: XML to CLI process interpretation

All the nodes from the configuration *Existing XML document*, concerned by this operation, are identified (bold text from Figure 11 in *Existing XML Document*). In order to do this, a XPath expression from the edit-config XML document is created, ignoring however, the node with "operation" attribute as well as all of its descendants. This XPath is in our case : `/bgprouter/neighbors/neighbor[remote-as = '1']`. The matching node is the `<neighbor>` node shown in *Step 1*.

Next, the **command envelop** is generated. The `<bgprouter>` node, (according to the Meta Translation Language), generates two commands, one for

entering into the context (obtained from the `<command>`), "router bgp 1", and the second for the corresponding leave (obtained from the `<exitcommand>`), "exit". Note that the value "1" from the entering command was extracted by the `<as-number>` node value from the node matching the XPath expression mentioned above. Next, the Meta Translation Language is used to deep search the descendants, until the `<neighbor>` is found. Once found, a follow command is executed. The follow command will be "neighbor 10.0.2.1", and the "10.0.2.1" value is obtained by the node matching the XPath expression above. The list of commands at this point is as show in Figure 11 *Step 2*.

Once the command envelop is created, the **applyRequest** is performed. Since the "operation" attribute in the "edit-config" requests is "merge", only the `<command>` nodes are processed, such that, a "passive" command is resulted. The obtained translation is in Figure 11 *Step 3*.

5 Meta Translation Language

The **Meta Translation Language** was designed to used as an structure to storage information which will be used to performs the bidirectional mapping from the configuration represented under the form of a XML document to a series of CLI commands. The difficulties arise from the two conceptually different semantics and from our objective to provide a module easily adaptable to any command line type of interface. For easy readability this language is expressed in XML and provides a series of constructs capable to drive generic XML to CLI mapping. As it will be show in this section, it provides a full support for conditional expressions, loops and dynamic types.

For illustration purposes consider the structure show in Figure 12.

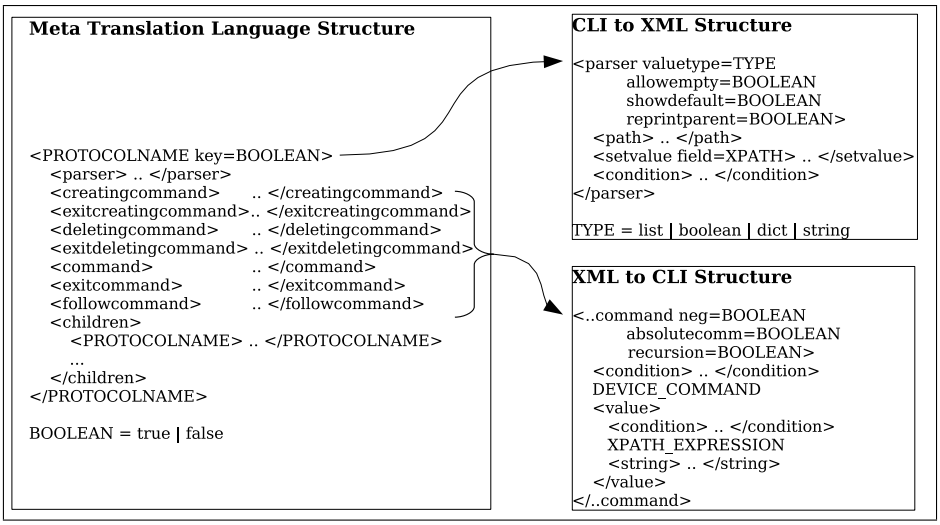

Figure 12: Meta Translation Language

The main node and all the nodes that belongs to the `<children>` nodes (i.e. all the "PROTOCOLNAME" nodes) stands for the real structure of the desire XML document, where "PROTOCOLNAME" should be change by the real

node name for the XML configuration document. Note that the `<children>` node avoid ambiguities with the name of the desire XML nodes and the *instruction* nodes of the XML Meta Translation Language nodes (i.e. `<command>`, `<parser>`,etc).

Tags can have a "key" attribute. In a edit-config type request a search has to be apply to find the context outside the node with the "operation" atributte and other inside it. For the first every node is specified as a matching criteria node but for the second search, some node specify the new propities that are desire to apply in the configuration, in orden to distinguish them it is why this "key" attibute is specified. If this attribute is set to "true", no edit-config operation with an "operation" attribute in this node is possible. Note that does not make sense to erase, replace or create one node that is suppose to be a key inside an allready existing structure.

In the following we will describe the language elements required for this mapping.

5.1 Language Elements for the CLI-to-XML mapping

The `<parser>` node (show in Figure 12 at *CLI to XML Structure*) is the data used to map the **typedData** structure with the target XML document. It structure can be specified as many time as needed in each "PROTOCOLNAME" node. It consist of the following nodes and attributes.

- **valuetype** attribute: is used to specify which is the type expected in the **typedData** structure. The possible values are "dict", "list", "boolean", "string". The default one is "dict" for the nodes that have a `<children>` node, otherwise it is "string".
- **allowempty** attribute: is used to allow this node to be created even if no node or text is in it. The default value is false for the nodes that have a `<children>` node, otherwise true.
- **showdefault** attribute: is used to specified that if the data wasn't found in the **typedData** structure it should or not aggregate this node as an empty one. The default value is false.
- **reprintparent** attribute: is used for reprint the node for each of the children that it generates. It only makes sense in a node which "typevalue" is equal to "list". The default value is false.
- `<path>` node: is used to change the relative path to the desire one in order to keep the equivalence between the **typedData** and the **Meta Translation Language**. Its formats is the same as the one for Unix folders. This node can be specified as many time and in any position as needed. Note that the nodes specified after it, will has the path offset according to the new set up.
- `<setvalue>` node: is used to set a specific static string value to a specific field. This field should be specified as a value for the "field" attribute contained by it node, which value should be a xpath expression scoped over the target XML document. The static string is the text contained by this node. This node also can be specified as many time and in any position as needed.

- `<condition>` node: representing a logical condition, the result is of boolean type, see detailed description at section 5.3 later. If the boolean value validate to false, the evaluation of the `<parser>` node, in this case, stop, rollback all the changes made by this node, if any, and look for the next node. Also can be specified as many time and in any position as needed.

In the case that the `<parser>` node is not specified, the default behavior will be to look over the **typeData** structure at the current offset if the "PROTOCOLNAME" node exists as a key, if it exists it will change the path to the value of this key as is showed next.

```
<parser>
   <condition>
      <exist-path>PROTOCOLNAME</exist-path>
   </condition>
   <path>PROTOCOLNAME</path>
</parser>
```

5.2 Language Elements for the XML-to-CLI mapping

The main purpose of the XML to CLI mapping is the generation of CLI commands semantically equivalent to a XML configuration settings. Our approach consists in representing in an XML document the structure of the CLI language with additional anchors to the generic pertinent XML configuration settings. The command node(Figure 12 *XML to CLI Structure*) is the essential Meta Translation language element.

- **neg** attribute: is use for negate the generated command in case is set to true. It becomes a very important component because for several reasons 1) if the command is composed of a followcommand (as will be explain later) it will concatenate the followcommand and the command and the only way to negate the command will be at the beginning of it rather that at the point of the concatenation 2) a negated command that becomes negated (also could occurs for the followcommand) because of obvious logic it shouldn't be negated at all, so it is why it is important to distinguish the real command from its negation. Note that when the command is generated it will be concatenated to the "no" string if the negation is set to true.

- **absolutecomm** attribute: is used to ignored the followcommand if is set to true. It is usefull when the command generated it is the absolute command.

