# Using the IOS XR Routing Policy Language for BGP

By Marko Berend, Network Consulting Engineer, CCIE #13074

22. 4. 2015

In a modern, BGP enabled network, where a service provider can peer with dozens of other carriers, it can be a daunting task to create BGP policies which both fulfil their goal and are easy to maintain by network administrators. This is a place where IOS XR's RPL feature can realize its full potential.

## What is Routing Policy Language?

If you ever used route maps to create BGP policies, you will quickly learn to appreciate the Route Policy Language (RPL) of IOS XR. Most of the drawbacks of classic Cisco IOS route maps come from their inability to scale well due to the non-modular structure. There is no way to reuse a part of a route map within another route map, for example. The mechanism feels rigid, your router configuration can grow very quickly and it is a matter of time before it gets to the point where it is practically unreadable.

RPL, on the other hand, was developed to benefit large-scale routing configurations of BGP speaking edge routers. While your configuration might still be very long even when using RPL policies, many of the benefits come to light very soon. So what do you get with RPL policies?

## The Benefits of RPL

First of all, RPL configurations are modular and more concise than any route map based setup, as we will demonstrate further along in this article. This brings us the benefits of faster deployment, reduced resources needed for the configuration both at the device level as well as the human effort level. The routing policy configuration can be streamlined and troubleshooting made easier. With the route map method, the number of configuration lines may grow at a much faster rate than that of a RPL counterpart, especially when a large number of routers has to be considered.

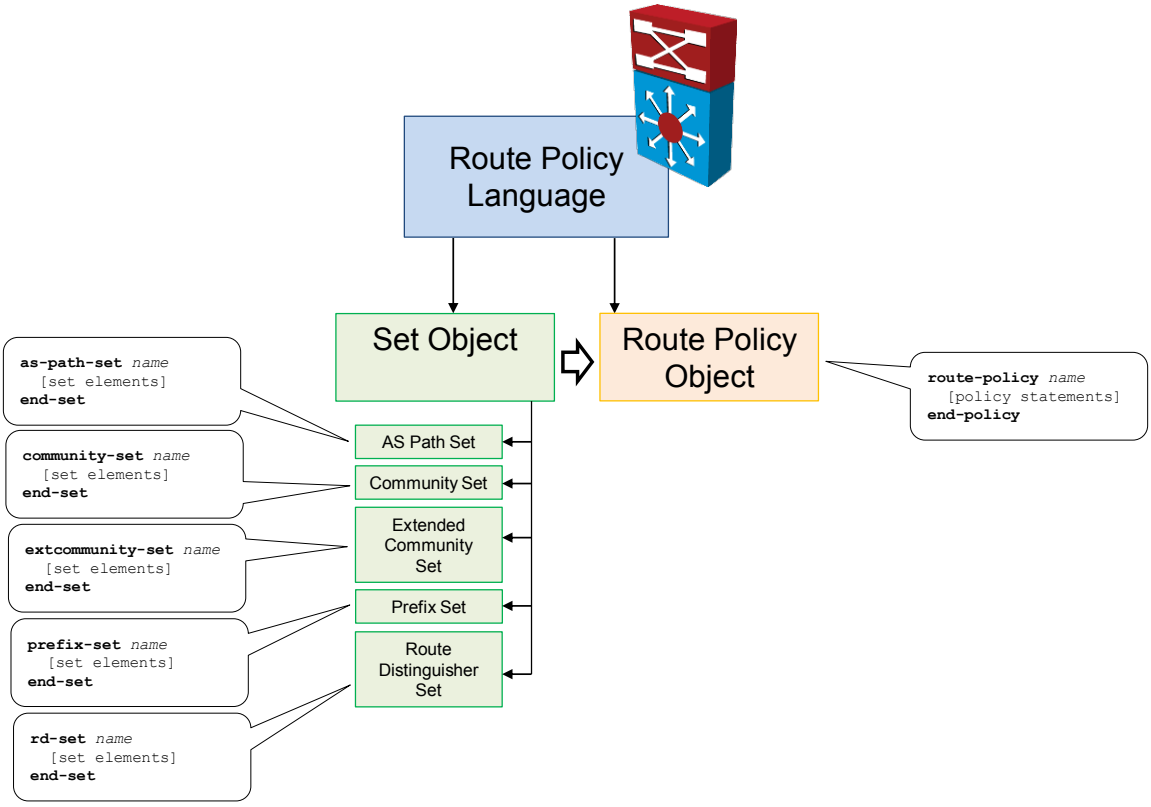
Another fundamental advantage of RPL is the ability to build hierarchical policies from common (created by you) building block policies, which can be referenced from within other policies and reused as needed. If you want, you can even build function like blocks which take parameters as variables. This can be very useful when calling a child policy within a parent policy, but with different parameter values. An example of a parameter is a different value of local preference BGP attributes that you can apply based on the parent policy requirements. In RPL, instead of using prefix lists, objects called sets are used to describe prefixes and more. Now, let us focus a little bit on the structure of the policies using RPL.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

