Autoconf

Autoconf is a solution that can be used to manage port configurations for data or voice VLAN, quality of service (QoS) parameters, storm control, and MAC-based port security on end devices that are deployed in the access layer of a network.

- Prerequisites for Autoconf, on page 1
- Restrictions for Autoconf, on page 1
- Information About Autoconf, on page 2
- How to Configure Autoconf, on page 7
- Configuration Examples for Autoconf, on page 16
- Feature History for Autoconf, on page 17

Prerequisites for Autoconf

- Before enabling Autoconf, disable the Auto SmartPort (ASP) macro, device classifier, and then access the session monitor.

Restrictions for Autoconf

- Interface templates are not applicable for wireless sessions.
- When the Autoconf feature is enabled using the **autoconf enable** command, the default Autoconf service policy is applied to all interfaces. No other service policy can be applied globally using the **service-policy** command. To apply a different service policy, you must disable Autoconf on that interface. When a service policy is applied globally, you must disable it before enabling the Autoconf feature.
- When both local (interface-level) and global service policies exist, the local policy take precedence. Events in the local service policy are handled and the global service policy is not applied. The global service policy comes into effect only when the local policy is removed.
- Service templates cannot be applied to interfaces, and interface templates cannot be applied to service instances.
- Only one service template can be nested inside an interface template.

Information About Autoconf

Benefits of Autoconf

The Autoconf feature permits hardbinding between the end device and the interface. Autoconf falls under the umbrella of the Smart Operations solution. Smart Operations is a comprehensive set of capabilities that can simplify and improve LAN switch deployment. Smart Operations help organizations deliver operational excellence and scale services on the network.

The Autoconf feature automatically applies the needed configurations on the device ports to enable the efficient performance of each directly connected end device using a set of interface configurations that are configured inside an interface template.

- Autoconf efficiently applies commands to an interface because the parser does not need to parse each command each time.
- Configurations that are applied through the Autoconf feature can be reliably removed from a port without impacting previous or subsequent configurations on the port.
- The Autoconf feature provides built-in and user-defined configurations using interface and service templates. Configurations applied through templates can be centrally updated with a single operation.
- Using the Autoconf feature, a configuration can be applied to ports and access sessions.
- The Autoconf feature reduces ongoing maintenance for devices and attached end devices by making them intuitive and autoconfigurable. This reduces operation expenses (OPEX) and lowers the total cost of ownership (TCO).

Identity Session Management and Templates

A key advantage of the Autoconf feature is that the core session management capability is decoupled from the application-specific logic; thus, allowing the same framework to be used regardless of the criteria for policy determination or the nature of the policies applied.

The identity session management infrastructure allows configurations and/or policies to be applied as templates.

Both service and interface templates are named containers of configuration and policy. Service templates may be applied only to access sessions, while interface templates may be applied only to ports. When a service template is applied to an access session, the contained configuration/policy is applied only to the target session and has no impact on other sessions that may be hosted on the same access port. Similarly, when an interface template is applied to an access port, it impacts all traffic exchanged on the port.

The Autoconf feature uses a set of built-in maps and built-in templates. The built-in templates are designed based on best practices for interface configurations. Built-in templates can be modified by the user to include customized configurations, limiting the need to create a new template.

The templates created by users are referred to as user-defined templates. User-defined templates can be defined on the device and can be mapped to any built-in or user-defined trigger.

Use the **show derived-config** command, to view the overall applied configurations applied by Autoconf template and manual configuration. The interface commands shown in the output of **show running-config interface** *type number* command are not necessarily the operational configuration. The Autoconf feature

dynamically applies a template to the interface, and overrides any conflicting static configuration that is already applied.

Autoconf Operation

Autoconf uses the Device Classifier to identify the end devices that are connected to a port.

The Autoconf feature uses the device classification information gleaned from Cisco Discovery Protocol, LLDP, DHCP, MAC addresses, and the Organizationally Unique Identifier (OUI) that is identified by the Device Classifier.