- **recursion** attribute: once the command at the specific node were generated, it will or won't generates the commands from its children scope according to this value. The default value depends in the kind of command. For the **command** it default is true, for the following command that attribute is ignored and for the rest, **deleting** and **creating** commands its default value is false.

- `<condition>` node: representing a logical condition, the result is of boolean type, see detailed description at section 5.3 later. If the boolean

value validate to false, the evaluation of the `<..command>` node, in this case, stop, rollback all the changes made by this node, if any, and look for the next node. Also can be specified as many time and in any position as needed.

- **DEVICE_COMMAND** text: this stands for a string that is the actual CLI command to generate. Note that this stand as a mixed node which could be a little complicated, the reason of this is to simplify the averages commands. In order to add dynamic data to it, this string follows the definition of the *String Formatting Operations* from the Python [7] language. The values passed to this string will be the ones obtained from the `<value>` node. Note that to adapt the module to another vendor specific command line interface is easily possible by changing values of this node.

- `<value>` node: generates a string that will be passed to the **DEVICE_COMMAND** text as explained above. It can be specified as many time as needed but note that the order in which they are defined will be the same as the one used for generate the final command string as specified by the *String Formatting Operations* from Python. It is a mixed node, which could contains any of these following 2 different elements and as many time as needed plus a `<condition>` node behaving which in case is validate to false it will return an empty string otherwise the string generated by the follow elements.

  - **xpath expression** text: The value should be a XPath expression scoped over the target XML document. The XPath evaluation will produce a string. As could be possible that more than one node match the xpath expression, the resulting list of node will be concatenated to generates only one string. Note that in the case a node matches the expression its name tag will be the value appended otherwise, in case of a text node, its value will be the one appended.
  - `<string>` node: it will be an static string. It is usefull in the case of some static string should be appended but only if the condition evaluates to true.

  Note that for each `<value>` node will be generated one string, so the resulting string for each node will be the concatenation of the XPATH_EXPRESSIONS and the `<string>` node values in the order they appear generating the final string.

As Figure 12 (*XML to CLI Structure*) shows, there are several types of command language elements :

- **creatingcommand** : used to generate the command when the edit-config requests to add a new node in the configuration document (i.e. for creating, replace and some times for merge operations) .

- **deletingcommand** : similar to the previous command, but used to delete a node in the configuration (i.e. for delete and replace operations).

- **command**: used to create an additional command envelop in order to reach the equivalent context of the XML document. If creating or deleting

commands are not specified, then by default a command node will be processed but the recursion is set to false. In case of a *deletingcommand*, it will negate the **neg** attribute.

- **exitcommands** : required to generate final CLI commands in the case of creating, deleting and respectively commands.
- **followcommand**: generates a CLI command that will be passed to each of the children nodes of the current node. These children nodes are responsible to append it to the beginning of their commands. Note that this command won't be added to the list of commands generated.

There is not default behavior for the `<commands>` nodes.

5.3 Conditional Language Elements

The `<condition>` node is defined to provides the ability to establish condition for generates the target *getconfig* XML document or the *CLI* device oriented commands. It will evaluates to a boolean value, this evaluation will rely in the **typeData** structure (in the case of **getconfig**) or in the XML device configuration and the XML request (for the case of **editconfig** related requests). As soon as the first `<condition>` node evaluate to false, it will stop evaluating the node containing this condition and proceeds to the next sibling of it. The structure of this node is show in Figure 13 and can contain any of the "boolean values" nodes, but just one of it at the top level.

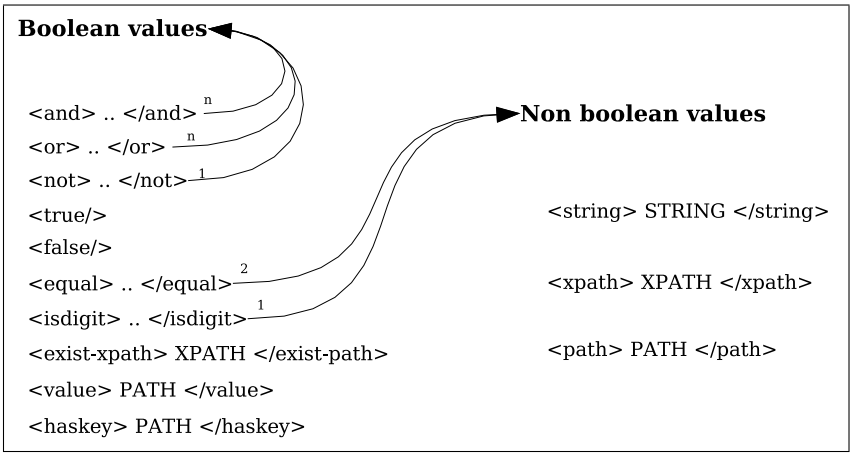

Figure 13: Meta Translation Language Condition

- `<and>` and `<or>`: can contain as many "boolean values" nodes inside it as needed.
- `<not>`: It should contain one of the "boolean values" node inside it.
- `<equal>`: It should contain two "non boolean values" nodes inside it.
- `<isdigit>`: It should contain one "non boolean values" node inside it.

- `<exist-xpath>`: It is only for conditions inside the `<commands>` nodes. It should contain one xpath expression that will be apply to the target document. Note that the current node where it is place in the target structure, is also the current node in the xpath.
- `<value>`: It is only for conditions inside the `<parser>` nodes. It should contain a valid path expression.
- `<haskey>`: It is only for conditions inside the `<parser>` nodes. It should contain a valid path expression.
- `<string>`: It is an string that doesn't need to be evaluated.
- `<xpath>`: It is only for conditions inside the `<commands>` nodes. It should contain a xpath expression, and it will evaluated in the XML target document.
- `<path>`: It is only for conditions inside the `<parser>` nodes. It should contain a path expression, and it will evaluated in **typedData**.

6 To do

There is something that we consider that should be added to the **XML Protocol Translator** library in order to provides more features than just the ones need for the BGP protocol and to make other modules for different protocol their implementation easier.

6.1 Attributes in the Meta Translation Language

As this library was created by the need of the BGP protocol, and our XML design document for it doesn't include any attribute, it wasn't necessary to implement them, even so, when the *Meta Translation Language* was designed this needs were considered. The changes need it to apply for the *Meta Translation Language* are.

`<attribute>` node: As for every "PROTOCOLNAME" there is a field specifying the children, the `<children>` node, also the attributes can be specified as a sibling of it. Of course if no `<attribute>` node is specified, means that this node doesn't have any attribute. Any children of it node should follow the same patter of the children for the `<children>` node, except that an attribute can not have any attribute neither children hanging by it, so this nodes shouldn't be specified.

`<parser>` node: It will be need an extra attribute in this node to specify where the data from the attribute in the **typedData** structure should be found. We propose the attribute "attributetype" which could have any of the following values.

- mixed: It will find the values in the same dictionary as the one used for the children. Note that could be problem if a children and an attribute have the same name, and they don't specify parser option for it. This should be the default value for this attribute.

- duple: it will separate the **typedData** for this node in two, the first object of the duple for attribute uses and the second for children.
- dict: It will be a dictionary containing 2 values 1) "attribute" which will specify the data for the attributes and 2) "children" for the children.

6.2 Classes generator

As the library has to load the XML **Meta Language Translation** in order to create or interpret a request and also to goes through a grand portion of it structure to decide which actions to take, it will be a great improve in performance that from the XML **Meta Language Translation** it generates classes according to it. Note that this follow the same ideas as a parser generator for example.

6.3 Parser Autogeneration

One of the weakness of this library that we experience through the implementation of the BGP Module is the connection between the **typedData** and the **Meta Translation Language**. This problem could be solved by autogenerating the CLI command parser by the information of the **Meta Translation Language**, which will require to add some new concepts to it.