## How does it Work?

Basic building blocks of the RPL are two kinds of persistent, nameable objects: sets and policies. There are five different types of sets, while the policy object is designed as a one command keyword, **route-policy** that covers many different functions. Let's take a look at the structure and configuration blocks for both types of objects in the following graphic.




The natural order of configuration would be first to define your sets, if you use any, and then to reference them from within the route policy objects.

These are the different actions that can be assigned to the policy object statements:

- **pass** – the matched prefix will be allowed (if not later dropped explicitly), the execution of statements continues on within the policy
- **set** – set the value of prefix attributes. The prefix is automatically allowed (if not later dropped explicitly), the execution of statements continues on within the policy. Multiple set statements can be configured
- **done** – the matched prefix will be allowed, stops execution of the policy for that prefix
- **drop** – the matched prefix is discarded



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

Please bear in mind that every route policy has an implicit *drop* at the end so be careful not to unintentionally drop any prefixes that are not covered by your conditions. If it is your intention not to drop any prefixes ever (just modify attributes), it is beneficial to add a *pass* statement at the very end of the route policy.

Policies, when talking about BGP, are attached to BGP neighbors. You can attach a different policy to the inbound and outbound directions, but only one per direction per neighbor.

When writing route policies, there are logical operators at your disposal to create even more complex conditions. The supported operators with an explicit order of precedence are NOT, AND, OR. For example, a simple IPv4 route policy:

```
route-policy myPolicy
  if med eq 50 and destination in (10.0.0.0/8) then
    set local-preference 50
  else
      pass ← this will pass all other prefixes without changes. Otherwise they would be dropped by the
             implicit drop and not appear in the BGP table. Use in case you dont want to drop any prefixes,
             but you do want to modify some of them
  endif
end-policy
```

If you would like to override the order of logical operations you are free to use parentheses to group conditions. In that case the order of operations will apply within the chunks and then between the chunks in parentheses. For example:

```
route-policy myPolicy
  if med eq 100 and (destination in (10.0.0.0/8) or as-path in (500)) then
```

Another very useful feature is the ability to use regular expressions within your policies. Usually you would see them when defining, for example as-path sets.

When talking about sets, if all you need is a simple policy it is possible to do this without any sets at all. Let us demonstrate an example of an equivalent IPv6 route policy using sets and without using sets:



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

```
prefix-set siteBprefixes
  2001:db8:b:10::/64,
  2001:db8:b:11::/64
end-set
!
route-policy SiteA-in-R1
  if destination in siteBprefixes then
    set local-preference 150
    pass ← not explicitly needed here as any set statement will grant a pass. Just for consistency.
  else
    set local-preference 200
    pass
  endif
end-policy
```

... is equivalent to:

```
route-policy SiteA-in-R1
  if destination in (2001:db8:b:10::/64,2001:db8:b:11::/64) then
    set local-preference 150
    pass
  else
    set local-preference 200
    pass
  endif
end-policy
```

Sets will come in handy if your list of prefixes, paths or any other objects supported by the sets feature is very long and/or requires frequent changes or additions. In that way you can modify your sets independently of the route policy.