The Device Classifier provides improved device classification capabilities and accuracy, and increased device visibility for enhanced configuration management.

Device classification is enabled when you enable the Autoconf feature using **autoconf enable** command in global configuration mode.

The device detection acts as an event trigger, which in turn applies the appropriate automatic template to the interface.

The Autoconf feature is based on a three-tier hierarchy.

- A policy map identifies the trigger type for applying the Autoconf feature.
- A parameter map identifies the appropriate template that must be applied, based on the end device.
- The templates contain the configurations to be applied.

The Autoconf built-in templates and triggers perform the these three steps automatically.

The Autoconf feature provides the following built-in templates:

- AP_INTERFACE_TEMPLATE
- DMP_INTERFACE_TEMPLATE
- IP_CAMERA_INTERFACE_TEMPLATE
- IP_PHONE_INTERFACE_TEMPLATE
- LAP_INTERFACE_TEMPLATE
- MSP_CAMERA_INTERFACE_TEMPLATE
- MSP_VC_INTERFACE_TEMPLATE
- PRINTER_INTERFACE_TEMPLATE
- ROUTER_INTERFACE_TEMPLATE
- SWITCH_INTERFACE_TEMPLATE
- TP_INTERFACE_TEMPLATE

Note By default built-in templates are not displayed under running configuration. The built-in templates show in the running configuration only if you edit them.

The template that is selected is based on parameter map information applied to an interface. This information can be based on the following criteria:

- End Device type
- MAC address
- OUI
- User role
- Username

The Autoconf feature provides one built-in parameter map BUILTIN_DEVICE_TO_TEMPLATE with the following configuration:

```
Parameter-map name: BUILTIN_DEVICE_TO_TEMPLATE
 Map: 10 map device-type regex "Cisco-IP-Phone"
  Action(s):
   20 interface-template IP_PHONE_INTERFACE_TEMPLATE
 Map: 20 map device-type regex "Cisco-IP-Camera"
  Action(s):
   20 interface-template IP_CAMERA_INTERFACE_TEMPLATE
 Map: 30 map device-type regex "Cisco-DMP"
  Action(s):
   20 interface-template DMP_INTERFACE_TEMPLATE
 Map: 40 map oui eq "00.0f.44"
  Action(s):
   20 interface-template DMP_INTERFACE_TEMPLATE
 Map: 50 map oui eq "00.23.ac"
  Action(s):
   20 interface-template DMP_INTERFACE_TEMPLATE
 Map: 60 map device-type regex "Cisco-AIR-AP"
  Action(s):
   20 interface-template AP_INTERFACE_TEMPLATE
 Map: 70 map device-type regex "Cisco-AIR-LAP"
  Action(s):
   20 interface-template LAP_INTERFACE_TEMPLATE
 Map: 80 map device-type regex "Cisco-TelePresence"
  Action(s):
   20 interface-template TP_INTERFACE_TEMPLATE
 Map: 90 map device-type regex "Surveillance-Camera"
  Action(s):
   10 interface-template MSP_CAMERA_INTERFACE_TEMPLATE
 Map: 100 map device-type regex "Video-Conference"
  Action(s):
   10 interface-template MSP_VC_INTERFACE_TEMPLATE
```

Note Use the **show parameter-map type subscriber attribute-to-service All** command to view the configuration for the built-in parameter map.

The Autoconf feature provides one built-in policy map BUILTIN_AUTOCONF_POLICY with the following configuration:

```
BUILTIN_AUTOCONF_POLICY
  event identity-update match-all
    10 class always do-until-failure
      10 map attribute-to-service table BUILTIN_DEVICE_TO_TEMPLATE
```

Note Use the **show policy-map type control subscriber BUILTIN_AUTOCONF_POLICY** command to view the configuration for the built-in policy map.

You can also manually create policy maps, parameter maps, and templates.