A BGP XML Configuration

The design of the BGP XML Document for the configuration follows almost the same patterns as the stablish by the BGP CLI configuration of Quagga. One of the main changes is as the BGP protocol uses the IPv4-Unicast casting by default, which to our belives, could be quite confusing whenever it want to be set or not. To solve this problem our design first specify the main propieties, those for example that are unique even for different address families, and in each address family it will specify which are the criterias, turnning the configuration, as we said to our belives, more understandible.

A.1 Main Structure of the BGP Protocol

- **Use**

```
<bgp>
   <bgprouter> .. </bgprouter>
   <filters> .. </filters>
   <communities> .. </communities>
</bgp>
```

- **Description** Root structure of the BGP XML Document configuration.
- **Contents**

  `<bgprouter>` – It is the configuration of the router look in section A.2.1.

  `<filters>` – It is the configuration of the filters look in section A.3.

  `<communities>` – It is the configuration of the communities look in section A.4.

A.2 BGP Router Configuration

A.2.1 bgprouter

- **Use**

```
<bgp>
   <bgprouter>
      <as-number> asn </as-number>
      <fast-external-failover/>
      <id> id </id>
      <log-neighbor-changes/>
      <always-compare-med/>
      <local-preference> value </local-preference>
      <client-to-client-reflection/>
      <cluster-id> cluster-id </cluster-id>
      <confederation-id> confederation-id </confederation-id>
      <confederation-peers>...<confederation-peers>
      <enforce-first-as/>
      <deterministic-med/>
      <bestpath>
         <as-path-ignore/>
         <compare-routerid/>
         <med>
            <confed/>
            <missing-as-worst/>
         </med>
```

```
      </bestpath>
      <network-import-check/>
      <scan-time> value </scan-time>
      <dampening>
         <half-life> value </half-life>
         <reuse> value </reuse>
         <suppress> value </suppress>
         <max-suppress-time> value </max-suppress-time>
      </dampening>
      <peer-groups>...</peer-groups>
      <neighbors>...</neighbors>
      <address-families>...</address-families>
      <distance-bgprouter/>
         <external> external-distance </external-routes>
         <internal> internal-distance </internal-routes>
         <local> local-distance </local-routes>
      </distance-bgprouter/>
      <distances>...</distances>
      <no-synchronization/>
      <no-auto-summary/>
   </bgprouter/>
</bgp>
```

- **Description** It describes the main features of the configuration for the BGP router.
- **Contents**

  `<as-number>` – Autonomous System Number
  `router bgp <1-65535>`

  `<id>` – Router Identification. Optional
  `bgp router-id A.B.C.D`

  `<distance-bgprouter>` – Change the distances value of BGP. Optional
  `distance bgp <1-255> <1-255> <1-255>`
  - `<external>` – External Routes range from 1 to 255
  - `<internal>` – Internal Routes range from 1 to 255
  - `<local>` – Local Routes range from 1 to 255

  `<distances>` – Define a distance to particular network Group. Apendix A.2.2.

  `<network-import-check/>` – Checks if the BGP network route is reachable through IGP or not
  `bgp network import-check`

  `<log-neighbor-changes/>` – Enable logging of BGP neighbor
  `bgp log-neighbor-changes`

  `<peer-groups>` – Define the Peers-Group Group. Optional. Appendix A.2.15.

  `<neighbors>` – Define the peer neighbors Group. Optional. Appendix A.2.16.

  `<address-families>` – Define a group of address families. Optional. Appendix A.2.3.

  `<cluster-id>` – Cluster Id for multiple route reflectors. Optional
  `bgp cluster-id A.B.C.D`
  `bgp cluster-id <1-4294967295>`

`<no-synchronization/>` – Disable the IGP synchronization in BGP
`no synchronization`

`<no-auto-summary/>` – Disable automatic network number summarization
`no auto-summary`

`<local-preference>` – Set the default local preference value. Optional
`bgp default local-preference <0-4294967295>`

`<deterministic-med/>` – Enforce the deterministic comparison of the MED
`bgp deterministic-med`

`<always-compare-med/>` – Always compare MED values
`bgp always-compare-med`

`<bestpath>` – Set the best path selection process. Optional

`<as-path-ignore/>` – Ignore the AS-path
`bgp best-path as-path ignore`

`<compare-routerid/>` – the route with the lowest router ID is selected when routes comparation are similiar
`bgp best-path compare-routerid`

`<med>` –
`bgp best-path med [confed] [missing-as-worst]`

`<confed/>` – Compares MED paths from confederation peers

`<missing-as-worst/>` – A path without a MED value turn to be the worst

`<client-to-client-reflection/>` – Set route reflection from a BGP router reflector to clients. Optional
`bgp client-to-client reflection`

`<confederation-id>` – Confederation identifier. Optional
`bgp confederation identifier <1-65535>`

`<confederation-peers>` – Define a Group of peers that belong to the same confederation. Appendix A.2.14.

`<dampening>` – Allow dampening or change various BGP route dampening factors. Optional
`bgp dampening [<1-45> [<1-20000> <1-20000> <1-255>]]`

`<half-life>` – Time in minutes after which a penalty is decreased. Optional

`<reuse>` – Reuse values based on accumulated penalties. Optional together with suppress and max-suppress-time

`<suppress>` – Suppres the route when its penalty exceeds this limit. Optional together with reuse and max-suppress-time

`<max-suppress-time>` – Maximum time in minutes a route can be suppre- ssed. Optional together with reuse and suppress

`<enforce-first-as/>` – Forces to compare the remote AS number of an external peer with the first AS
`bgp enforce-first-as`

`<fast-external-failover/>` – Immediately reset the BGP sessions of any peers if the link goes down
`bgp fast-external-failover`

`<scan-time>` – Set the time for the BGP scanner
`bgp scan-time <5-60>`

A.2.2 distance

- **Use**

```
<bgp>
   <bgprouter>
      <distances>
         <distance>
            <ip> ip-address </ip>
            <mask> mask </mask>
            <value> distance-value </value>
            <word> word </word>
         </distance>
      </distances>
   </bgprouter/>
</bgp>
```

- **Description**
Set the distance for the specified value

`distance <1-255> A.B.C.D/M [WORD] )`

- **Contents**

`<ip>` – IPv4 address

`<mask>` – Length of the mask

`<value>` – Define the distance value

`<word>` – The name of the access list to be applied to the selected routes . Optional

A.2.3 address-families

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv4-address-family> .. </ipv4-address-family>
         <vpnv4-address-family> .. </vpnv4-address-family>
         <ipv6-address-family> .. </ipv6-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
Specify the setting for the casting advertisiment.

- **Contents**

`<ipv4-address-family>` – Related to the IPv4 unicast and multicast family. See Appendix A.2.4.

`<vpnv4-address-family>` – Related to the VPnv4 unicast and multicast family. See Appendix A.2.5.

`<ipv6-address-family>` – Related to the IPv6 unicast family. See Appendix A.2.6.

A.2.4 ipv4-address-family

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv4-address-family>
            <type> type-choice </type>
            <networks> ... </networks>
            <aggregate-addresses> ... </aggregate-addresses>
            <redistributes> ... </redistributes>
            <groups> ... </groups>
            <neighbors> ... </neighbors>
         </ipv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Defines a ipv4 address family

  `address-family ipv4 [unicast|multicast]`

- **Contents**

  `<type>` –
  - unicast :
  - multicast : enables BGP to carry IP multicast routes

  `<networks>` – Defines the announcement Network Group. See Appendix A.2.7. Optional

  `<aggregate-addresses>` – Defines the aggregate address Group. See Appendix A.2.10. Optional

  `<redistributes>` – Defines the redistribute Group. Optional. See Appendix A.2.4. This field MUST NOT BE specified for IPv4 address family of type Multicast

  `<groups>` – Defines a group of Group-peers. Optional. The fields are exactly the same as neighbor(Casting) in Appendix A.2.17.

