



Managing Advanced DHCP Properties

This chapter describes how to set up some of the more advanced DHCP server properties. Before clients can use DHCP for address assignment, you must add at least one scope (dynamic address pool) to the server. This is described in [Chapter 11, “Configuring DHCP Scopes and Policies.”](#)

Configuring BOOTP

BOOTP (the BOOTstrap Protocol) was originally created for loading diskless computers. It was later used to allow a host to obtain all the required TCP/IP information to use the Internet. Using BOOTP, a host can broadcast a request on the network and get information required from a BOOTP server. The BOOTP server is a computer that listens for incoming BOOTP requests and generates responses from a configuration database for the BOOTP clients on that network. BOOTP differs from DHCP in that it has no concept of lease or lease expiration. All IP addresses that a BOOTP server allocates are permanent.

You can configure Cisco CNS Network Registrar to act like a BOOTP server. In addition, although BOOTP normally requires static address assignments, you can choose to either reserve IP addresses (and, therefore, use static assignments) or have IP addresses dynamically allocated for BOOTP clients.

About BOOTP

When you configure the DHCP server to return a BOOTP packet, be aware that BOOTP requires information in the DHCP packet in fields other than the option space. BOOTP devices often need information in the bootfile (*file*), server’s IP address (*siaddr*), and server’s host name (*sname*) fields of the DHCP packet (see RFC 2131).

Every Network Registrar DHCP policy has fields where you can configure the information you want returned directly in the *file*, *siaddr*, or *sname* fields. The Network Registrar DHCP server also supports a configuration parameter with which you can configure the policy options and determine which of the *file*, *sname*, or *siaddr* values you want returned to the BOOTP device.

Network Registrar supports an analogous configuration parameter with which you can configure the options and *file*, *sname*, or *siaddr* values you want returned to the DHCP client. This is in addition to any options requested by the DHCP clients in the *dhcp-parameter-request* option in the DHCP request. Thus, you can configure both the BOOTP and DHCP response packets appropriately for your devices.

Step 1 Decide the values that you want in the BOOTP packet reply fields:

- *file*—Name of the bootfile
- *siaddr*—Server’s IP address

- *sname*—Optional server host name
- Step 2** Decide the list of options and their values that you want returned to the BOOTP client.
- Step 3** Set these values in the policy you want associated with the BOOTP request:
- Fields—*Packet-siaddr*, *packet-file-name*, *packet-server-name* to send to the BOOTP client.
 - Option values, such as the server addresses and domain name to return to the BOOTP client.
 - List of fields and options you want returned to the BOOTP client.
- Step 4** Enable the associated scope or scopes for BOOTP processing.
- Step 5** Enable dynamic BOOTP processing if you want to have this scope provide an address for any BOOTP client that requests one. If you do not enable dynamic BOOTP, you must make reservations for each BOOTP client for which you want this scope to provide an address.
-

Enabling BOOTP for Scopes

You can enable BOOTP processing for a scope. Set certain attributes and BOOTP reply options for a created policy in the local cluster Web UI, or use the **policy name create** and **policy name set** commands in the CLI, to configure BOOTP. Set the policy attributes and options as a comma-separated list. The attributes are entities to use in a client's boot process:

- *packet-siaddr*—IP address of the next server
- *packet-file-name*—Name of the boot file
- *packet-server-name*—Host name of the server

The server looks through the *bootp-reply-options* list to find any matches for these attributes. In the local cluster Web UI, click the options in the BOOTP Compatible section of the Options drop-down list. The options are the same in the local cluster Web UI and CLI.

After setting attributes, set the correct option values.

In the local cluster Web UI, click and set the options from the Options drop-down list and in the attributes.

In the CLI, the **policy name setOption** command requires spaces (not equal signs) before values. Also, enable BOOTP and dynamic BOOTP, if desired, and ensure that the DHCP server updates the DNS server with BOOTP requests. The options are:

- Set the option *dhcp-lease-time*.
- Enable the *dynamic-bootp* attribute.
- Enable the *update-dns-for-bootp* attribute.
- Enable the *update-dns-for-bootp* attribute.

Moving or Decommissioning BOOTP Clients

When you move or decommission a BOOTP client, you can reuse its lease. To decommission a BOOTP client, you must remove its lease reservation from the scope and force its lease to be available.

Force the lease available in the local cluster Web UI, or set the **scope name removeReservation** and **lease ipaddr force-available** commands in the CLI.

Using Dynamic BOOTP

When you use dynamic BOOTP, there are additional restrictions placed on the address usage in scopes, because BOOTP clients are allocated IP addresses permanently and receive leases that never expire.

