



# Configuring Flexible Command Line Interface

This module describes how to configure and use flexible command line interface (CLI) configuration groups.

- [Flexible CLI Configuration Groups, on page 1](#)
- [Flexible Configuration Restrictions, on page 1](#)
- [Configuring a Configuration Group, on page 3](#)
- [Verifying the Configuration of Configuration Groups, on page 5](#)
- [Regular Expressions in Configuration Groups, on page 6](#)

## Flexible CLI Configuration Groups

Flexible command line interface (CLI) configuration groups provide the ability to minimize repetitive configurations by defining a series of configuration statements in a configuration group, and then applying this group to multiple hierarchical levels in the router configuration tree.

Flexible CLI configuration groups utilize regular expressions that are checked for a match at multiple submodes of the configuration tree based on where the group is applied within the hierarchy. If a match is found at a configuration submode, the corresponding configuration defined in the group is inherited within the matched submode.

Flexible CLI configuration groups also provide an auto-inheritance feature. Auto-inheritance means that any change done to a CLI configuration group is automatically applied to the configuration in any matched submodes that have an apply-group at that hierarchical level. This allows you to make a configuration change or addition once, and have it applied automatically in multiple locations, depending on where you have applied the flexible CLI configuration group.

## Flexible Configuration Restrictions

Note these restrictions while using flexible configuration groups:

- Flexible CLI configuration groups are not supported in administration configurations and corresponding apply-groups are not supported in administration configurations.
- Use of preconfigured interfaces in configuration groups is not supported.
- Downgrading from an image that supports configuration groups to an image that does not support them is not supported.

- Access lists, quality of service and route policy configurations do not support the use of configuration groups. Configurations such as these are not valid:

```
group g-not-supported
  ipv4 access-list ...
  !
  ipv6 access-list ...
  !
  ethernet-service access-list ...
  !
  class-map ...
  !
  policy-map ...
  !
  route-policy ...
  !
end-group
```

You can, however, reference such configurations, as shown in this example:

```
group g-reference-ok
  router bgp 6500
  neighbor 7::7
  remote-as 65000
  bfd fast-detect
  update-source Loopback300
  graceful-restart disable
  address-family ipv6 unicast
    route-policy test1 in
    route-policy test2 out
  soft-reconfiguration inbound always
  !
  !
  !
  interface Bundle-Ether1005
  bandwidth 10000000
  mtu 9188
    service-policy output input_1
  load-interval 30
  !
end-group
```

- Some regular expressions are not supported within groups. For example, ‘?’ , ‘|’ and ‘\$,’ are not supported within groups. Also some characters such as /d and /w are not supported.

- The choice operator “|” to express multiple match expressions within a regular expression is not supported. For example, these expressions are not supported:

Gig.\*|Gig.\*\..\*—To match on either Gigabit Ethernet main interfaces or Gigabit Ethernet sub-interfaces.

Gig.\*0/0/0/[1-5]|Gig.\*0/0/0/[10-20]—To match on either Gig.\*0/0/0/[1-5] or Gig.\*0/0/0/[10-20].

'TenGigE.\*|HundredGigE.\*—To match on either TenGigE.\* or HundredGigE.\*.

- Commands that require a node identifier for the **location** keyword are not supported. For example, this configuration is not supported:

```
lpts pifib hardware police location 0/RP0/CPU0
```

- Overlapping regular expressions within a configuration group for the same configuration are not supported. For example:

```
group G-INTERFACE
interface 'gig.*a.*'
    mtu 1500
!
interface 'gig.*e.* '
    mtu 2000
!
end-group

interface gigabitethernet0/0/0/* ---- where * is 0 to 31
    apply-group G-INTERFACE
```

This configuration is not permitted because it cannot be determined whether the `interface GigabitEthernet0/0/0/*` configuration inherits `mtu 1500` or `mtu 2000`. Both expressions in the configuration group match `GigabitEthernet0/0/0/*`.

- Up to eight configuration groups are permitted on one `apply-group` command.

## Configuring a Configuration Group

A configuration group includes a series of configuration statements that can be used in multiple hierarchical levels in the router configuration tree. By using regular expressions in a configuration group, you can create generic commands that can be applied in multiple instances.

Use this task to create and use a configuration group.