When a trigger is created that is based on specific user information, a local 802.1X Cisco Identity Services Engine (ISE) server authenticates it ensuring the security of the operation.

An interface template can be dynamically activated (on an interface) using any of the following methods:

- RADIUS CoA—While Change of Authorization (CoA) commands are targeted to one or more access sessions, any referenced template must be applied to the interface hosting the referenced session.
- RADIUS Access-Accept for client authentication or authorization—Any referenced interface template returned in an Access-Accept must be applied to the port that is hosting the authorized access session.
- Service template—If an interface template is referenced in a service template that is either locally defined or sourced from the AAA server, the interface template must be applied to the interface hosting any access-session on which the service template is applied (add a new command for interface template reference from within a locally defined service template).
- Subscriber control-policy action—A mapping action under the subscriber control policy activates service and/or interface template (as referenced in a parameter map) based on the type of filter, and removes any templates associated with a previous policy.
- Device-to-template parameter map—A subscriber parameter map that allows the filter type to service and/or interface template mappings to be specified in an efficient and readable manner.

Advantages of Using Templates

Using templates for autoconfiguration has the following benefits:

- Templates are parsed once when they are being defined. This makes dynamic application of the templates very efficient.
- Templates can be applied to an Ethernet interface that is connected to an end device, based on the type of the end device.
- Service templates allow the activation of session-oriented features, whereas interface templates apply configurations to the interface that is hosting a session.
- Service templates are applied to access sessions and hence only impact the traffic exchanged with a single endpoint on a port.
- Startup and running configurations of the device are not modified by the dynamic application of the template.
- Policy application is synchronized with the access-session life cycle, which is tracked by the framework by using all available techniques, including just link-up/link-down.
- Templates can be updated with a single operation. All applied instances of the templates are updated.
- Constituent commands of the templates do not appear in the running configuration.

- Templates can be removed with no impact on previous or subsequent configurations.
- Template application is acknowledged, allowing for synchronization and performing remedial actions where failures occur.
- Data VLAN, quality of service (QoS) parameters, storm control, and MAC-based port security are configured automatically based on the end device that is connected to the switch.
- The switch port is cleaned up completely by removing configurations when the device is disconnected from a port.
- Human error is reduced in the installation and configuration process.

Autoconf Functionality

The Autoconf feature is disabled by default in global configuration mode. When you enable the Autoconf feature in global configuration mode, it is enabled by default at the interface level. The built-in template configurations are applied based on the end devices detected on all interfaces.

Use the **access-session inherit disable autoconf** command to manually disable Autoconf at the interface level, even when Autoconf is enabled at the global level.

If you disable Autoconf at the global level, all interface-level configurations are disabled.

Global	Interface Level	AutoConf Status
Disable	Disable	No automatic configurations are applied when an end device is connected.
Enable	Enabled by default	If Autoconf is enabled at the global level, it is enabled at the interface level by default. Built-in template configurations are applied based on the end devices that are detected on all interfaces.
Enable	Disable	Enabled at global level. Disabled at interface level. No automatic configurations are applied when an end device is connected to the interface on which Autoconf is disabled.

Autoconf allows you to retain the template even when the link to the end device is down or the end device is disconnected, by configuring the Autoconf sticky feature. Use the **access-session interface-template sticky** command to configure the Autoconf sticky feature in global configuration mode. The Autoconf sticky feature avoids the need for detecting the end device and applying the template every time the link flaps or device is removed and connected back.

The **access-session interface-template sticky** command is mandatory to apply an inbuilt template that contains **access-session** commands on an interface. Configure the **access-session interface-template sticky** command to apply interface template on a port using a service policy.

If you want to disable the Autoconf feature on a specific interface, use the **access-session inherit disable interface-template-sticky** command in interface configuration mode.

How to Configure Autoconf

Applying a Built-in Template to an End Device

The following task shows how to apply a built-in template on an interface that is connected to an end device, for example, a Cisco IP phone.