If you are using DHCP failover, when a server whose scope does not have the *dynamic-bootp* option enabled goes into PARTNER-DOWN state, it can allocate any available IP address from that scope, no matter whether it was initially available to the main or backup server. However, when the *dynamic-bootp* option is enabled, the main server and backup servers can only allocate their own addresses.

Consequently scopes that enable the *dynamic-bootp* option require more addresses to support failover.

When using dynamic BOOTP:

1. Segregate dynamic BOOTP clients to a single scope. Disable DHCP clients from using that scope. In the local cluster Web UI, under the BOOTP attributes for the scope, disable the *dhcp* attribute. In the CLI, use the **scope name disable dhcp** command.
2. If you are using DHCP failover, set the *failover-dynamic-bootp-backup-percentage* attribute for the DHCP server to allocate a greater percentage of addresses to the backup server for this scope. This percentage can be as much as 50 percent higher than a regular backup percentage.

BOOTP Relay

Any router that supports BOOTP relay usually has an address that points to the DHCP server. For example, if you are using a Cisco router, it uses the term *IP helper-address*, which contains an address for a specific machine. In this case, use this address to forward all BOOTP (and therefore DHCP) broadcast packets. Be sure that you configure this address on the router closest to your host.



Tip

If your clients do not receive addresses, the router might be configured with another DHCP server, where the Network Registrar DHCP server cannot see it. If so, configure a second helper address so that the server can also respond to the packets.

Setting DHCP Forwarding

You can use Network Registrar to forward DHCP traffic from one DHCP server to another. For example, you might want to redirect address requests from certain clients, with specific MAC address prefixes, to another DHCP server. This can be useful and important in situations where the server being forwarded to is not one that you manage. This occurs in environments where multiple service providers supply DHCP services for clients on the same virtual LAN.

Enabling DHCP forwarding requires implementing an extension script. The DHCP server intercepts the specified clients and calls its forwarding code, which checks the specified list of forwarded server addresses. It then forwards the requests rather than process them itself. You attach and detach extensions to and from the DHCP server using the **dhcp attachExtension** and **dhcp detachExtension** commands.

You can forward DHCP traffic from one DHCP server to another under the control of a Network Registrar extension. The DHCP forwarding attribute is important in situations where the other server is not one that you manage. This is most likely to occur in environments where multiple vendors supply DHCP services for clients on the same virtual LAN.

The DHCP forwarding attribute works like this:

1. When DHCP is initialized, the server opens a UDP socket, which it uses to send forwarded packets. To support servers with multiple IP addresses, the socket address pair consists of INADDR_ANY and any port number. This enables clients to use any one of the server's IP addresses.
2. When the DHCP server receives a request from a client, it processes these extension point scripts:
 - post-packet-decode
 - pre-client-lookup
 - post-client-lookup

As the DHCP server processes these scripts, it checks the environment dictionary for this string:

```
cnr-forward-dhcp-request
```

3. When it finds that string and it has the value *true* (enabled), the server calls its forwarding code.
4. The forwarding code checks the environment dictionary for a string with this key:

```
cnr-request-forward-address-list
```

It expects a list of comma-separated IP addresses with an optional colon-delimited port number, as in this example:

```
192.168.168.15:1025,192.168.169.20:1027
```

By default, the server forwards to port 67. It sends a copy of the entire client request to each IP address and port in turn. If any element in the list is invalid, the server stops trying to parse the list.

5. After the forwarding code returns, the server stops processing the request. In the post-client-lookup extension point script, however, this might create an optional log message with client-entry details.

The following example of a portion of a TCL extension script tells the DHCP server to forward a request to another server based on the information in the request. You can use such a script if there are multiple device provisioning systems in the same environment. In this case, you would run the extension script on the DHCP server to which routers forward broadcast requests. The script would determine which (if any) other server or servers should handle the request, and tell the original server to forward the request.

The sample script uses a static mapping of MAC address prefix to send modems from a specific vendor to a specific system:

```
proc postPktDecode {req resp env} {
    set mac [$req get chaddr]
    set addr ""
    ;# Very simple, static classifier that forwards all requests from devices
    ;# with a vendor-id of 01:0c:10 to the DHCP servers at 10.1.2.3 and 10.2.2.3:
    switch -glob -- $mac {
        01:0c:10* {
            set addr "10.1.2.3,10.2.2.3"
        }
    }
    ;# If we decide to forward the packet, the $addr var will have the IP addresses
    ;# where to forward the packet:
    if {$addr != ""} {
        ;# Tell the DHCP server to forward the packet...
        $env put cnr-forward-dhcp-request true
        ;# ...and where to forward it:
        $env put cnr-request-forward-address-list $addr
        ;# No more processing is required.
        return
    }
}
```

A more flexible script could use a per-client configuration object, such as the Network Registrar client entry, to indicate which DHCP server should get the request.