It is worth noting that in IOS XR there was a change in the way the prefixes are announced from eBGP. By default, when you configure an eBGP neighbor and omit the route-policy definitions, the result will be that no prefixes will be advertised to that peer or received from that peer, more precisely, installed in the BGP table. This is a radical change motivated by making the BGP networks safer by preventing unintended announcements which may poison the BGP tables of other routers; or unfiltered received prefixes to bring down your network. So, unlike IOS, in IOS XR you start with advertising nothing and gradually add prefixes to the policy.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

If you can not be bothered to think too much (practical, but not so safe), the bare minimum configuration that will allow all prefixes, existing and future ones, can be summarized as:

```
route-policy allow_all
  pass
end-policy
```

However, this *allow all* type policy is rarely how you would end up with your configuration, so it might be a good idea to think ahead and create a limiting policy in advance to save yourself a headache or two.

Surely by now you might have concluded that defining a policy does nothing until it is attached to a specific BGP peer. You can easily do this by defining routing policies for every BGP neighbor in your router:

```
router bgp 64500
 neighbor 10.1.1.1
  remote-as 65000
  address-family ipv4 unicast
   route-policy myPolicy_in in
   route-policy myPolicy_out out
```

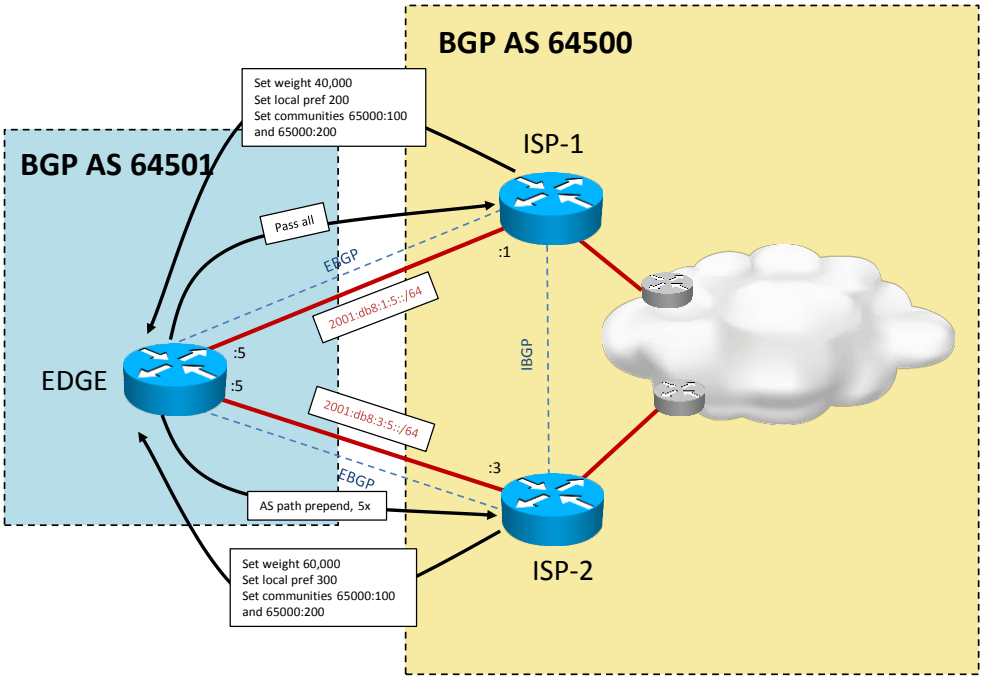
## Hierarchical Policies and Passing Parameters

In this section we will discuss a practical example of a hierarchical policy, meaning that a policy will be built from different reusable blocks with the added functionality of passing parameters onto the policy.

Let's consider a practical example from the following figure. Here we have a scenario of eBGP sessions between the EDGE router and two upstream ISP routers. We will use IPv6 in this scenario but the route policy can also be used for IPv4 with minimal adjustments.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED




The routers in the picture are using BGP to exchange prefixes. We are announcing our local prefix with 5 times AS path prepending towards router ISP-2. When prefixes arrive at the EDGE, we want to change the weight, local preference and add some communities at the same time, to make some destinations preferred over specific links for the outbound traffic, that is the traffic that leaves EDGE. For the exact policy required please refer to the picture above.