---

**Note** Flexible CLI configurations are not available through the XML interface.

---

### Procedure

- 
- Step 1** **configure**
- Example:**
- ```
RP/0/RP0/CPU0:router# configure
```
- Enters mode.
- Step 2** **group** *group-name*
- Example:**
- ```
RP/0/RP0/CPU0:router(config)# group g-interf
```
- Specifies a name for a configuration group and enters group configuration mode to define the group. The *group-name* argument can have up to 32 characters and cannot contain any special characters.
- Step 3** Enter configuration commands, starting from global configuration mode. Use regular expressions for interface names and other variable instances.

**Example:**

```
RP/0/RP0/CPU0:router (config) # group g-interf
RP/0/RP0/CPU0:router (config-GRP) # interface 'GigabitEthernet.*'
RP/0/RP0/CPU0:router (config-GRP-if) # mtu 1500
```

Specifies the configuration statements that you want included in this configuration group.

For more information regarding the use of regular expressions, see [Configuration Group Inheritance with Regular Expressions: Example, on page 15](#). This example is applicable to all Gigabit Ethernet interfaces.

**Step 4 end-group****Example:**

```
RP/0/RP0/CPU0:router (config-GRP-if) # end-group
```

Completes the configuration of a configuration group and exits to global configuration mode.

**Step 5 apply-group****Example:**

```
RP/0/RP0/CPU0:router (config) # interface GigabitEthernet0/2/0/0
RP/0/RP0/CPU0:router (config-if) # apply-group g-interf
```

Adds the configuration of the configuration group into the router configuration applicable at the location that the group is applied. Groups can be applied in multiple locations, and their effect depends on the location and context.

The MTU value from the group g-interf is applied to the interface TenGigE0/11/0/0. If this group is applied in global configuration mode, the MTU value is inherited by all Gigabit Ethernet interfaces that do not have an MTU value configured.

---

## Simple Configuration Group: Example

This example shows how to use configuration groups to add a global configuration to the system:

```
RP/0/RP0/CPU0:router (config) # group g-logging
RP/0/RP0/CPU0:router (config-GRP) # logging trap notifications
RP/0/RP0/CPU0:router (config-GRP) # logging console debugging
RP/0/RP0/CPU0:router (config-GRP) # logging monitor debugging
RP/0/RP0/CPU0:router (config-GRP) # logging buffered 10000000
RP/0/RP0/CPU0:router (config-GRP) # end-group

RP/0/RP0/CPU0:router (config) # apply-group g-logging
```

When this configuration is committed, all commands contained in the g-logging configuration group are committed.

## Configuration Group Applied to Different Places: Example

Configuration groups can be applied to different places, and their effect depends on the context within which they are applied. Consider this configuration group:

```
RP/0/RP0/CPU0:router(config)# group g-interfaces
RP/0/RP0/CPU0:router(config-GRP)# interface 'TenGigE.*'
RP/0/RP0/CPU0:router(config-GRP-if)# mtu 1500
RP/0/RP0/CPU0:router(config-GRP-if)# exit
RP/0/RP0/CPU0:router(config-GRP)# interface 'GigabitEthernet.*'
RP/0/RP0/CPU0:router(config-GRP-if)# mtu 1000
RP/0/RP0/CPU0:router(config-GRP-if)# exit
RP/0/RP0/CPU0:router(config-GRP)# interface 'HundredGigE.*'
RP/0/RP0/CPU0:router(config-GRP-if)# mtu 2000
RP/0/RP0/CPU0:router(config-GRP-if)# end-group
```

This group can be applied to Gigabit Ethernet, TenGigE and HundredGigE interface and in each instance the applicable MTU is applied. For instance, in the following example, the Gigabit Ethernet interface is configured to have an MTU of 1000:

```
RP/0/RP0/CPU0:router(config)# interface GigabitEthernet0/2/0/0
RP/0/RP0/CPU0:router(config-if)# apply-group g-interfaces
RP/0/RP0/CPU0:router(config-if)# ipv4 address 2.2.2.2 255.255.255.0
```

In the following example, the TenGigE interface is configured to have an MTU of 1500:

```
RP/0/RP0/CPU0:router(config)# interface TenGigE0/0/0/16
RP/0/RP0/CPU0:router(config-if)# apply-group g-interfaces
RP/0/RP0/CPU0:router(config-if)# ipv4 address 3.3.3.3 255.255.255.0
```

The same configuration group is used in both cases, but only the applicable configuration statements are used.