Setting Vendor-Specific DHCP Options

You can send vendor-specific option data to accommodate DHCP clients that request them. The server sends vendor-encapsulated options in DHCP option 43. Each vendor has a special syntax for the data part of the option. The Network Registrar Web UI does not support option 43, but the CLI does.

There are four main steps (and their commands) in configuring vendor-specific options for a client:

1. Create the vendor-specific option—**vendor-option name create**.
2. Create and define any vendor-specific data types—**option-datatype name create** and **defineField**.
3. Define the required suboptions with the data types—**vendor-option name defineSuboption**.
4. Set the vendor option values inside a policy—**policy name setVendorOption**.

Step 1 See your DHCP client device vendor's manual for the device's class identifier string that the client sends in DHCP option 60. Use this string in the **vendor-option name create** command. The following command creates the `device1_vso` vendor option, using the exact string that the vendor uses for the device's class identifier. The class identifier must be unique to each vendor option:

```
nrcmd> vendor-option device1_vso create "device1:Arch:xxxxxx:UNDI:yyzzz"
```

Step 2 Identify the format of the DHCP option 43 data the DHCP client device requires. The format could range from a simple IP address to several suboptions. Each suboption has a data type and corresponding data. Network Registrar arranges the suboption data in sequence, using the suboption number, to form the packet returned to the client. The suboptions can have standard DHCP data types or more complex vendor-specific ones.

List all the suboption numbers and their data types. If you use standard data types, such as BYTE, STRING, and INT, go to [Step 4](#). If you use multiple or more complex vendor data types, you must define an option-datatype. For example, `suboption_8` holds an array of boot server addresses available to the device and requires a special array data type.

First create the *option-datatype* using the **option-datatype name create** command. Then, use **option-datatype name defineField** command to define the data type fields. For the example of the `device1_suboption_8` data type to be applied to `suboption_8`, the first field, `boot_server_type`, has a WORD data type. The second field, `boot_server_IP_list`, has an IPADDR data type in a counted array. The *enable read-only* attribute in the last command prevents further changes to the definition:

```
nrcmd> option-datatype device1_suboption_8 create
nrcmd> option-datatype device1_suboption_8 defineField boot_server_type 1 WORD
nrcmd> option-datatype device1_suboption_8 defineField boot_server_IP_list 2 IPADDR_ARRAY
counted-array
nrcmd> option-datatype device1_suboption_8 enable read-only
```

Note that you can only use these complex *option-datatypes* with the **vendor-option** command in the next step, not with the **custom-option** command.

Step 3 Confirm your settings by showing or listing the data type or its fields. If necessary, undefine or redefine a field, or delete the data type and start over again. (If undefining or redefining fields, first disable *read-only* for the suboption.) Here are the commands:

```
nrcmd> option-datatype device1_suboption_8 show
nrcmd> option-datatype list
nrcmd> option-datatype listnames
```

```
nrcmd> option-datatype device1_suboption_8 listFields
nrcmd> option-datatype device1_suboption_8 disable read-only
nrcmd> option-datatype device1_suboption_8 undefineField boot_server_type
nrcmd> option-datatype device1_suboption_8 undefineField boot_server_IP_list
nrcmd> option-datatype device1_suboption_8 delete
```

Step 4 Based on the suboption numbers you listed, use the **vendor-option name defineSuboption** command to define each suboption and map it to its appropriate data type. If the suboption has a standard data type, you can use a simpler version of the command with the standard data type specified:

```
nrcmd> vendor-option standard_type create "device1:Arch:xxxxxx:UNDI:yyyaaa"
nrcmd> vendor-option standard_type defineSuboption suboption_1 1 IPADDR
```

In the device_1_vso example, however, the client expects the server to return several suboptions as an array. In this case, define suboption_8 with the special device1_suboption_8 *option-datatype* you created in [Step 2](#). (Because you deleted the suboption in the previous step, you need to recreate it.) The *array* flag allows setting the multiple instances of the suboption in [Step 5](#):

```
nrcmd> option-datatype device1_suboption_8 create
nrcmd> vendor-option device1_vso defineSuboption suboption_8 8 device1_suboption_8 array
```