Before you begin

Make sure that the end device, for example, a Cisco IP phone, is connected to the switch port.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device(config)# configure terminal`	Enters global configuration mode.
Step 3	**autoconf enable** **Example:** `Device(config)# autoconf enable`	Enables the Autoconf feature.
Step 4	**end** **Example:** `Device(config)# end`	Exits global configuration mode and enters privileged EXEC mode.
Step 5	(Optional) **show device classifier attached interface** *interface-type interface-number* **Example:** `Device# show device classifier attached interface Gi3/0/26`	Displays whether the end device is classified by the device classifier with correct attributes.
Step 6	**show template binding target** *interface-type interface-number* **Example:** `Device# show template binding target gi3/0/26`	Displays the configuration applied through the template on the interface.

Verifying th device classification of an End Device

Verifying the Interface Template on an Interface

Verifying the Interface Configuration

Verifying Interface Configuration for Cisco IOS 4500 Series, Cisco IOS 3650 Series, Cisco IOS 3560 Series, and Cisco IOS 2960 Switches

Verifying Global Configuration after Applying Autoconf

The following example shows that an IP phone is classified by the Device Classifier with correct attributes:

```
Device# show device classifier attached interface GigabitEthernet 3/0/26

Summary:

MAC_Address       Port_Id       Profile Name                        Device Name
==============    ==========    ==================                  ==================
0026.0bd9.7bbb    Gi3/0/26      Cisco-IP-Phone-7962                 Cisco IP Phone 7962
```

The following example shows that a built-in interface template is applied on the interface:

```
Device# show template binding target GigabitEthernet 3/0/26

Interface Templates
===================
Interface: Gi4/0/11
Method              Source            Template-Name
------              ------            -------------
dynamic             Built-in          IP_PHONE_INTERFACE_TEMPLATE
```

The following example shows how to verify the interface configuration after the interface template is applied to the IP phone connected to the GigabitEthernet interface 3/0/26 :

```
Device# show running-config interface GigabitEthernet 3/0/26
Building configuration...

Current configuration : 624 bytes
!
interface GigabitEthernet3/0/26
!
End

Device# show derived-config interface GigabitEthernet 3/0/26

Building configuration...

Derived configuration : 649 bytes
!
interface GigabitEthernet3/0/26
 switchport mode access
 switchport block unicast
 switchport port-security maximum 3
 switchport port-security maximum 2 vlan access
 switchport port-security violation restrict
 switchport port-security aging time 2
 switchport port-security aging type inactivity
 switchport port-security
```

```
 load-interval 30
 storm-control broadcast level pps 1k
 storm-control multicast level pps 2k
 storm-control action trap
 spanning-tree portfast
 spanning-tree bpduguard enable
 service-policy input AutoConf-4.0-CiscoPhone-Input-Policy
 service-policy output AutoConf-4.0-Output-Policy
 ip dhcp snooping limit rate 15
end
```