## Verifying the Configuration of Configuration Groups

Use this task to verify the router configuration using configuration groups:

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>show running-config group</b> [ <i>group-name</i> ]  <b>Example:</b> <pre>RP/0/RP0/CPU0:router# show running-config group group g-int-ge interface 'GigabitEthernet.*' mtu 1000 negotiation auto ! end-group</pre>	Displays the contents of all or a specific configured configuration group.
<b>Step 2</b>	<b>show running-config</b>  <b>Example:</b> <pre>show running-config Example:</pre>	Displays the running configuration. Any applied groups are displayed. There is no indication as to whether these configuration groups affect the actual configuration or not. In this example, although the group G-INTERFACE-MTU is

	Command or Action	Purpose
	<pre>RP/0/RP0/CPU0:router# show running-config interface group G-INTERFACE-MTU interface 'GigabitEthernet.*' mtu 1500 ! end-group  interface GigabitEthernet0/4/1/1 apply-group G-INTERFACE-MTU ! interface GigabitEthernet0/0/0/1 apply-group group G-INTERFACE-MTU mtu 2000</pre>	<p>applied to interface GigabitEthernet0/0/0/1, the configured MTU value is 2000 and not 1500. This happens if the command <code>mtu 2000</code> is configured directly on the interface. An actual configuration overrides a configuration group configuration if they are the same.</p>
<b>Step 3</b>	<p><b>show running-config inheritance</b></p> <p><b>Example:</b></p> <pre>RP/0/RP0/CPU0:router# show running-config inheritance  group G-INTERFACE-MTU interface 'GigabitEthernet.*' mtu 1500 ! end-group . . interface GigabitEthernet0/4/1/1 ## Inherited from group G-INTERFACE-MTU mtu 1500 ! interface GigabitEthernet0/0/0/1 mtu 2000 ! . .</pre>	<p>Displays the inherited configuration wherever a configuration group has been applied.</p>
<b>Step 4</b>	<p><b>show running-config interface x/y/z inheritance detail</b></p> <p><b>Example:</b></p> <pre>RP/0/RP0/CPU0:router# show running-config interface GigabitEthernet0/4/1/1 inheritance detailExample:  interface GigabitEthernet0/4/1/1 ## Inherited from group G-INTERFACE-MTU mtu 1500</pre>	<p>Displays the inherited configuration for a specific configuration command.</p>

## Regular Expressions in Configuration Groups

Regular expressions are used in configuration groups to make them widely applicable. Portable Operating System Interface for UNIX (POSIX) 1003.2 regular expressions are supported in the names of configuration statements. Single quotes must be used to delimit a regular expression.




---

**Note** Not all POSIX regular expressions are supported.

---

### Regular Expressions for Interface Identifiers

Configuration groups do not accept exact interface identifiers. You must use a regular expression to identify a group of interfaces that are applicable to the configuration group. The regular expression `.*` is not allowed. You must begin the regular expression for an interface identifier with an unambiguous word, followed by the regular expression. For example, to configure Gigabit Ethernet interfaces, use the regular expression `'GigabitEthernet.*'`.

To display a list of available interface types for your router configuration, enter **interface ?** at the configuration group prompt:

```
RP/0/RP0/CPU0:router(config-GRP) # interface ?

ATM                'RegExp': ATM Network Interface(s)
BVI                 'RegExp': Bridge-Group Virtual Interface
Bundle-Ether       'RegExp': Aggregated Ethernet interface(s)
GigabitEthernet    'RegExp': GigabitEthernet/IEEE 802.3 interface(s)
IMA                 'RegExp': ATM Network Interface(s)
Loopback            'RegExp': Loopback interface(s)
MgmtEth             'RegExp': Ethernet/IEEE 802.3 interface(s)
Multilink           'RegExp': Multilink network interface(s)
Null                'RegExp': Null interface
PW-Ether            'RegExp': PWHE Ethernet Interface
PW-IW               'RegExp': PWHE VC11 IP Interworking Interface
Serial              'RegExp': Serial network interface(s)
tunnel-ip           'RegExp': GRE/IPinIP Tunnel Interface(s)
tunnel-mte          'RegExp': MPLS Traffic Engineering P2MP Tunnel interface(s)
tunnel-te           'RegExp': MPLS Traffic Engineering Tunnel interface(s)
tunnel-tp           'RegExp': MPLS Transport Protocol Tunnel interface
```




---

**Note** Although you are required to enter only enough characters for the interface type to be unique, it is recommended that you enter the entire phrase. All interface types used in regular expressions are case-sensitive.