  `<neighbors>` – Defines the group of neighbor. Optional. Appendix A.2.17 If a neighbor is specified means that this is activated in the address-family.
  `neighbor (A.B.C.D|X:X::X:X|GROUP) activate`
  Also the neighbor must be specified in the group of neighbors in the bgp configuration. Appendix A.2.16

A.2.5 vpnv4-address-family

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <vpnv4-address-family>
            <type> type-choice </type>
            <networks> ... </networks>
            <neighbors> ... </neighbors>
         </vpnv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Provides IPv4 VPN services via an MPLS backbone

  `address-family vpn4 unicast`

- **Contents**

  `<type>` –
  – unicast :

  `<networks>` – Define the group of network. Appendix A.2.8. Optional

  `<groups>` – Define a group of Group-peers. Optional. The fields are exactly the same as neighbor(Casting) in Appendix A.2.17.

  `<neighbors>` – Define the group of neighbor. Appendix A.2.17 Optional. If a neighbor is specified means that this is activated in the address-family.
  `neighbor (A.B.C.D|X:X::X:X|GROUP) activate` Also the neighbor must be specified in the group of neighbors in the bgp configuration as in Appendix A.2.16.

A.2.6 ipv6-address-family

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv6-address-family>
            <type> type-choice </type>
            <networks> ... </networks>
            <aggregate-addresses> ... </aggregate-addresses>
            <redistributes> ... </redistributes>
            <groups> ... </groups>
            <neighbors> ... </neighbors>
         </ipv6-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Defines a IPv6 address family

  `address-family ipv6 [unicast]`

- **Contents**

  `<type>` –

  – unicast

  `<networks>` – Define the group of network. Appendix A.2.9.Optional

  `<aggregate-addresses>` – Define the group of aggregate-address. Appendix A.2.11. Optional.

  `<redistributes>` – Define the group of redistribute routes. See Appendix A.2.13. Optional.

  `<groups>` – Define a group of Group-peers. Optional. The fields are exactly the same as neighbor(Casting) in Appendix A.2.17.

  `<neighbors>` – Define the group of neighbor. Appendix A.2.17. Optional. If a neighbor is specified means that this is activated in the address-family.
  `neighbor (A.B.C.D|X:X::X:X|GROUP) activate`
  Also the neighbor must be specified in the group of neighbors in the bgp configuration. See neighbor (Description) in Appendix A.2.16

A.2.7 network (IPv4)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv4-address-family>
            <networks>
               <network>
                  <ip> ip-address </ip>
                  <mask> mask </mask>
                  <backdoor/>
                  <route-map> route-map-tag </route-map>
               </network>
            </networks>
         </ipv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Add the announcement Network

  `network A.B.C.D mask A.B.C.D [backdoor|route-map WORD]`
  `(network A.B.C.D route-map WORD`
  `network A.B.C.D/M [backdoor|route-map WORD]`

- **Contents**

  `<ip>` – IPv4 Address

  `<mask>` – Could be the mask or the length of it

  `<backdoor/>` – Set this network as the backdoor. If the IPv4 address family for which is specified is of type Multicast, this field MUST NOT BE specified.

  `<route-map>` – Set a route-map for this network. Optional

A.2.8 network (VPNv4)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <vpnv4-address-family>
         <networks>
               <network>
                  <ip-prefix> ip </ip-prefix>
                  <mask> mask </mask>
                  <rd> rd-value </rd>
                  <route-map> route-map-tag </route-map>
               </network>
            </networks>
         </vpnv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Add the announcement Network for the VPNv4 protocol

```
address-family vpnv4 [unicast]
network A.B.C.D/M rd ASN_or_IP tag WORD
exit-address-family
```

- **Contents**

  `<ip-prefix>` – IP

  `<mask>` – Could be the mask or the length of it

  `<rd>` – AS number or IP address

  `<route-map>` – Set a route-map for this network. Optional

A.2.9 network (IPv6)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv6-address-family>
            <networks>
               <network>
                  <ip-prefix> ip </ip-prefix>
                  <mask> mask </mask>
                  <route-map> route-map-tag </route-map>
               </network>
            </networks>
         </ipv6-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Announcements Network address for IPv6 protocol

```
address-family ipv6 [unicast]
network X:X::X:X/M [route-map WORD]
exit-address-family
```
Backward compatibility.

```
ipv6 bgp network X:X::X:X/M
```

- **Contents**

`<ip-prefix>` – IPv6

`<mask>` – It is the length of the mask

`<route-map>` – Set a route-map for this network. Optional

A.2.10 aggregate-address (IPv4)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv4-address-family>
            <aggregate-addresses>
               <aggregate-address>
                  <ip-prefix> ip </ip-prefix>
                  <mask> mask </mask>
                  <as-set/>
                  <summary-only/>
               </aggregate-address>
            </aggregate-addresses>
         </ipv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
Specifies an aggregate address

```
aggregate-address A.B.C.D/M [as-set|summary-only] [as-set|summary-only]
aggregate-address A.B.C.D A.B.C.D [as-set|summary-only] [as-set|summary-only]
```

- **Contents**

`<ip-prefix>` – IP

`<mask>` – Could be the mask or the lenght of it.

`<as-set/>` – Resulting routes includes AS set.

`<summary-only/>` – Aggregated routes will not be announce.

A.2.11 aggregate-address (IPv6)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv6-address-family>
            <aggregate-addresses>
```

```
            <aggregate-address>
               <ip-prefix> ipv6 </ip-prefix>
               <mask> ipv6-mask </mask>
               <summary-only/>
            </aggregate-address>
         </aggregate-addresses>
      </ipv6-address-family>
   </address-families>
</bgprouter/>
</bgp>
```

- **Description**
  Specifies an aggregate address for IPv6 protocol

  `address-family ipv6 [unicast]`
  `aggregate-address X:X::X:X/M [summary-only]`
  `exit-address-family`
  Backward compatibility.

  `ipv6 bgp aggregate-address X:X::X:X/M [summary-only]`

- **Contents**

  `<ip-prefix>` – IPv6

  `<mask>` – It is the lengh of the mask

  `<summary-only/>` – Aggregated routes will not be announce.

A.2.12 redistribute (IPv4-Unicast)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv4-address-family>
            <redistributes>
               <redistribute>
                  <type> type-choice </type>
                  <metric> value </metrix>
                  <route-map> route-map-tag </route-map>
               </redistribute>
            </redistributes>
         </ipv4-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Redistribute the specified choice route to BGP process

  `redistribute (kernel|static|connected|rip| OSPF) [metric <0-4294967295>] [route-map WORD]`

- **Contents**

  `<type>` –

  – kernel

- static
- connected
- rip
- ospf

`<metric>` – Set a metrix value for this redistribution. Optional

`<route-map>` – Set a route-map for this redistribution. Optional

A.2.13 redistribute (IPv6)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <ipv6-address-family>
            <redistributes>
               <redistribute> redistribute-choice </redistribute>
               <metric> value </metrix>
               <route-map> route-map-tag </route-map>
            </redistributes>
         </ipv6-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Redistribute the specified choice route to BGP process for IPv6 protocol