You will note that we can reuse some of the policy building blocks on other local routers we might have, thanks to the parameters feature.

The complete configuration of the policy and BGP of router EDGE follows.

First we define the building blocks which will be reused:



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

```
prefix-set neighbors
  2001:db8:1:5::1,
  2001:db8:3:5::3
end-set
!
route-policy parameters($weight, $localpref)
  set weight $weight
  set local-preference $localpref
end-policy
!
route-policy communities
  set community (65000:100) additive
  set community (65000:200) additive
end-policy
!
route-policy eBGP_neighbor($neighbors, $weight, $localpref)
  if next-hop in $neighbors then
    apply parameters($weight, $localpref)
    apply communities
  else
    apply parameters(32000, 100)
    set community (100:100)
    set community (100:200)
  endif
end-policy
!
```

Note how we use policy calls within other policies with the **apply** command. Also, the use of parameters that are passed between policies can be seen by using the **$** sign before the arbitrary parameter name. The complete flow of the eBGP_neighbor policy so far would be: if you see that the next-hop of a prefix is contained in the **neighbors** prefix set (which in fact contains the list of BGP peers we are peering with), then execute the two apply statements, else set a generic set of attributes – 32000, 100 and two generic attributes 100:100 and 100:200. The else clause ensures we don't drop any unexpected prefixes from neighbors we might add in the future, however we are marking them in a different way, so we can recognize/filter them later.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

Next we define the actual policies to be attached to the BGP neighbors, which will comply with our original policy requirements.

```
route-policy ISP_1_in
  apply eBGP_neighbor(neighbors, 40000, 200)
end-policy
!
route-policy ISP_2_in
  apply eBGP_neighbor(neighbors, 60000, 300)
end-policy
!
route-policy ISP_1_out
  pass
end-policy
!
route-policy ISP_2_out
  if as-path in (ios-regex '^$') then
    prepend as-path 64501 5
  endif
end-policy
!
!  The BGP part
!
router bgp 64501
 bgp router-id 5.5.5.5
 address-family ipv6 unicast
  network 2001:db8::5:5:5:5/128
  network 2001:db8:a:10::/64
  network 2001:db8:a:11::/64
 !
 neighbor 2001:db8:1:5::1
  remote-as 64500
  keychain EBGP
  address-family ipv6 unicast
   route-policy ISP_1_in in
   route-policy ISP_1_out out
  !
 !
 neighbor 2001:db8:3:5::3
  remote-as 64500
  address-family ipv6 unicast
   route-policy ISP_2_in in
   route-policy ISP_2_out out
  !
```

Note how we are applying the same **eBGP_neighbor** policy template from within ISP_1_in and ISP_2_in policies, but with different parameters.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

Now let us see if our policies are getting the desired results. We are expecting all prefixes from ISP-1 to have weight 40,000, local preference 200 and communities set. Prefixes from ISP-2 should have the weight 60,000 and local preference 300, same communities.

```
RP/0/0/CPU0:EDGE#show bgp ipv6 unicast
Thu Jan 22 03:01:24.991 UTC
BGP router identifier 5.5.5.5, local AS number 64501
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0xe0800000   RD version: 181
BGP main routing table version 181
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*  2001:db8::1:1:1:1/128
                      2001:db8:1:5::1          0    200  40000 64500 i
*>                    2001:db8:3:5::3               300  60000 64500 i
*  2001:db8::2:2:2:2/128
                      2001:db8:1:5::1               200  40000 64500 i
*>                    2001:db8:3:5::3               300  60000 64500 i
*  2001:db8::3:3:3:3/128
                      2001:db8:1:5::1               200  40000 64500 i
*>                    2001:db8:3:5::3          0    300  60000 64500 i
*  2001:db8::4:4:4:4/128
                      2001:db8:1:5::1               200  40000 64500 i
*>                    2001:db8:3:5::3               300  60000 64500 i

(output omitted)

Processed 19 prefixes, 35 paths
RP/0/0/CPU0:EDGE#
```



NIL LEARNING ©2015 All rights reserved.