Note that although some DHCP clients do not request suboptions, you should still define a single suboption, at number 1. Then, use the *no-suboption-opcode* flag to prevent adding additional ones. This example uses the standard STRING data type for the suboption:

```
nrcmd> vendor-option no_opt create "device1:Arch:xxxxxx:UNDI:yyyxxx"
nrcmd> vendor-option no_opt defineSuboption suboption_1 1 STRING no-suboption-opcode
```

You can also undefine a suboption:

```
nrcmd> vendor-option no_opt undefineSuboption suboption_1
```

Step 5 Set each suboption value in a policy. Use the **policy name setVendorOption** command to set the values and **policy name unsetVendorOption** command to unset them. For created suboptions with standard data types, you can set the suboption directly with a value. Then, list the vendor options:

```
nrcmd> policy default setVendorOption standard_type suboption_2 "string-data"
nrcmd> policy default listVendorOptions
```

In the more complex array example, you need to include the array index value. The array definition requires the special braced and square-bracketed suboption-index syntax, *{suboption[index]}*:

```
nrcmd> policy network-1.2.3 setVendorOption device1_vso {suboption_8[0]}
boot_server_type 2
nrcmd> policy network-1.2.3 setVendorOption device1_vso {suboption_8[0]}
boot_server_IP_list 192.168.25.4,192.168.25.5
nrcmd> policy network-1.2.3 setVendorOption device1_vso {suboption_8[1]}
boot_server_type 8
nrcmd> policy network-1.2.3 setVendorOption device1_vso {suboption_8[1]}
boot_server_IP_list 192.168.25.6
```

In the example, the network-1.2.3 policy sets these suboption_8 array element values for device1_vso:

- Array element 0 boot_server_type field—Type 2 (Microsoft Windows boot server)
- Array element 0 boot_server_IP_list field—List of Windows boot server addresses
- Array element 1 boot_server_type field—Type 8 (HP OpenView boot server)
- Array element 1 boot_server_IP_list field—List of OpenView boot server addresses

- Step 6** Show or list the results. If they are not what you expected, delete the vendor option and start over again. Vendor options are write-enabled by default. You can change each vendor option to be read-only:

```
nrcmd> vendor-option listnames
nrcmd> vendor-option list
nrcmd> vendor-option standard_type show
nrcmd> vendor-option no_opt delete
nrcmd> vendor-option standard_type enable read-only
```

Defining Advanced Server Parameters

You can set advanced DHCP server parameters, including custom DHCP options.

Setting Advanced DHCP Server Parameters

Table 14-1 describes the advanced DHCP server parameters that you can set in the local cluster Web UI and CLI.

Table 14-1 DHCP Advanced Parameters

Advanced Parameter	Action	Description
<i>max-dhcp-requests</i>	set/ unset	<p>Controls the number of buffers that the DHCP server allocates for receiving packets from DHCP clients and failover partners. If this setting is too large, a burst of DHCP activity can clog the server with requests that become stale before being processed. This results in an increasing processing load that can severely degrade performance as clients try to obtain a new lease. A low buffer setting throttles requests and could affect server throughput.</p> <p>In a nonfailover deployment, the default setting is sufficient. In a failover deployment, you can increase it to 1000 if the DHCP logs indicate a consistently high number of request buffers in use. You should then also modify the number of DHCP responses (see the <i>max-dhcp-responses</i> parameter) to four times the request buffers.</p> <p>When using LDAP client lookups, buffers should not exceed the LDAP lookup queue size defined by the total number of LDAP connections and the maximum number of requests allowed for each connection. Set the LDAP queue size to match the LDAP server's capacity to service client lookups.</p> <p>Required. The default is 500.</p>

Table 14-1 DHCP Advanced Parameters (continued)

Advanced Parameter	Action	Description
<i>max-dhcp-responses</i>	set/ unset	Controls the number of response buffers that the DHCP server allocates for responding to DHCP clients and performing failover communication between DHCP partners. In a non-failover deployment, the default setting of twice the number of request buffers is sufficient. In a failover deployment, you can increase this so that it is four times the number of request buffers. In general, increasing the number of response buffers is not harmful, while reducing it to below the previously recommended ratios might be harmful to server responsiveness. Required. The default is 1000.
<i>max-ping-packets</i>	set/ unset	If you enable the <i>Ping address before offering it</i> option at the scope level, packet buffers are used to send and receive ICMP messages. If you enable pinging, you should have enough ping packets allocated to handle the peak load of possible ping requests. The default is 500 ping packets.
<i>hardware-unicast</i>	enable/ disable	Controls whether the DHCP server sends unicast rather than broadcast responses when a client indicates that it can accept a unicast. This feature is only available on Windows NT, Windows 2000, and Solaris platforms; other operating systems broadcast instead. The default is enabled.
<i>defer-lease-extensions</i>	enable/ disable	Controls whether the DHCP server extends leases that are less than half expired. The default is checked or true. This means that a client renewing a lease less than halfway through it can only get the remaining part of it and not be extended. See the “Deferring Lease Extensions” section on page 14-9.
<i>last-transaction-time-granularity</i>	set/ unset	Specifies the number of seconds that guarantees the last transaction time to be accurate. Do not set this lower than 30 seconds. For optimal performance, set it to a value that is greater than half the lease interval. There is no default setting.