---

To specify a subinterface, prefix the expression with the characters `\.` (backslash period). For example, use `interface 'GigabitEthernet.*\.*'` to configure all Gigabit Ethernet subinterfaces.

You can specify Layer 2 transport interfaces or point-to-point interfaces as shown in these examples:

```
group g-l2t
  interface 'Gi.*\.*' l2transport
  .
  .
end-group
group g-ptp
  interface 'Gi.*\.*' point-to-point
  .
  .
end-group
```





## Resolving to a Uniform Type

Regular expressions must resolve to a uniform type. This is an example of an illegal regular expression:

```
group g-invalid
  interface \.*'
    bundle port-priority 10
  !
  interface \.*Ethernet.*'
    bundle port-priority 10
  !
end-group
```

In this example, the **bundle** command is supported for interface type GigabitEthernet but not for interface type 'FastEthernet'. The regular expressions `\.*'` and `\.*Ethernet.*'` match both GigabitEthernet and FastEthernet types. Because the **bundle** command is not applicable to both these interface types, they do not resolve to a uniform type and therefore the system does not allow this configuration.




---

**Note** If the system cannot determine from the regular expression what the configuration should be, the expression is not considered valid.

---




---

**Note** The regular expression `\.*'` is not allowed when referring to an interface identifier. You must begin the regular expression for an interface identifier with an unambiguous word, followed by the regular expression. Refer to *Regular Expressions for Interface Identifiers* in this section for more information.

---

## Overlapping Regular Expressions

Regular expressions are used in names of configuration statements within a configuration group. This permits inheritance by the configuration when applied to matching names. Single quotes are used to delimit the regular expression. Overlapping regular expression within a configuration group for the same configuration is permitted.

The example, given below, illustrates the process of creating and applying multiple configuration groups:

```
RP/0//CPU0:router(config)#group FB_flexi_snmp
RP/0//CPU0:router(config-GRP)# snmp-server vrf \.*'
RP/0//CPU0:router(config-GRP-snmp-vrf)# host 1.1.1.1 traps version 2c group_1
RP/0//CPU0:router(config-GRP-snmp-vrf)# host 1.1.1.1 informs version 2c group_1
RP/0//CPU0:router(config-GRP-snmp-vrf)# context group_1

RP/0//CPU0:router(config-GRP-snmp-vrf)#
RP/0//CPU0:router(config-GRP-snmp-vrf)#commit

RP/0//CPU0:router(config-GRP-snmp-vrf)#root
RP/0//CPU0:router(config)#
RP/0//CPU0:router(config)#snmp-server vrf vrf1
RP/0//CPU0:router(config-snmp-vrf)#snmp-server vrf vrf10
RP/0//CPU0:router(config-snmp-vrf)#!
RP/0//CPU0:router(config-snmp-vrf)#snmp-server vrf vrf100
RP/0//CPU0:router(config-snmp-vrf)#
RP/0//CPU0:router(config-snmp-vrf)#commit

RP/0//CPU0:router(config-snmp-vrf)#root
```

```

RP/0//CPU0:router(config)#
RP/0//CPU0:router(config)#apply-group FB_flexi_snmp
RP/0//CPU0:router(config)#do sh running-config group
group FB_flexi_snmp
  snmp-server vrf '.*'
    host 1.1.1.1 traps version 2c group_1
    host 1.1.1.1 informs version 2c group_1
  context group_1
!
end-group
apply-group FB_flexi_snmp
snmp-server vrf vrf1
!
snmp-server vrf vrf10
!
snmp-server vrf vrf100
!
RP/0//CPU0:ios#show running-config inheritance detail

```

```

group FB_flexi_snmp
  snmp-server vrf '.*'
    host 1.1.1.1 traps version 2c group_1
    host 1.1.1.1 informs version 2c group_1
  context group_1
!
end-group
snmp-server vrf vrf1
## Inherited from group FB_flexi_snmp
host 1.1.1.1 traps version 2c group_1
## Inherited from group FB_flexi_snmp
host 1.1.1.1 informs version 2c group_1
## Inherited from group FB_flexi_snmp
context group_1
!
snmp-server vrf vrf10
## Inherited from group FB_flexi_snmp
host 1.1.1.1 traps version 2c group_1
## Inherited from group FB_flexi_snmp
host 1.1.1.1 informs version 2c group_1
## Inherited from group FB_flexi_snmp
context group_1
!
snmp-server vrf vrf100
## Inherited from group FB_flexi_snmp
host 1.1.1.1 traps version 2c group_1
## Inherited from group FB_flexi_snmp
host 1.1.1.1 informs version 2c group_1
## Inherited from group FB_flexi_snmp
context group_1