Security tag: PROTECTED

```
RP/0/0/CPU0:EDGE#show bgp ipv6 unicast  2001:db8::1:1:1:1/128
Thu Jan 22 03:04:48.177 UTC
BGP routing table entry for 2001:db8::1:1:1:1/128
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                176         176
Last Modified: Jan 22 02:15:53.551 for 00:48:54
Paths: (2 available, best #2)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  64500
    2001:db8:1:5::1 from 2001:db8:1:5::1 (1.1.1.1)
      Origin IGP, metric 0, localpref 200, weight 40000, valid, external, import
suspect
      Received Path ID 0, Local Path ID 0, version 0
      Community: 65000:100 65000:200
      Origin-AS validity: not-found
  Path #2: Received by speaker 0
  Not advertised to any peer
  64500
    2001:db8:3:5::3 from 2001:db8:3:5::3 (3.3.3.3)
      Origin IGP, localpref 300, weight 60000, valid, external, best, group-best,
import-candidate, import suspect
      Received Path ID 0, Local Path ID 1, version 176
      Community: 65000:100 65000:200
      Origin-AS validity: not-found
```



NIL LEARNING ©2015 All rights reserved.

Security tag: PROTECTED

And finally how does our prepending AS path look like on ISP-2? Let us take a look:

```
RP/0/0/CPU0:ISP-2#show bgp ipv6 unicast dfa-regex 64501
Thu Jan 22 03:07:56.082 UTC
BGP router identifier 3.3.3.3, local AS number 64500
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0xe0800000   RD version: 106
BGP main routing table version 106
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*>i2001:db8::5:5:5:5/128
                      2001:db8::1:1:1:1
                                                0    100      0 64501 i
*                      2001:db8:3:5::5           0              0 64501 64501 64501
64501 64501 64501 i
*>i2001:db8:a:10::/64 2001:db8::1:1:1:1
                                                0    100      0 64501 i
*                      2001:db8:3:5::5           0              0 64501 64501 64501
64501 64501 64501 i
*>i2001:db8:a:11::/64 2001:db8::1:1:1:1
                                                0    100      0 64501 i
*                      2001:db8:3:5::5           0              0 64501 64501 64501
64501 64501 64501 i

Processed 3 prefixes, 6 paths
```

## Editing Your Sets and Policies

As you might have noticed, there are no line numbers in a policy configuration or set definitions. Here we might have a very serious problem because the behavior of IOS XR is slightly different than regular IOS when it comes to adding or modifying lines of configuration. Remember how on IOS you can simply add lines to an ACL? Well, you cannot do that for route policies in IOS XR. If you start to configure an existing policy or a set, from the configure mode of CLI, the policy/set will be erased and you will end up with your addition only. One alternative is to copy everything in a text editor on your PC, edit there, then paste the whole thing again in the CLI of



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

the device. This might prove impractical and cumbersome, but fear not, there is a more elegant solution to this problem.

IOS XR features multiple built in text editors which can be used to edit your policies and sets on the fly. All you need to do is to call your object with the **edit** CLI command. How nice is that? Let us see how it is done:

```
RP/0/0/CPU0:R7#edit ?
  as-path-set       edit an as-path-set
  community-set     edit a community-set
  extcommunity-set  edit an extended-community-set
  policy-global     edit policy-global definitions
  prefix-set        edit a prefix-set
  rd-set            edit a rd-set
  route-policy      edit a route-policy
  tag-set           edit a tag-set
RP/0/0/CPU0:R7#edit route-policy setOrigin ?
  emacs  to use Emacs editor
  nano   to use nano editor
  vim    to use Vim editor
  <cr>
RP/0/0/CPU0:R7#edit route-policy setOrigin vim

(at this point we get a VI based editor with the policy, then we save and exit)

Wed Jan 21 23:34:05.095 UTC
[OK]
"/dev/shmem/rpl_edit.778520" 4 lines, 53 characters written
Proceed with commit (yes/no/cancel)? [cancel]: yes
Parsing.cy
53 bytes parsed in 1 sec (51)bytes/sec
Committing.
Prepared commit in 0 sec
~
1 items committed in 1 sec (0)items/sec
Updating.
Updated Commit database in 1 sec
```

From the parsing output we see that everything went well. You would surely get warnings and errors if you made any semantic errors or typos while editing and commit would not succeed.