The following example shows how to verify the interface configuration:

```
Device# show template interface source built-in all
Building configuration...

Template Name       : AP_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 switchport mode trunk
 switchport nonegotiate
 mls qos trust cos
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : DMP_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security
 mls qos trust dscp
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : IP_CAMERA_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security
 mls qos trust dscp
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : IP_PHONE_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security maximum 3
 switchport port-security maximum 2 vlan access
 switchport port-security violation  restrict
 switchport port-security aging time 2
 switchport port-security aging type inactivity
 switchport port-security
 storm-control broadcast level pps 1k
 storm-control multicast level pps 2k
 storm-control action trap
```

```
 mls qos trust cos
 service-policy input AUTOCONF-SRND4-CISCOPHONE-POLICY
 ip dhcp snooping limit rate 15
 load-interval 30
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : LAP_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security violation  protect
 switchport port-security aging time 2
 switchport port-security aging type inactivity
 switchport port-security
 storm-control broadcast level pps 1k
 storm-control multicast level pps 2k
 storm-control action trap
 mls qos trust dscp
 ip dhcp snooping limit rate 15
 load-interval 30
 srr-queue bandwidth share 10 10 60 20
 priority-queue out
!
Template Name       : MSP_CAMERA_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security
!
Template Name       : MSP_VC_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport block unicast
 switchport port-security violation  restrict
 switchport port-security aging time 2
 switchport port-security aging type inactivity
 switchport port-security
 ip dhcp snooping limit rate 15
 load-interval 30
!
Template Name       : PRINTER_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport port-security maximum 2
 switchport port-security
 load-interval 60
!
Template Name       : ROUTER_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast trunk
```

```
 spanning-tree bpduguard enable
 switchport mode trunk
 mls qos trust dscp
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : SWITCH_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 switchport mode trunk
 mls qos trust cos
 srr-queue bandwidth share 1 30 35 5
 priority-queue out
!
Template Name       : TP_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
 spanning-tree portfast
 spanning-tree bpduguard enable
 switchport mode access
 switchport port-security maximum 3
 switchport port-security maximum 2 vlan access
 switchport port-security violation  restrict
 switchport port-security aging time 2
 switchport port-security aging type inactivity
 switchport port-security
 storm-control broadcast level pps 1k
 storm-control multicast level pps 2k
 storm-control action trap
 ip dhcp snooping limit rate 15
 load-interval 30
!
```

```
Device# show running config
class-map match-any AutoConf-4.0-Scavenger-Queue
 match dscp cs1
 match cos  1
 match access-group name AutoConf-4.0-ACL-Scavenger
class-map match-any AutoConf-4.0-VoIP
 match dscp ef
 match cos  5
class-map match-any AutoConf-4.0-Control-Mgmt-Queue
 match cos  3
 match dscp cs7
 match dscp cs6
 match dscp cs3
 match dscp cs2