```

The example given below demonstrates the regular expression. In this example `snmp-server vrf '.*'` and `snmp-server vrf '[\w]+'` are two different regular expressions.

```

group FB_flexi_snmp
snmp-server vrf '.*'
host 1.1.1.1 traps version 2c group_1
host 1.1.1.1 informs version 2c group_1
context group_1
!
snmp-server vrf '[\w]+'

```

```

host 2.2.2.2 traps version 2c group_2
host 2.2.2.2 informs version 2c group_2
context group_2
!
end-group

```

This individual regular expression gets combined to all the three expressions - `snmp-server vrf vrf1`, `snmp-server vrf vrf10` and `snmp-server vrf vrf100` as given below.

```

apply-group FB_flexi_snmp
snmp-server vrf vrf1
!
snmp-server vrf vrf10
!
snmp-server vrf vrf100
!

```

In a configuration group, there can be instances of regular expressions overlap. In such cases, the regular expression with the highest priority is activated and inherited, when applied. It has that regular expression, which comes first in the lexicographic order that has the highest priority.

The following example shows how to use overlapping regular expressions and how the expression with higher priority is applied:

```

group FB_flexi_snmp

snmp-server vrf '.*'

host 1.1.1.1 traps version 2c group_1

host 1.1.1.1 informs version 2c group_1

context group_1

!

snmp-server vrf '[\w]+'

host 2.2.2.2 traps version 2c group_2
host 2.2.2.2 informs version 2c group_2
context group_2
!
end-group

```

The expression shown below has the highest priority:

```

group FB_flexi_snmp

snmp-server vrf '.*'

host 1.1.1.1 traps version 2c group_1

host 1.1.1.1 informs version 2c group_1

context group_1

```

The examples given above, show two different regular expression `snmp-server vrf '.*'` and `snmp-server vrf '[\w]+'`.

The expression below, shows how these two expressions get merged together:

```
apply-group FB_flexi_snmp

snmp-server vrf vrf1
!
snmp-server vrf vrf10
!
snmp-server vrf vrf100
!
```

Any change in a regular expression with lower priority will not affect the inheritance.

Any changes made to an existing regular expression, which is of less (non-top) priority, it will not have any effect on the inheritance.

```
snmp-server vrf '[\w]+'
```

```
host 2.2.2.2 traps version 2c group_2
host 2.2.2.2 informs version 2c group_2
context group_2
```

The expression with the higher priority gets inherited, as shown below:

```
group FB_flexi_snmp

snmp-server vrf '.*'

host 1.1.1.1 traps version 2c group_1

host 1.1.1.1 informs version 2c group_1

context group_1
```

### Apply Groups Priority Inheritance

Priority governs inheritance.




---

**Note** From the Cisco IOS XR, Release 6.3.1 onwards, you are able to enter the Flexible CLI config group definition, **apply-group** and **exclude-group** command in any order as long as the entire commit has all the group definitions needed.

---

Apply groups priority inheritance helps flexible configuration groups to handle common configuration statements between groups. When multiple configuration groups have common configuration statements, the inheritance priority is such that the configuration statements present in inner groups have precedence over those configuration statements present in outer groups. In case of tiebreakers, the priority is assigned in accordance to the lexicographical order of regular expressions. User defined order of commands are not accepted.

For example, a configuration statement in configuration group ONE has precedence over another group. A configuration statement in configuration group SEVEN is used only if it does not exist in any other group. Within a configuration group, inheritance priority is the longest match.

```

apply-group SIX SEVEN
  router ospf 0
apply-group FOUR FIVE
  area 0
apply-group THREE
  interface TenGigE0/11/0/0
apply-group ONE TWO

!
!
!
```

The above example shows two scenarios. The inner most group (**apply-group ONE TWO**) has the highest priority. Case 1

The first scenario shows which group gets the priority. The example states which group is applied between different configuration groups (different groups with nothing in common). While applying group one (ONE TWO), all the seven groups matches the interface `interface TenGigE0/11/0/0-` is applied.

Case 2

Here, when all have the same (common) configuration, group one will be active. That is `apply-group ONE TWO` is active. If group ONE is deleted, then group TWO will be active.