```
address-family ipv6 [uniscast]
redistribute (kernel|static|connected|ripng|ospf6) [metric <0-4294967295>]
[route-map WORD]
exit-address-family
```

- **Contents**

`<redistribute>` – redistribute-choice :
  - kernel
  - static
  - connected
  - ripng
  - ospf6

`<metric>` – Set a metrix value for this redistribution. Optional

`<route-map>` – Set a route-map for this redistribution. Optional

A.2.14 confederation-peer

- **Use**

```
<bgp>
   <bgprouter>
      <confederation-peers>
         <confederation-peer> AS-peer </confederation-peer>
      </confederation-peers>
   </bgprouter>
</bgp/>
```

- **Description**
  Configure the autonomous systems that belong to the confederation

  `bgp confederation peers .<1-65535>`

- **Contents**

  `<confederation-peer>` – Confederation AS peer

A.2.15 peer-group (description)

- **Use**

```
<bgp>
   <bgprouter>
      <peer-groups>
         <peer-group>
            <name> group-name </name>
            <remote-as> asn </remote-as>
            <description> description-text </description>
            <shutdown/>
            <passive/>
            <ebgp-multihop>
               <ttl> value </ttl>
            </ebgp-multihop>
            <enforce-multihop/>
            <update-source> ifname <update-source>
            <timers>
               <keep-alive> value </keep-alive>
               <hold-time> value </hold-time>
            </timers>
            <weight> weight-number </weight>
            <capability>
               <dynamic/>
               <route-refresh/>
            </capability>
            <dont-capability-negotiate/>
            <override-capability/>
         </peer-group>
      </peer-groups>
   </bgprouter/>
</bgp>
```

- **Description**
  Defines a group

  `neighbor NAME peer-group`

- **Contents**

  `<name>` – Name of the group

  `<...>` – See neighbor (description), appendix A.2.16

A.2.16 neighbor (description)

- **Use**

```
<bgp>
   <bgprouter>
      <neighbors>
         <neighbor>
            <ip-address> ip </ip-address>
            <remote-as> asn </remote-as>
            <activated-group> group-name </activated-group>
            <description> description-text </description>
            <shutdown/>
            <port> port </port>
            <interface> ifname </interface>
            <passive/>
            <ebgp-multihop>
               <ttl> value </ttl>
            </ebgp-multihop>
            <enforce-multihop/>
            <update-source> ifname <update-source>
            <version> bgp-version </version>
            <advertisement-interval> value <advertisement-interval/>
            <timers>
               <keep-alive> value </keep-alive>
               <hold-time> value </hold-time>
            </timers>
            <timers-connect> value </timers-connect>
            <weight> weight-number </weight>
            <capability>
               <dynamic/>
               <route-refresh/>
            </capability>
            <dont-capability-negotiate/>
            <strict-capability-match/>
            <override-capability/>
            <transparent-as/>
            <transparent-nexthop/>
         </neighbor>
      <neighbors>
   </bgprouter/>
</bgp>
```

- **Description**
  Defines a neighbor

- **Contents**

  `<ip-address>` – Peer's IP (IPv4, IPv6 or Group name) address
  `neighbor (A.B.C.D|X:X::X:X|GROUP)`

  `<remote-as>` – Peer's AS
  `neighbor (A.B.C.D|X:X::X:X|GROUP) remote-as <1-65535>`

  `<bind-group>` – Bind this group to the peer
  `neighbor (A.B.C.D|X:X::X:X) peer-group GROUP`

  `<description>` – Description of the peer
  `neighbor (A.B.C.D|X:X::X:X|GROUP) description TEXT`

  `<shutdown/>` – Specifies if the peer connection is off.
  `neighbor (A.B.C.D|X:X::X:X|GROUP) shutdown`

  `<version>` – Peer's BGP version
  `neighbor (A.B.C.D|X:X::X:X) version (4|4-)`

`<interface>` – When connected to a BGP peer over an IPv6 link-local address
`neighbor (A.B.C.D|X:X::X:X) interface IFNAME`

`<port>` – Specifies the BGP peer's port
`neighbor (A.B.C.D|X:X::X:X) port <0-65535>`

`<update-source>` – Specifies the interface to be used as the source. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) update-source IFNAME`

`<weight>` – Specifies a default weight for the peer's route
`neighbor (A.B.C.D|X:X::X:X|GROUP) weight <0-65535>`

`<enforce-multihop/>` – Enforce eBGP neighbors to perform multihop
`neighbor (A.B.C.D|X:X::X:X|GROUP) enforce-multihop`

`<ebgp-multihop>` – To accept and attempt BGP connections to external peers on networks that are not directly connected
`neighbor (A.B.C.D|X:X::X:X|GROUP) ebgp-multihop [<1-255>]`
`<ttl>` – Time to live in number of hops. Optional

`<strict-capability-match/>` – Strictly compares remote capabilities and local capabilities. It MUST NOT BE specified if the override-capability is.
`neighbor A.B.C.D|X:X::X:X) strict-capability-match`

`<dont-capability-negotiate/>` – Disables sending Capability Negotiation as OPEN message
`neighbor (A.B.C.D|X:X::X:X|GROUP) dont-capability-negotiate`

`<override-capability/>` – Override the result of Capability Negotiation with local configuration. It MUST NOT BE specified if the field strict-capability-match is.
`neighbor (A.B.C.D|X:X::X:X|GROUP) override-capability`

`<capability>` – Control advertisement of BGP capabilities to peers
`<dynamic/>` – Advertise or withdraw an address family capability to a peer without performing a hard clear of the BGP session. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) capability dynamic`
`<route-refresh/>` – Indicates support of route-refresh messages that request the peer to resend its routes to the system. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) capability route-refresh`

`<passive/>` – Sets a BGP neighbor as passive
`neighbor (A.B.C.D|X:X::X:X|GROUP) passive`

`<transparent-as/>` – Specifies not to append the AS path number
`neighbor (A.B.C.D|X:X::X:X) transparent-as`

`<transparent-nexthop/>` – Keep the nexthop value of the route even if the peer is an eBGP peer
`neighbor (A.B.C.D|X:X::X:X) nexthop`

`<advertisement-interval>` – Set the minimum interval between the sending of BGP routing updates
`neighbor (A.B.C.D|X:X::X:X) advertisement-interval <0-600>`

`<timers-connect>` – Specifies the connect timer in seconds
`neighbor (A.B.C.D|X:X::X:X) timers connect <0-65535>`

`<timers>` – Set the timers for a specific BGP peer. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) timers <0-65535> <0-65535>`

`<keep-alive>` – Frequency in seconds with sends keepalive messages to it peer

`<hold-time>` – Interval in seconds after not receiving keepalive messages that the software declares a peer dead

A.2.17 neighbor (Casting)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <ip-address> ip-address </ip-address>
               <group-activated/>
               <capability-orf>
                  <prefix-list> prefix-list-choice </prefix-list>
               </capability-orf>
               <route-reflector-client/>
               <next-hop-self/>
               <remove-private-as/>
               <send-community>
                  <type> type-choice </type>
               </send-community>
               <default-originate>
                  <route-map> route-map-tag </route-map>
               </default-originate>
               <soft-reconfiguration> reconfiguration-choice </soft-reconfiguration>
               <maximum-prefix>
                  <maximum> Maximum number </maximum>
                  <warning-only/>
               </maximum>
               <route-server-client/>
               <allowas-in>
                  <quant> value </quant>
               </allowas-in>
               <bind-filters>...</bind-filters>
               <attribute-unchanged>
                  <as-path/>
                  <next-hop/>
                  <med/>
               </attribute-unchanged>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Defines the specific propities for a neighbor in this address family, where XXXX-address-family is any of the posible families

- **Contents**

`<ip-address>` – Peer's IP (IPv4, IPv6 or Group name) address

`<group-activated/>` – Activate the group for this specifc address family. The group is specified in neighbor(Description)
`neighbor (A.B.C.D|X:X::X:X) peer-group GROUP`

`<next-hop-self/>` – Announce route's nexthop as being equivalent to the address of the BGP router
`neighbor (A.B.C.D|X:X::X:X|GROUP) next-hop-self`

`<capability-ort>` – Support of cooperative route filtering to install a BGP speaker's inbound route filter as an outbound route filter on the peer. Is the address family is vpnv4-address-family, this field MUST NOT BE specified. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) capability ort prefix-list [both|send|receive]`

`<prefix-list>` – both,send or receive

`<route-reflector-client/>` – Specifies that the peer is client of the route reflector sever
`neighbor (A.B.C.D|X:X::X:X|GROUP) route-reflector-client`

`<route-server-client/>` – Specified the peer as route server client
`neighbor (A.B.C.D|X:X::X:X|GROUP) route-server-client`

`<default-originate>` – Announce the default route 0.0.0.0/0. Is the address family is vpnv4-address-family, this field MUST NOT BE specified. Optional
`neighbor (A.B.C.D|X:X::X:X|GROUP) default-originate [route-map ROUTE]`

`<route-map>` – Route map tag. Optional

`<allowas-in>` –
`neighbor (A.B.C.D|X:X::X:X|GROUP) allowas-in <1-10>`

`<quant>` – Quantity of AS of a received route may contain the reci- pient BGP speaker's AS number to be accepted. Optional

`<maximum-prefix>` – how many prefixes can be received from a neighbor
`neighbor (A.B.C.D|X:X::X:X|GROUP) maximum-prefix <1-4294967295> [warning-only]`

`<maximum>` – Maximum number

`<warning-only/>` – Allows the router to generate log message when the maximum is exceeded. Optional

`<remove-private-as/>` – Remove the private AS from the outgoing messages
`neighbor (A.B.C.D|X:X::X:X|GROUP) remove-private-AS`

`<attribute-unchanged>` – Advertise unchanged BGP attributes to the peer.
`neighbor (A.B.C.D|X:X::X:X|GROUP) attribute-unchanged [as-path] [next-hop] [med]`

`<as-path/>` – AS path

`<next-hop/>` – Next hop

`<med/>` – MED

`<send-community>` – Send the community attribute
`neighbor (A.B.C.D|X:X::X:X|GROUP) send-community [both|extended|standard]`
`<type>` – both, extended or standard. Optional

`<bind-filters>` – Define a group of filter to bind to this peer. Appendix A.2.18.

`<soft-reconfiguration>` – To configure the software to start storing updates, there is only one choice :

- inbound: Indicates that the update to be stored is an incoming update.
  `neighbor (A.B.C.D|X:X::X:X|GROUP) soft-reconfiguration inbound`

A.2.18 bind-filters

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <bind-filters>
                  <distribute-list> .. </distribute-list>
                  <prefix-list> .. </prefix-list>
                  <filter-list> .. </filter-list>
                  <router-map> .. </router-map>
                  <unsuppress-map> .. </unsuppress-map>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Specify the filters binded to a neighbor, all the filter in it can be specified as many times as need.

- **Contents**

`<distribute-list>` – Distributes list

`<prefix-list>` – Prefixes list

`<filter-list>` – Filters list

`<router-map>` – Route Map

`<unsuppress-map>` – Unsuppress Map

A.2.19 distribute-list (access-list)

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
```

```
            <neighbor>
               <bind-filters>
                  <distribute-list>
                     <name> name </name>
                     <direct> direct-choice </direct>
                  </distribute-list>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Bind this distribute list(access-list) to the peer

  `neighbor (A.B.C.D|X:X::X:X|GROUP) distribute-list (<1-199>|<1300-2699>|NAME>) (in|out)`

- **Contents**

  `<name>` – Name of the distribute list

  `<direct>` –
  - in : Inbound filter
  - out : Outbound filter

A.2.20 prefix-list

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <bind-filters>
                  <prefix-list>
                     <name> name </name>
                     <direct> direct-choice </direct>
                  </prefix-list>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Bind this prefix list to the peer

  `neighbor (A.B.C.D|X:X::X:X|GROUP) prefix-list NAME (in|out)`

- **Contents**

  `<name>` – Name of the prefix list

  `<direct>` –
  - in : Inbound filter
  - out : Outbound filter

A.2.21 filter-list

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <bind-filters>
                  <filter-list>
                     <name> name </name>
                     <direct> direct-choice </direct>
                  </filter-list>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Bind this filter list to the peer

  `neighbor (A.B.C.D|X:X::X:X|GROUP) filter-list NAME (in|out)`

- **Contents**

  `<name>` – Name of the filter list

  `<direct>` –
  - in : Inbound filter
  - out : Outbound filter

A.2.22 router-map

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <bind-filters>
                  <router-map>
                     <name> name </name>
                     <direct> direct-choice </direct>
                  </router-map>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Bind this filter to the peer

  `neighbor (A.B.C.D|X:X::X:X|GROUP) router-map NAME (in|out)`

- **Contents**

  `<name>` – Name of the router map

  `<direct>` –

  - in : Inbound filter
  - out : Outbound filter

A.2.23 unsuppress-map

- **Use**