 match access-group name AutoConf-4.0-ACL-Signaling
class-map match-any AutoConf-4.0-Multimedia-Conf
 match dscp af41
 match dscp af42
 match dscp af43
class-map match-all AutoConf-4.0-Broadcast-Vid
 match dscp cs5
class-map match-any AutoConf-4.0-Bulk-Data
 match dscp af11
 match dscp af12
 match dscp af13
class-map match-all AutoConf-4.0-Realtime-Interact
 match dscp cs4
class-map match-any AutoConf-4.0-VoIP-Signal
 match dscp cs3
 match cos  3
class-map match-any AutoConf-4.0-Trans-Data-Queue
 match cos  2
```

```
 match dscp af21
 match dscp af22
 match dscp af23
 match access-group name AutoConf-4.0-ACL-Transactional-Data
class-map match-any AutoConf-4.0-VoIP-Data
 match dscp ef
 match cos  5
class-map match-any AutoConf-4.0-Multimedia-Stream
 match dscp af31
 match dscp af32
 match dscp af33
class-map match-all AutoConf-4.0-Internetwork-Ctrl
 match dscp cs6
class-map match-all AutoConf-4.0-VoIP-Signal-Cos
 match cos  3
class-map match-any AutoConf-4.0-Multimedia-Stream-Queue
 match dscp af31
 match dscp af32
 match dscp af33
class-map match-all AutoConf-4.0-Network-Mgmt
 match dscp cs2
class-map match-all AutoConf-4.0-VoIP-Data-Cos
 match cos  5
class-map match-any AutoConf-4.0-Priority-Queue
 match cos  5
 match dscp ef
 match dscp cs5
 match dscp cs4
class-map match-any AutoConf-4.0-Bulk-Data-Queue
 match cos  1
 match dscp af11
 match dscp af12
 match dscp af13
 match access-group name AutoConf-4.0-ACL-Bulk-Data
class-map match-any AutoConf-4.0-Transaction-Data
 match dscp af21
 match dscp af22
 match dscp af23
class-map match-any AutoConf-4.0-Multimedia-Conf-Queue
 match cos  4
 match dscp af41
 match dscp af42
 match dscp af43
 match access-group name AutoConf-4.0-ACL-Multimedia-Conf
class-map match-all AutoConf-4.0-Network-Ctrl
 match dscp cs7
class-map match-all AutoConf-4.0-Scavenger
 match dscp cs1
class-map match-any AutoConf-4.0-Signaling
 match dscp cs3
 match cos  3
!
!
policy-map AutoConf-4.0-Cisco-Phone-Input-Policy
 class AutoConf-4.0-VoIP-Data-Cos
  set dscp ef
  police cir 128000 bc 8000
   exceed-action set-dscp-transmit cs1
   exceed-action set-cos-transmit 1
 class AutoConf-4.0-VoIP-Signal-Cos
  set dscp cs3
  police cir 32000 bc 8000
   exceed-action set-dscp-transmit cs1
   exceed-action set-cos-transmit 1
```

```
 class class-default
  set dscp default
  set cos 0
policy-map AutoConf-4.0-Output-Policy
 class AutoConf-4.0-Scavenger-Queue
  bandwidth remaining percent 1
 class AutoConf-4.0-Priority-Queue
  priority
  police cir percent 30 bc 33 ms
 class AutoConf-4.0-Control-Mgmt-Queue
  bandwidth remaining percent 10
 class AutoConf-4.0-Multimedia-Conf-Queue
  bandwidth remaining percent 10
 class AutoConf-4.0-Multimedia-Stream-Queue
  bandwidth remaining percent 10
 class AutoConf-4.0-Trans-Data-Queue
  bandwidth remaining percent 10
    dbl
 class AutoConf-4.0-Bulk-Data-Queue
  bandwidth remaining percent 4
    dbl
 class class-default
  bandwidth remaining percent 25
    dbl
policy-map AutoConf-DMP
 class class-default
  set dscp cs2
policy-map AutoConf-IPVSC
 class class-default
  set cos dscp table AutoConf-DscpToCos
policy-map AutoConf-4.0-Input-Policy
 class AutoConf-4.0-VoIP
 class AutoConf-4.0-Broadcast-Vid
 class AutoConf-4.0-Realtime-Interact
 class AutoConf-4.0-Network-Ctrl
 class AutoConf-4.0-Internetwork-Ctrl
 class AutoConf-4.0-Signaling
 class AutoConf-4.0-Network-Mgmt
 class AutoConf-4.0-Multimedia-Conf
 class AutoConf-4.0-Multimedia-Stream
 class AutoConf-4.0-Transaction-Data
 class AutoConf-4.0-Bulk-Data
 class AutoConf-4.0-Scavenger
```