## Configuration Examples Using Regular Expressions

### Configuration Group with Regular Expression: Example

This example shows the definition of a configuration group for configuring Gigabit Ethernet interfaces with ISIS routing parameters, using regular expressions for the exact interface:

```

RP/0/RP0/CPU0:router(config)# group g-isis-gige
RP/0/RP0/CPU0:router(config-GRP)# router isis '*'
RP/0/RP0/CPU0:router(config-GRP-isis)# interface 'GigabitEthernet.*'
RP/0/RP0/CPU0:router(config-GRP-isis-if)# lsp-interval 20
RP/0/RP0/CPU0:router(config-GRP-isis-if)# hello-interval 40
RP/0/RP0/CPU0:router(config-GRP-isis-if)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-GRP-isis-if-af)# metric 10
RP/0/RP0/CPU0:router(config-GRP-isis-if-af)# end-group
RP/0/RP0/CPU0:router(config)#
```

To illustrate the use of this configuration group, assume that you want to configure these Gigabit Ethernet interfaces with the ISIS routing parameters:

```

router isis green
  interface TenGigE0/11/0/0
    lsp-interval 20
    hello-interval 40
    address-family ipv4 unicast
    metric 10
  !
!
```

```

interface TenGigE0/11/0/1
  lsp-interval 20
  hello-interval 40
  address-family ipv4 unicast
  metric 10
!
!
interface TenGigE0/11/0/2
  lsp-interval 20
  hello-interval 40
  address-family ipv4 unicast
  metric 10
!
!
interface TenGigE0/11/0/3
  lsp-interval 20
  hello-interval 40
  address-family ipv4 unicast
  metric 10
!
!
!
```

There are three possible ways to use the configuration group to configure these interfaces. The first is by applying the group within the interface configuration, as shown here:

```

router isis green
  interface TenGigE0/11/0/0
    apply-group g-isis-gige
  !
  !
  interface TenGigE0/11/0/1
    apply-group g-isis-gige
  !
  !
  interface TenGigE0/11/0/2
    apply-group g-isis-gige
  !
  !
  interface TenGigE0/11/0/3
    apply-group g-isis-gige
  !
  !
!
```

In this situation, only the interfaces to which you apply the configuration group inherit the configuration.

The second way to configure these interfaces using the configuration group is to apply the configuration group within the **router isis** configuration, as shown here:

```

router isis green
  apply-group g-isis-gige
  interface TenGigE0/11/0/0
  !
  interface TenGigE0/11/0/1
  !
  interface TenGigE0/11/0/2
  !
  interface TenGigE0/11/0/3
  !
!
```

```
!
```

In this way, any other Gigabit Ethernet interfaces that you configure in the ISIS green configuration also inherit these configurations.

The third way to configure these interfaces using the configuration group is to apply the group at the global level as shown here:

```

    apply-group g-isis-gige
router isis green
interface TenGigE0/11/0/0
!
interface TenGigE0/11/0/1
!
interface TenGigE0/11/0/2
!
interface TenGigE0/11/0/3
!
!
```

In this example, the configuration of the group is applied to all Gigabit Ethernet interfaces configured for ISIS.

## Configuration Group Inheritance with Regular Expressions: Example

### Local Configuration Has Precedence Over Configuration Group

An explicit configuration takes precedence over a configuration applied from a configuration group. For example, assume that this configuration is running on the router:

```

router ospf 100
  packet-size 1000
!
```

You configure this configuration group, apply it, and commit it to the configuration.

```

RP/0/RP0/CPU0:router(config)# group g-ospf
RP/0/RP0/CPU0:router(config-GRP)# router ospf '.*'
RP/0/RP0/CPU0:router(config-GRP-ospf)# nsf cisco
RP/0/RP0/CPU0:router(config-GRP-ospf)# packet-size 3000
RP/0/RP0/CPU0:router(config-GRP-ospf)# end-group

RP/0/RP0/CPU0:router(config)# apply-group g-ospf
```

The result is effectively this configuration:

```

router ospf 100
  packet-size 1000
  nsf cisco
```

Note that `packet-size 3000` is not inherited from the configuration group because the explicit local configuration has precedence.