In the local cluster Web UI:

-
- Step 1** On the Primary Navigation bar, click **DHCP**.
 - Step 2** On the Secondary Navigation bar, click **DHCP Server**.
 - Step 3** Click the name of the server. This opens the Edit DHCP Server page.
 - Step 4** Add or modify attributes on this page.
 - Step 5** Click **Modify Server** to make the changes.
-

In the CLI, use the **dhcp show** and **dhcp get** commands to show the current server parameters, then use the **dhcp set**, **dhcp unset**, **dhcp enable**, and **dhcp disable** commands to change them (see [Table 14-1](#)).

Deferring Lease Extensions

The *defer-lease-extensions* attribute allows the DHCP server to optimize its response to a sudden flood of DHCP traffic. The parameter is enabled by default. An example of a network event that could result in such a traffic spike is a power failure at a cable internet service provider (ISP) data center that results in all of its cable modem termination systems (CMTS) rebooting at once. If this were to happen, the devices attached to the CMTSs produce a flood of DHCP traffic as they quickly come back online.

When you enable the *defer-lease-extensions* attribute, a DHCP client that renews a lease that is less than halfway to its expiration will not have its lease extended to the full lease time. Instead, the client receives a lease corresponding to the remaining time on its existing lease. Because the absolute lease expiration time does not change, the server can avoid database updates that result in a significantly higher server throughput.

If a client is more than halfway to its expiration, this setting has no effect, and the lease is extended to the full configured lease interval as usual, including the database writes.



Note

This feature significantly increases the server's performance while remaining in compliance with the DHCP RFC, which stipulates that client binding information is committed to persistent storage when the lease changes.

These three specific situations are described from the server's point of view:

- **Client retries**—When the server gets behind, it is possible for a client to retransmit requests. The DHCP server does not maintain enough information to recognize these as retransmissions, and processes each to completion, granting a full lease duration again and updating the database. When the server is already behind, doing extra work worsens the situation. To prevent this, the DHCP server does not extend leases that are less than 30 seconds old, regardless of the state of the *defer-lease-extensions* attribute.
- **Client reboots**—The effective renew time for a client's lease is really the minimum of the configured renew time and the time between client reboots. In many installations this may mean that clients get fresh leases one (in a typical enterprise) or two (in a typical cable network) times per day, even if the renew time is set for many days. Setting the *defer-lease-extensions* attribute can prevent these early renews from causing database traffic.
- **Artificially short renewal times**—Because there is no way for a DHCP server to proactively contact a DHCP client with regard to a lease, you might configure fairly short client renewals to provide a means of doing network renumbering, address re-allocation, or network reconfiguration (for example, a change in DNS server address) in a timely fashion. The goal is to allow you to do this without incurring unacceptable database update overhead.

As a complication, the server also keeps track of the time when it last heard from the client. Known as the last transaction time, sites sometimes use this information as a debugging aid. Maintaining this time robustly requires a write to the database on every client interaction. The *last-transaction-time-granularity* attribute is the one to set. It is the time, in seconds, that Network Registrar guarantees that the last transaction time is accurate.

Do not set this granularity lower than the default of 60 seconds. For optimal performance, set it to a value that is greater than half of your lease interval. Because the last transaction time is not integral to the protocol, you need not update it synchronously. Also, because it is primarily a debugging aid, it need not be entirely accurate. Furthermore, because the in-memory copy is always accurate, you can use the **export leases -server** command to display the current information, even if the data is not up-to-date in the database.

Configuring Custom DHCP Options

Along with assigning values to predefined DHCP options, you can create your own custom options.

Adding Custom Options

You can add a custom option to a policy and assign or edit its value. This function is not available in the Web UI. In the CLI, use the **custom-option name create** command to create a custom option. For example, to create the option myoption, mapped to number 100 and of type BYTE, enter:

```
nrcmd> custom-option myoption create 100 BYTE desc="custom option 100"
```



Tip

Do not map numbers to options that DHCP or BOOTP already uses. For a list of pre-assigned numbers, see [Appendix B, “DHCP Options.”](#) Ensure that the option number is the same as requested by the client.