```
<bgp>
   <bgprouter>
      <address-families>
         <XXXX-address-family>
            <neighbor>
               <bind-filters>
                  <unsuppress-map>
                     <name> name </name>
                     <direct> direct-choice </direct>
                  </unsuppress-map>
               </bind-filters>
            </neighbor>
         </XXXX-address-family>
      </address-families>
   </bgprouter/>
</bgp>
```

- **Description**
  Bind this unsuppress-map to the peer

  `neighbor (A.B.C.D|X:X::X:X|GROUP) unsuppress-map NAME (in|out)`

- **Contents**

  `<name>` – Name of the unsuppress map

  `<direct>` –

  - in : Inbound filter
  - out : Outbound filter

A.3 Filters

A.3.1 access-list (distribute-list)

- **Use**

```
<bgp>
   <filters>
      <access-list>
         <list-name> name </list-name>
         <remark> line </remark>
         <list>
            <state> state-choice </state>
            <protocol> protocol-choice </protocol>
            <source> source </source>
            <source-wild> source-wildcard </source-wild>
            <type> type-option </type>
```

```
            <destination> destination </destination>
            <dest-wild> destination-wildcard </dest-wild>
          </list>
        </access-list>
    </filters>
</bgp/>
```

- **Description**
  Defines a new AS path access list

  ```
  access-list (<1-2699>|NAME) remark .LINE
  access-list NAME (permit|deny) (A.B.C.D/M [exact-match]| any)
  access-list (<1-99>|<1300-1999>) (permit|deny) SOURCE SOURCE-WILDCARD
  access-list (<100-199>|<2000-2699>) (permit|deny) PROTOCOL SOURCE
  SOURCE-WILDCARD DESTINATION DESTINATION-WILDCARD
  ```

- **Contents**

  `<list-name>` – There are 3 possibilities:
  - NAME : Defines an standard access-list
  - `<1-99>|<1300-1999>` : Defines an standard access-list
  - `<100-199>|<2000-2699>` : Defines an extended access-list

  `<remark>` – Access list remark. Optional Non of the following field MUST BE specified

  `<list>` – Specified for the many different lists.

  `<state>` –
  - deny
  - permit

  `<protocol>` – MUST BE specified only if the list-name if of the type `<100-199>|<2000-2699>`. Just one protocol is defined
  - ip

  `<source>` –
  - A.B.C.D : ip address
  - any : specify a source and mask of 0.0.0.0 255.255.255.255
  - host : followed by the host address A.B.C.D which specifies source-mask of 0.0.0.0. MUST NOT BE specified for list-name of NAME

  `<source-wild>` – depends on the choice of the source field
  - `<0-32> |A.B.C.D` : should be the length of the mask applied to the source. For list-name of NAME it MUST BE the length of the mask. For the other MUST BE and A.B.C.D mask. It is Optional for standard access-list
  - any : this field MUST NOT BE specified
  - host : host address A.B.C.D

  `<type>` – MUST BE specified just for list-name of NAME. Optional Just one type is defined
  - exact-match

`<destination>` – apply the same as explained above for source. MUST NOT BE specified for standard access-list.

`<dest-wild>` – apply the same as explained above for source-wild. MUST NOT BE specified for standard access-list.

A.3.2 ipv6-access-list

- **Use**