### Compatible Configuration Is Inherited

The configuration in the configuration group must match the configuration on the router to be inherited. If the configuration does not match, it is not inherited. For example, assume that this configuration is running on the router:

```
router ospf 100
  auto-cost disable
!
```

You configure this configuration and commit it to the configuration.

```
RP/0/RP0/CPU0:router(config)# group g-ospf
RP/0/RP0/CPU0:router(config-GRP)# router ospf '.*'
RP/0/RP0/CPU0:router(config-GRP-ospf)# area '.*'
RP/0/RP0/CPU0:router(config-GRP-ospf-ar)# packet-size 2000
RP/0/RP0/CPU0:router(config-GRP-ospf)# end-group

RP/0/RP0/CPU0:router(config)# apply-group g-ospf

RP/0/RP0/CPU0:router(config)# router ospf 200
RP/0/RP0/CPU0:router(config-ospf)# area 1
```

The result is effectively this configuration:

```
router ospf 100
  auto-cost disable

router ospf 200
  area 1
  packet-size 2000
```

The packet size is inherited by the ospf 200 configuration, but not by the ospf 100 configuration because the area is not configured.

## Layer 2 Transport Configuration Group: Example

This example shows how to configure and apply a configuration group with Layer 2 transport subinterfaces:

```
RP/0/RP0/CPU0:router(config)# group g-l2trans-if
RP/0/RP0/CPU0:router(config-GRP)# interface 'TenGigE.*\.*' l2transport
RP/0/RP0/CPU0:router(config-GRP)# mtu 1514
RP/0/RP0/CPU0:router(config-GRP)# end-group

RP/0/RP0/CPU0:router(config)# interface TenGigE0/0/0/0.1 l2transport
RP/0/RP0/CPU0:router(config-if)# apply-group g-l2trans-if
```

When this configuration is committed, the Ten Gigabit Ethernet interface 0/11/0/0.1 inherits the 1514 MTU value. This is the output displayed from the **show running-config inheritance** command for the Ten Gigabit Ethernet interface:



```
interface TenGigE0/11/0/0.1 l2transport
  ## Inherited from group g-l2trans-if
  mtu 1514
!
```

## Configuration Group Precedence: Example

When similar configuration statements are contained in multiple configuration groups, groups applied in inner configuration modes take precedence over groups applied in outer modes. This example shows two configuration groups that configure different cost values for OSPF.

```
RP/0/RP0/CPU0:router(config)# group g-ospf2
RP/0/RP0/CPU0:router(config-GRP)# router ospf '*'
RP/0/RP0/CPU0:router(config-GRP-ospf)# area '*'
RP/0/RP0/CPU0:router(config-GRP-ospf-ar)# cost 2
RP/0/RP0/CPU0:router(config-GRP-ospf-ar)# end-group

RP/0/RP0/CPU0:router(config)# group g-ospf100
RP/0/RP0/CPU0:router(config-GRP)# router ospf '*'
RP/0/RP0/CPU0:router(config-GRP-ospf)# area '*'
RP/0/RP0/CPU0:router(config-GRP-ospf-ar)# cost 100
RP/0/RP0/CPU0:router(config-GRP-ospf-ar)# end-group
```

If these configuration groups are applied as follows, the cost 2 specified in g-ospf2 is inherited by OSPF area 0 because the group is applied in a more inner configuration mode. In this case, the configuration in group g-ospf100 is ignored.

```
RP/0/RP0/CPU0:router(config)# router ospf 0
RP/0/RP0/CPU0:router(config-ospf)# apply-group g-ospf100
RP/0/RP0/CPU0:router(config-ospf)# area 0
RP/0/RP0/CPU0:router(config-ospf-ar)# apply-group g-ospf2
```

## Changes to Configuration Group are Automatically Inherited: Example

When you make changes to a configuration group that is committed and applied to your router configuration, the changes are automatically inherited by the router configuration. For example, assume that this configuration is committed:

```
group g-interface-mtu
  interface 'GigabitEthernet.*'
  mtu 1500
!
end-group

interface POS0/0/0/0
  apply-group g-interface-mtu
!
```

Now you change the configuration group as in this example:

```
RP/0/RP0/CPU0:router(config)# group g-interface-mtu
RP/0/RP0/CPU0:router(config-GRP)# interface 'GigabitEthernet.*'
RP/0/RP0/CPU0:router(config-GRP-if)# mtu 2000
```

```
RP/0/RP0/CPU0:router(config-GRP-if)# end-group
```

When this configuration group is committed, the MTU configuration for interface GigabitEthernet0/0/0/0 is automatically updated to 2000.