Use the **custom-option name [show]** command to show an option’s values, and the **custom-option list** command to show all the custom options, or list just the names. Use the **custom-option name get** command to show individual properties of the custom option. You can also unset a custom option.

Editing and Removing Custom Options

You can edit or remove custom options.



Caution

Changing custom option properties or deleting the option altogether can have unexpected side effects on policies. If you delete a custom option, also remove it from the policies that include it.

This function is not available in the Web UI. In the CLI, use the **custom-option name set** command to change the option type (*opttype*) or description (*desc*). To change a custom option’s number, you must delete the option and recreate it with the new number. Use the **custom-option name delete** command to delete an option. After you delete a custom option, also unset it from all policies that include it by using the **policy name unsetOption** command.

Integrating Windows System Management Servers

You can have the DHCP server interact with the Microsoft System Management Server (SMS) so that SMS is current with DHCP changes. Normally, SMS pulls updated data through a DHCPDISCOVER request from the server about any new clients that joined the network. Network Registrar, however, pushes these updates to SMS when you use the **dhcp updateSms** command. Before you do, check if the:

- SMS client installation and initialization step is complete.
- Network Registrar Server Agent is set to run under a login account with sufficient privileges.
- SMS site ID is correct and matches that of the SMS server.

These steps describe how to integrate Windows SMS into Network Registrar.

Step 1 Install the Microsoft BackOffice 4.5 Resource Kit on the same machine as the Network Registrar DHCP server. Follow the installation instructions and choose the default settings.

- Step 2** After the installation, modify the User Variable search path on the Environment tab of the System control panel to:
- ```
\program files\ResourceKit\SMS\diagnose
```
- Step 3** If the DHCP and SMS servers are on different machines, install the SMS client on the same machine as the DHCP server. The SMS library has the necessary API calls to communicate with the SMS server. You must assign the correct site code from the DHCP server machine. In your Network Neighborhood, go to the path `\\SMS-servername\SMSLOGON\x86.bin\00000409\smsman.exe`.
- Run the program and follow the instructions, using the default settings. The program creates two icons that you can use later from the control panel, marked SMS and Remote Control.
- Step 4** Stop and then restart the Network Registrar server agent under a trusted domain account with sufficient privileges. Both the DHCP and SMS servers must be aware of this account. Use this short procedure:
- Stop the local cluster server agent process.
  - Configure the account under which the Network Registrar services run. Create an account name that is a member of both the trusted SMS site server group and a member of the DHCP server machine's administrator group, with the corresponding password.
  - Restart the local cluster server agent process.
- Step 5** Use the `dhcp set sms-library-path` command (or the `sms-library-path` attribute under the Microsoft Systems Management Server category on the Edit DHCP Server page of the local cluster Web UI) to configure the DHCP server to push lease information to SMS. Include the full path to the SMSRsGen.dll. If you omit a value, the path defaults to the internal server default location of this file.
- ```
nrcmd> dhcp set sms-library-path /conf/dll
```
- When you install the Microsoft BackOffice Resource Kit, the system path is not updated to reflect the location of the SMS data link library (DLL). Use one of these methods to configure this attribute:
- Set the attribute to the relative path. Add this line to the system path on the DHCP server machine:


```
sms-install-directory\diagnose
```

 Then, set the `sms-library-path` attribute to the name of the DLL, such as `smsrsgen.dll`. You can also accept the system default by unsetting the attribute.
 - Set the attribute to the absolute path. If you do not want to change the system path, set this attribute to the absolute path of the DLL location:


```
"\\Program Files\Resource Kit\sms\diagnose\smsrsgen.dll"
```
- Step 6** Turn SMS network discovery on by setting the `sms-network-discovery` DNS attribute to 1. If you use the default of 0, you disable SMS network discovery.
- Step 7** Enter the SMS site code that you entered in [Step 3](#) by setting the `sms-site-code` DHCP server attribute. The default string is empty, but for data discovery to be successful, you must provide the site code.
- Step 8** Enter the SMS lease interval value by setting the `sms-lease-interval` attribute. The lease interval is the time between sending addresses to SMS, or how long, in milliseconds, the DHCP server should wait before pushing the next lease to the SMS server when you execute the `server dhcp updateSms` command. Early versions of the SMSRsGen.dll file (SMS Version 2.0) did not allow SMS to reliably receive multiple updates within a one-second window (1000 ms); the default value, therefore, was set to 1.1 second (1100 ms). If you install a future version of the Microsoft BackOffice Resource Kit, which might contain an enhanced version of the SMSRsGen.dll file, then reduce this interval or set it to 0 to increase performance.
- Step 9** Reload the DHCP server and check the `name_dhcp_1_log` file for a successful load.