Applying a Modified Built-in Template to an End Device

The following task shows how to modify a built-in template when multiple wireless access points and IP cameras are connected to a switch.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device(config)# configure terminal`	Enters global configuration mode.
Step 3	**template** *template-name* **Example:** `Device(config)# template AP_INTERFACE_TEMPLATE`	Enters template configuration mode for the builtin template.
Step 4	**switchport access vlan** *vlan-id* **Example:** `Device(config-template)# switchport access vlan 20`	Sets the VLAN when the interface is in access mode.
Step 5	**description** *description* **Example:** `Device(config-template)# description modifiedAP`	Modifies the description of the built-in template.
Step 6	**exit** **Example:** `Device(config-template)# exit`	Exits template configuration mode and enters global configuration mode.
Step 7	**autoconf enable** **Example:** `Device(config)# autoconf enable`	Enables the Autoconf feature.
Step 8	**end** **Example:** `Device(config)# end`	Exits global configuration mode and enters privileged EXEC mode.
Step 9	**show template interface binding all** **Example:** `Device# show template interface binding all`	Displays whether the template is applied on the interface.

Verifying the Device classification of an End Device

Verifying the Interface Template on an Interface

The following example shows that the IP camera and access points are classified by the Device Classifier with correct attributes:

```
Device# show device classifier attached detail

DC default profile file version supported = 1
```

```
Detail:
MAC_Address     Port_Id    Cert Parent Proto    ProfileType     Profile Name
Device_Name
==============================================================================================
001d.a1ef.23a8  Gi1/0/7     30    3    C     M    Default      Cisco-AIR-AP-1130      cisco
 AIR-AP1131AG-A-K9
001e.7a26.eb05  Gi1/0/30    70    2    C     M    Default      Cisco-IP-Camera        Cisco
 IP Camera
```

The following example shows that a built-in interface template is applied on the interface:

```
Device# show template interface binding all

Template-Name                   Source            Method          Interface
-------------                   ------            ------          ---------
IP_CAMERA_INTERFACE_TEMPLATE     Built-in          dynamic         Gi1/0/30
AP_INTERFACE_TEMPLATE          Modified-Built-in   dynamic         Gi1/0/7
```

Migrating from ASP to Autoconf

Before you begin

Verify that the AutoSmart Port (ASP) macro is running using the **show running-config | include macro auto global** command.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**no macro auto global processing** **Example:** `Device(config)# no macro auto global processing`	Disables ASP on a global level.
Step 4	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 5	**clear macro auto configuration all** **Example:**	Clears macro configurations for all interfaces.

	Command or Action	Purpose
	`Device# clear macro auto configuration all`	
Step 6	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 7	**autoconf enable** **Example:** `Device(config)# autoconf enable`	Enables the Autoconf feature.
Step 8	**end** **Example:** `Device(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.

Configuration Examples for Autoconf

Example: Applying a Built-in Template to an End Device

The following example shows how to apply a built-in template to an end device connected to an interface.

```
Device> enable
Device(config)# configure terminal
Device(config)# autoconf enable
Device(config)# end
Device# show device classifier attached interface Gi3/0/26
Device# show template binding target GigabitEthernet 3/0/26
```

Example: Applying a Modified Built-in Template to an End Device

The following example shows how to modified built-in template and verify the configuration:

```
Device> enable
Device(config)# configure terminal
Device(config)# template AP_INTERFACE_TEMPLATE
Device(config-template)# switchport access vlan 20
Device(config-template)# description modifiedAP
Device(config-template)# exit
Device(config)# autoconf enable
Device(config)# end
Device# show template interface binding all
```

Example: Migrating from ASP Macros to Autoconf

The following example shows how to migrate from ASP to Autoconf:

```
Device> enable
Device# configure terminal
Device(config)# no macro auto global processing
Device(config)# exit
Device# clear macro auto configuration all
Device# configure terminal
Device(config)# autoconf enable
Device(config)# end
```

Feature History for Autoconf

This table provides release and related information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Table 1: Feature History for Autoconf

Release	Feature Name	Feature Information
Cisco IOS XE Everest 16.5.1a	Autoconf	Autoconf is a solution that can be used to manage port configurations for data or voice VLANs, quality of QoS parameters, storm control, and MAC-based port security on end devices that are deployed in the access layer of a network. The Autoconf feature automatically applies the configurations needed on the device ports to enable the efficient performance of each directly connected end device using a set of interface configurations that are configured inside an interface template. This mechanism ensures that no configurations are needed from the end device. Support for this feature was introduced only on the C9500-12Q, C9500-16X, C9500-24Q, C9500-40X models of the Cisco Catalyst 9500 Series Switches
Cisco IOS XE Fuji 16.8.1a	Autoconf	Support for this feature was introduced only on the C9500-32C, C9500-32QC, C9500-48Y4C, and C9500-24Y4C models of the Cisco Catalyst 9500 Series Switches.
Cisco IOS XE Cupertino 17.7.1	Autoconf	This feature was implemented on the C9500X-28C8D model.

Use the Cisco Feature Navigator to find information about platform and software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com.