## Conclusion

In this article we covered the use of RPL in configuring the BGP policies, however RPL can be used in conjunction with other features as well. You



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

can create route policies, for example, to facilitate route redistribution. For more detailed information on RPL, regex and other topics related to IOS XR please refer to **cisco.com/go/iosxr**.



NIL LEARNING ©2015 All rights reserved.

Security tag: PROTECTED

## NIL – More Than Just a Training Company

NIL Learning delivers the leading-edge Cisco training to IT professionals and companies around the globe. Through field-proven experts — each both active engineer and instructor — NIL Learning enhances the standard learning curriculum with real-life experience and helps clients to maximize their training investment.

NIL Learning is part of NIL, a leading global IT solutions provider. Since 1992, NIL has been at the forefront of advanced contributors to strategic partner Cisco's technologies, learning curriculum and value-added solutions deployed to clients around the globe. Today, NIL has earned the highest certifications offered by Cisco, VMware, EMC, HP, IBM, Microsoft, F5, MobileIron, RSA, VCE and others. Their portfolio of solutions consists of managed services, professional services and learning services.

NIL is headquartered in Slovenia, with regional offices in Croatia, Serbia, Saudi Arabia, the U.S., South Africa, Morocco, Nigeria, Kenya and Botswana.

## Why Learn at NIL LEARNING?

- All NIL LEARNING instructors are **field-proven experts** - each both active engineer, content developer and instructor.
- **75% of NIL LEARNING engineers hold CCSI certifications**, and **20 have already achieved the respected CCIE rank**.
- NIL LEARNING **enhances the standard learning curriculum with real-life experience** and helps clients to maximize their training investment.
- NIL has been a Cisco training partner since 1993; today NIL holds **Cisco Learning Partner Specialized** status and **Cisco Business Learning Partner** status.
- NIL was awarded the **Cisco Most Business Relevant Learning Partner in MEA** in 2010 and the **Most Innovative Learning Partner** in MEA.
- NIL received the **Innovation Award** for its Technology Led Training and its extensive contribution to Cisco learning solutions at the Cisco EMEAR Learning Partner Summit in 2012.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

- NIL received the **Innovation Award** for its Technology Led Training and Advanced Engineer Program at the Cisco Global Learning Partner Summit in 2013.
- NIL won the **Cisco Global Learning Partner of the Year** award at the Cisco Partner Summit in 2014.
- NIL Learning runs a centralized training schedule across the whole EMEA region.



NIL LEARNING ©2015 All rights reserved. Security tag: PROTECTED

# More Info


**Slovenia**

T: +386 1 4746 500

E: sales-support@nil.com

**Botswana**

T: +267 318 1684

E: training@it-iq.bw

**Croatia**

T: +385 (0)51 583 255

E: info-nilcroatia@nil.com

**Kenya**

T: +254 773 145 680

E: mea_sales@nil.com

**Morocco**

T: +212(0) 660 808 394

E: info-nilmorocco@nil.com

**Nigeria**

T: +234 1 903 53 47

E: mea_sales@nil.com

**Saudi Arabia**

T: +966 1 465 4641

E: info.nilme@nil.com

**Serbia**

T: +381 11 2420 388

E: info-nilserbia@nil.co.rs

**South Africa**

T: +27 (0)11 575 4637

E: mea_sales@nil.com

**Turkey**

T: +386 1 4746 500

E: sales-support@nil.com

**USA**

T: +1 612 886 3900

E: info-nilusa@nil.com

learning.nil.com




NIL LEARNING ©2015 All rights reserved.

Security tag: PROTECTED