- Step 10** In the CLI, use the `server dhcp updateSms` command to initiate SMS processing. This command can take an optional *all* keyword to send all leased addresses from the DHCP server to SMS. If you omit this keyword, the DHCP server sends only new leases activated since the last time the command ran. (If you reload the DHCP server while processing the command, reloading does not resume until after you restart the DHCP server). Then, verify that both the DHCP and SMS logs indicate successful completion.

Using Extensions to Affect DHCP Server Behavior

Network Registrar provides the ability to alter and customize the operation of the DHCP server through *extensions*, programs that you can write in TCL or C/C++.

For example, you might have an unusual routing hub that uses BOOTP configuration. This device issues a BOOTP request with an Ethernet hardware type (1) and MAC address in the *chaddr* field. It then sends out another BOOTP request with the same MAC address, but with a hardware type of Token Ring (6). The DHCP server normally distinguishes between a MAC address with hardware type 1 and one with type 6, and considers them to be different devices. In this case, you might want to write an extension that prevents the DHCP server from handing out two different addresses to the same device.

You can solve the problem of the two IP addresses by writing either of these extensions:

- One that causes the DHCP server to drop the Token Ring (6) hardware type packet.
- One that changes the Token Ring packet to an Internet packet and then switches it back again on exit. Although this extension would be more complex, the DHCP client could thereby use either return from the DHCP server.

You can write extensions in TCL or C/C++:

- **TCL**—Makes it a bit easier and quicker to write an extension. If the extension is short, the interpreted nature of TCL does not have a serious effect on performance. When you write an extension in TCL, you are less likely to introduce a bug that can crash the server.
- **C/C++**—Provides the maximum possible performance and flexibility, including communicating with external processes. However, the complexity of the C/C++ API is greater and the possibility of a bug in the extension crashing the server is more likely than with TCL.

You create extensions at specific *extension points*. Extension points include three types of dictionaries—request, response, and environment. Each extension point can use one or more of these dictionaries. The extension points are:

1. **init-entry**—Extension point that the DHCP server calls when it configures or unconfigures the extension. This occurs when starting, stopping, or reloading the server. This entry point has the same signature as the others for the extension. Dictionaries used: environment only.
2. **post-packet-decode**—Used to rewrite the input packet. Dictionaries used: request and environment.
3. **post-class-lookup**—Used to evaluate the result of a *client-class-lookup-id* operation on the client-class. Dictionaries used: request and environment.
4. **pre-client-lookup**—Used to affect the client being looked up, possibly by preventing the lookup or supplying data that overrides the existing data. Dictionaries used: request and environment.
5. **post-client-lookup**—Used to review the operation of the client-class lookup process, such as examining the internal server data structures filled in from the client-class processing. You can also use it to change any data before the DHCP server does additional processing. Dictionaries used: request and environment.

6. **check-lease-acceptable**—Used to change the results of the lease acceptability test. Do this only with extreme care. Dictionaries used: request, response, and environment.
7. **lease-state-change**—Used to determine when the lease state changes this only with extreme care. Dictionaries used: response and environment.
8. **pre-packet-encode**—Used to change the data sent back to the DHCP client in the response, or change the address to which to send the DHCP response. Dictionaries used: request, response, and environment.
9. **pre-dns-add-forward**—Used to alter the name used for the DNS forward (A record) request. Dictionaries used: environment only.
10. **post-send-packet**—Used after sending a packet for processing that you want to perform outside of the serious time constraints of the DHCP request-response cycle. Dictionaries used: request, response, and environment.

To extend the DHCP server, do the following:

Step 1 Write the extension in Tcl, C or C++ and install it in the server extensions directory, on:

- UNIX:
 - Tcl—`/opt/nwreg2/extensions/DHCP/tcl`
 - C or C++—`/opt/nwreg2/extensions/DHCP/dex`
- Windows:
 - For Tcl—`\program files\Network Registrar\extensions\dhcp\tcl`
 - C or C++—`\program files\Network Registrar\extensions\dhcp\dex`

It is best to place these extensions in the appropriate directory for TCL or C/C++ extensions. Then, when configuring the filename, just enter the filename itself, without slash (/) or backslash (\).

If you want to place extensions in subdirectories, enter the filename with a path separator. These are different depending on the operating system on which your DHCP server is running.