```
<bgp>
   <filters>
      <ipv6-access-list>
         <list-name> name </list-name>
         <remark> line </remark>
         <list>
            <state> state-choice </state>
            <source> source </source>
            <mask> mask </mask>
            <type> type-option </type>
         </list>
      </ipv6-access-list>
   </filters>
</bgp/>
```

- **Description**
  Defines a new AS path access list

```
ipv6 access-list NAME remark .LINE
ipv6 access-list NAME (permit|deny) (X:X::X:X/M [exact-match]|
any)
```

- **Contents**

`<list-name>` – name of the access list

`<remark>` – Access list remark. Optional Non of the following field MUST BE specified

`<list>` – Specified for the many different lists.

`<state>` – (deny—permit)

`<source>` –

– X:X::X:X : IPv6 address

– any

`<mask>` – depends on the choice of the source field

– `<0-128>` : should be the length of the mask applied to the source.

– any : this field MUST NOT BE specified

`<type>` – MUST BE specified just if the source is not any. Optional Just one type is defined

– exact-match

A.3.3 as-path (filter-list)

- **Use**

```
<bgp>
   <filters>
      <as-path>
         <name> name </name>
         <state> state-choice </state>
         <regexp> line </regexp>
      </as-path>
   </filters>
</bgp/>
```

- **Description**
  Defines a new AS path access list

  `ip as-path access-list NAME (permit|deny) REGEXP`

- **Contents**

  `<name>` – Name of the access list

  `<state>` –
  - permit
  - deny

  `<regexp>` – Regular expression

A.3.4 prefix-list

- **Use**

```
<bgp>
   <filters>
      <prefix-list>
         <name> name </name>
         <description> description </description>
         <sequences>
            <seq> seq-value </seq>
            <state> state-choice </state>
            <ip-prefix> ip </ip-prefix>
            <mask> mask </mask>
            <ge> greater-eq-value </ge>
            <le> lower-eq-value </le>
         </sequence>
      </prefix-list>
   </filters>
</bgp/>
```

- **Description**
  Defines a Prefix list

  `ip prefix-list NAME [seq <1-4294967295>] (permit|deny) (A.B.C.D/M|any) [ge <0-32>] [le <0-32>]`

  `ip prefix-list NAME description .LINE`

- **Contents**

  `<name>` – Name of the access list

  `<description>` – Description of the access list. Optional

  `<sequences>` – Define a group of prefix list of different sequences

  `<seq>` – Specified the sequence order. Optional

  `<state>` – (permit—deny)

  `<ip-prefix>` –

  – A.B.C.D : ip address

  – any : specify a source and mask of 0.0.0.0 255.255.255.255

  `<mask>` – Length of the Mask. MUST NOT BE specified if ip-prefix is any

  `<ge>` – The greater-than-or-equal-to value of the mask length to be applied to this prefix. MUST NOT BE specified if ip-prefix is any

  `<le>` – The lower-than-or-equal-to value of the mask length to be applied to this prefix. MUST NOT BE specified if ip-prefix is any

A.3.5 ipv6-prefix-list

- **Use**

```
<bgp>
   <filters>
      <ipv6-prefix-list>
         <name> name </name>
         <description> description </description>
         <sequences>
            <seq> seq-value </seq>
            <state> state-choice </state>
            <ip-prefix> ip </ip-prefix>
            <mask> mask </mask>
            <ge> greater-eq-value </ge>
            <le> lower-eq-value </le>
         </sequences>
      </prefix-list>
   </filters>
</bgp/>
```

- **Description**
  Defines a Prefix list

```
ipv6 prefix-list NAME [seq <1-4294967295>] (permit|deny) X:X::X:X/M
[ge <0-128>] [le <0-128>]

ipv6 prefix-list NAME description .LINE
```

- **Contents**
  Same as explained above in the prefix-list

A.3.6 route-map

- **Use**

```
<bgp>
   <filters>
      <route-map>
         <map-tag> tag </map-tag>
         <sequences>
            <seq-number> sequence-number </seq-number>
            <state> state-choice </state>
            <match>...</match>
            <set>...</set>
         </sequences>
      </route-map>
   </filter>
</bgp/>
```

- **Description**
  Specifies a Route Map

  `route-map NAME (permit|deny) <1-65535>`

- **Contents**

  `<map-tag>` – Map Route's Name

  `<sequences>` – Define a group of sequences for the route-map

  `<seq-number>` – Define the Sequence Number

  `<state>` – (permit—deny)

  `<match>` – Specifies the match operation. Optional

  `<set>` – Specifies the set operation. Optional

A.3.7 match

- **Use**

```
<bgp>
   <filters>
      <route-map>
         <sequences>
            <match>
               <metrics>
                  <metric-value> value </metric-value>
               </metrics>
               <origin> origin-choice </origin>
               <as-paths>
                  <as-path-name> as-path-name </as-path-name>
               </as-paths>
               <access-lists>
                  <list-name> name </list-name>
               </access-lists>
               <ipv6-access-lists>
                  <list-name> name </list-name>
               </ipv6-access-lists>
               <prefix-lists>
                  <list-name> name </list-name>
               </prefix-lists>
               <ipv6-prefix-lists>
                  <list-name> name </list-name>
```

```
            </ipv6-prefix-lists>
            <next-hop-access-lists>
               <list-name> name </list-name>
            </next-hop-access-lists>
            <next-hop-prefix-lists>
               <list-name> name </list-name>
            </next-hop-prefix-lists>
            <ipv6-next-hop>
               <ip> ip </ip>
            </ipv6-next-hop>
            <community>
               <name> community-name </name>
               <exact-match/>
            </community>
            <extcommunity> extcommunity-name </extcommunity>
         </match>
      </sequences>
   </route-map>
 </filter>
</bgp/>
```

- **Description**
  Adds matching clauses to the specified route map

- **Contents**

  `<metric>` – Matches a route for the specifed metric value. Quagga supportsonly one
  `match metric <0-4294967295>`
  `<metric-value>` – metric value

  `<origin>` – Matches a route for the specifed origin :
  `match origin (egp|igp|incomplete)`
  - egp
  - igp
  - incomplete

  `<as-path>` – Matches a BGP AS path access list. Quagga supports only one
  `match as-path NAME`
  `<as-path-name>` – AS path name

  `<access-lists>` – Matches any routes that have a destination network number address that is permitted by the access lists. Quagga supports only one
  `match ip address (<1-199>|<1300-2699>|NAME)`
  `<list-name>` – access list name

  `<ipv6-access-lists>` – Same as explained above (access-lists)
  `match ipv6 address NAME`
  `<list-name>` – access list name

  `<prefix-lists>` – Same as access-list but for prefix-list
  `match ip address prefix-list NAME`
  `<list-name>` – prefix list name

`<ipv6-prefix-lists>` – Same as explained above (prefix-list)

`match ipv6 address prefix-list NAME`

`<list-name>` – prefix list name

`<next-hop-access-lists>` – Matches any routes that have a next-hop router address passed by the specified access list. Quagga supports only one.

`match ip next-hop (<1-199>|<1300-2699>|NAME)`

`<list-name>` – prefix list name

`<next-hop-prefix-lists>` – Same as next-hop-access-list but for prefix-list

`match ip next-hop prefix-list NAME`

`<list-name>` – prefix list name

`<ipv6-next-hop>` – Same as next-hop-access-list but for IPv6 address. Quagga supports only one.

`match ipv6 next-hop X:X::X:X`

`<ip>` – IPv6 address

`<community>` – Matches a route for the specifed community :

`match community (<1-199>|NAME) [exact-match]`

`<name>` – name of the community

`<exact-match/>` – match happen only when BGP updates have completely the same communities value specified in the community list

`<extcommunity>` – Matches a route for the specifed extcommunity

`match extcommunity (<1-199>|NAME)`

A.3.8 set

- **Use**

```
<bgp>
   <filters>
      <route-map>
         <sequences>
            <set>
               <metric>
                  <negative/>
                  <value> value </value>
               </metric>
               <weight> value </weight>
               <local-preference> preference-value </local-preference>
               <origin> origin-choice </origin>
               <originator-id> id </originator-id>
               <atomic-aggregate/>
               <aggregator>
                  <router-as> as-path </router-as>
                  <router-id> id </router-id>
               </aggregator>
               <as-path>
                  <prepend> values </prepend>
               </as-path>
               <next-hop> value </next-hop>
               <vpnv4-next-hop> value </vpnV4-next-hop>
               <ipv6-next-hop>
```

```
                <local> IPv6-next-hop-local </local>
                <global> IPv6-next-hop-global </global>
             </ipv6-next-hop>
             <comm-list-del> list-name </comm-list-del>
             <communities>
                <community> community-name </community>
             </communities>
             <extcommunities>
                <extcommunity>
                   <type> type-choice </type>
                   <value> value </value>
                </extcommunity>
             </extcommunities>
          </set>
       </sequences>
    </route-map>
  </filter>
</bgp/>
```

- **Description**
  Adds setting clauses to the specified route map

- **Contents**

  `<metric>` – Sets the route's metric value. Optional
  `set metric (+|-) <0-4294967295>`
  `<negative/>` –
  `<value>` –

  `<weight>` – Sets the route weight. Optional
  `set weight <0-4294967295>`

  `<local-preference>` – Sets the preference value for the AS path. Optional
  `set local-preference <0-4294967295>`

  `<origin>` – Sets the route origin code for inbound BGP routes. Optional
  `set origin (egp|igp|incomplete)`

  `<originator-id>` – set the originator ID attribute. Optional
  `set originator-id A.B.C.D`

  `<atomic-aggregate/>` – set an atomic aggregate attribute. Optional
  `set atomic-aggregate`

  `<aggregator>` – set the AS number for the route map and router ID. Optional
  `set aggregator as <1-65535> A.B.C.D`
  `<router-as>` – router AS
  `<router-id>` – router ID

  `<as-path>` – modify an autonomous system path for a route. Optional
  `set as-path prepend .<1-65535>`
  `<prepend>` – Value to prepend

  `<next-hop>` – Sets the next-hop address. Optional The possible values are:

  – A.B.C.D : IP address of the next hop
  `set ip next-hop A.B.C.D`

– peer-address : IP address of a peer
`set ip next-hop peer-address`

`<vpnv4-next-hop>` – set a VPNv4 next-hop address. Optional
`set vpnv4 next-hop A.B.C.D`

`<ipv6-next-hop>` – Sets the next-hop IPv6 address. Optional
`<local>` – Set the BGP-4+ link local IPv6 nexthop address
`set ipv6 next-hop local X:X::X:X`
`<global>` – Set the BGP-4+ global IPv6 nexthop address
`set ipv6 next-hop global X:X::X:X`

`<comm-list-del>` – delete matching communities from inbound or outbound updates. Optional
`set comm-list (<1-99>|<100-199>|WORD) delete`

`<communities>` – Identifies a group of communities. Optional
`set community none`
`set community .AA:NN`
`<community>` – Adds a community set clause to the route map. If there is no communities (¡community¿) specified removes the community attribute from route prefixes that pass the route-map.

`<extcommunities>` – Identifies a group of extended communities. Optional
`set extcommunity rt .ASN:nn_or_IP-address:nn`
`set extcommunity soo .ASN:nn_or_IP-address:nn`
`<extcommunity>` – Adds the specified extended communities
`<type>` –
– rt : Set Route Target value
– soo : Set Site of Origin value
`<value>` – an AS number or IP address follow by an integer

A.4 Communities

A.4.1 community-list

- **Use**

```
<bgp>
   <communities>
      <community-list>
         <type> type-choice </type>
         <name> name </name>
         <state> state-choice </state>
         <value> value </value>
      </community-list>
   </communities>
</bgp/>
```

- **Description**
Defines a new community list

```
ip community-list WORD (deny|permit) .AA:NN
ip community-list (standard WORD|<1-99>) (deny|permit) [.AA:NN]
ip community-list (expanded WORD|<100-199>) (deny|permit) .LINE
```

- **Contents**

  `<type>` – This value can be omitted (backward compatibility) and the community list type is automatically detected.
  - standard : Defines an standard community list
  - expanded : Defines an expanded community list
  - `<1-99>` : Defines an standard community list as well as the standard type
  - `<100-199>` : Defines an expanded community list as well as the expanded type

  `<name>` – Community name. This value MUST NOT BE specified just when the type is `<1-199>`

  `<state>` – (permit—deny)

  `<value>` – The value depends on the type of the community list:
  - `standard|<1-99>`: it is a community value. Could be not specified
  - Otherwise : it is an string expression of communities attributes.Can include regular expression to match communities attributes in BGP updates

A.4.2 extcommunity-list

- **Use**

```
<bgp>
   <communities>
      <extcommunity-list>
         <type> type-choice </type>
         <name> name </name>
         <state> state-choice </state>
         <value> value </value>
      </extcommunity-list>
   </communities>
</bgp/>
```

- **Description**
  Defines a new community list

```
ip extcommunity-list WORD (deny|permit) .AA:NN
ip extcommunity-list (standard WORD|<1-99>) (deny|permit) [.AA:NN]
ip extcommunity-list (expanded WORD|<100-199>) (deny|permit) .LINE
```

- **Contents**

  `<type>` –
  - standard : Defines an standard extended community list
  - expanded : Defines an expanded extended community list
  - `<1-99>` : Defines an standard extended community list as well as the standard type
  - `<100-199>` : Defines an expanded extended community list as well as the expanded type

`<name>` – Community name. This value MUST NOT BE specified only when the type is `<1-199>`

`<state>` – (permit—deny)

`<value>` – The value depends on the type of the community list:

- standard: it is a extcommunity value. Could be empty
- Otherwise : it is an string expression of extended communities attributes. Can include regular expression to match communities attributes in BGP updates

References

[1] 4suite: An open-source platform for xml and rdf processing. http://www.4Suite.org.

[2] MADYNES webpage. http://madynes.loria.fr/pyenca.

[3] Quagga Routing Suite. http://www.quagga.net.

[4] Uml: User mode kernel. http://user-mode-linux.sourceforge.net/.

[5] Vnuml: Virtual network user mode kernel. http://jungla.dit.upm.es/~vnuml/.

[6] Yapps (yet another python parser system). http://theory.stanford.edu/ amitp/yapps/.

[7] Mark Pilgrim. *Dive into Python.* Apress, 2004.