Note When entering a filename that contains a backslash (\) character on Windows NT, you must enter it with a double-backslash (\\), because backslash (\) is an escape character in the CLI. For example, enter the filename `debug\myextension.tcl` as **`debug\\myextension.tcl`**.

Step 2 Use the **extension** command to configure the DHCP server to recognize this extension.

Step 3 Attach the configured extension to one or more DHCP extension points using the **dhcp attachExtension** command.

Step 4 Reload the server.

Troubleshooting DHCP Servers

Be sure to follow any server property changes with a reload.

Other helpful hints in tuning your DHCP performance include:

- Set the request (*max-dhcp-requests*) and response (*max-dhcp-responses*) buffers for optimal throughput. See [Table 14-1 on page 14-7](#) for details.

- Keep the *defer-lease-extensions* attribute enabled. This reduces writes to the database.
- Set the *last-transaction-time-granularity* attribute to at least 60 seconds, optimally a value greater than half your lease interval.
- Disable the *allow-lease-time-override* attribute for policies offering production leases.
- Choose from the DHCP log settings to give you greater control over existing log messages. Use the *log-settings* attribute for the DHCP server with one or more of the attributes described in [Table 14-2](#).

Table 14-2 DHCP Log Settings

Log Setting (Numeric Equivalent)	Description
default (1)	Displays basic DHCP activity logging (the default setting).
incoming-packets (2)	Logs a separate line for each incoming DHCP packet (the default).
missing-options (3)	Displays missing policy options expected by a client (the default).
incoming-packet-detail (4)	The same as <i>incoming-packets</i> , but in human-readable form.
outgoing-packet-detail (5)	Logs each incoming DHCP packet in a human-readable form.
unknown-criteria (6)	Logs whenever a client entry has a <i>selection-criteria</i> or <i>selection-criteria-excluded</i> that is not found in any scope appropriate for that client's current network location.
dns-update-detail (7)	Logs each sent and replied DNS update.
client-detail (8)	After every client-class client lookup operation, logs the composite of the data found for the client and its client-class. Useful when setting a client-class configuration and debugging problems in client-class processing.
client-criteria-processing (9)	Logs whenever a scope is examined to find an available lease or to determine if a lease is still acceptable for a client who already has one. Can be very useful when configuring or debugging client-class scope criteria processing. (Causes moderate amount of information to be logged and should not be left enabled as a matter of course.)
failover-detail (10)	Logs detailed failover activity.
ldap-query-detail (11)	Logs whenever the DHCP server initiates a query to an LDAP server, receives a response, and retrieves a result or error messages.
ldap-update-detail (12)	Logs whenever the DHCP server initiates an update lease state to the LDAP server, receives a response, and retrieves a result or error messages.
ldap-create-detail (13)	Logs whenever the DHCP server initiates a lease state entry create to the LDAP server, receives a response, and retrieves a result or error messages.
leasequery (14)	Logs a message for every ACK- or NAK-responded lease query packet.
dropped-waiting-packets (15)	If the value of <i>max-waiting-packets</i> is non-zero, packets can be dropped if the queue length for any IP address exceeds the value. If <i>dropped-waiting-packets</i> is set, the server logs whenever it drops a waiting packet from the queue for an IP address.
no-success-messages (16)	Inhibits logging successful outgoing response packets.
no-dropped-dhcp-packets (17)	Inhibits logging dropped DHCP packets.

Table 14-2 DHCP Log Settings (continued)

Log Setting (Numeric Equivalent)	Description
no-dropped-bootp-packets (18)	Inhibits logging dropped BOOTP packets.
no-failover-activity (19)	Inhibits logging normal activity and some warning messages logged for failover. However, serious error log messages continue to appear.
activity-summary (20)	Enables logging a summary message every five minutes (useful if the following no- type flags are set), showing the activity in the previous interval (you can adjust this interval using the dhcp set-activity-summary-interval command).
no-invalid-packets (21)	Inhibits logging invalid packets.
no-reduce-logging-when-busy (22)	Inhibits reducing logging when receive buffers reach 66%.
no-timeouts (23)	Inhibits logging time-outs of leases and offers.
minimal-config-info (24)	Reduces the number of configuration messages in the log.
no-failover-conflict (25)	Logs conflicts between failover partners.

- Adjust the *mcd-blobs-per-bulk-read* attribute value to tune DHCP start and reload times. Generally, a higher *mcd-blobs-per-bulk-read* attribute value results in faster server start and reload times, at the cost of using more memory. Values can be set to any number between 1 and 2500 using the *mcd-blobs-per-bulk-read* DHCP server attribute. The current default is 256 blobs.

