



# CHAPTER 16

## Using Cisco Access Registrar Server Features

---

**Revised: March 20, 2009, OL-17222-03**

This chapter provides information about how to use the following Cisco Access Registrar (CAR) server features:

- [“Incoming Traffic Throttling” section on page 16-2](#)
- [“Backing Store Parsing Tool” section on page 16-3](#)
- [“Configurable Worker Threads Enhancement” section on page 16-4](#)
- [“Session-Key Lookup” section on page 16-5](#)
- [“Query-Notify” section on page 16-5](#)
- [“Support for Windows Provisioning Service” section on page 16-9](#)
- [“Command Completion” section on page 16-12](#)
- [“Service Grouping Feature” section on page 13](#)
- [“SHA-1 Support for LDAP-Based Authentication” section on page 20](#)
- [“Dynamic Attributes” section on page 16-22](#)
- [“Tunneling Support Feature” section on page 16-24](#)
- [“xDSL VPI/VCI Support for Cisco 6400” section on page 16-25](#)
- [“Apply Profile in Cisco Access Registrar Database to Directory Users” section on page 16-26](#)
- [“Directory Multi-Value Attributes Support” section on page 16-28](#)
- [“MultiLink-PPP \(ML-PPP\)” section on page 16-28](#)
- [“Dynamic Updates Feature” section on page 16-29](#)
- [“NAS Monitor” section on page 16-31](#)
- [“Automatic Information Collection \(arbug\)” section on page 16-31](#)
- [“Simultaneous Terminals for Remote Demonstration” section on page 16-32](#)
- [“Support for RADIUS Check Item Attributes” section on page 16-32](#)
- [“User-Specific Attributes” section on page 16-34](#)
- [“Packet of Disconnect” section on page 16-34](#)
- [“Dynamic DNS” section on page 16-39](#)
- [“Dynamic Service Authorization Feature” section on page 16-42](#)

# Incoming Traffic Throttling

CAR offers two options to tackle traffic bursts by limiting incoming traffic. You will find two properties, `MaximumIncomingRequestRate` and `MaximumOutstandingRequests`, under `/Radius/Advanced` to limit the incoming traffic.

## MaximumIncomingRequestRate

You can use the `MaximumIncomingRequestRate` property to limit incoming traffic in terms of “allowed requests per second”.

For example, if you set the `MaximumIncomingRequestRate` to  $n$ , then at any given second, only  $n$  requests are accepted for processing. In the next second, another  $n$  requests are accepted regardless of whether the requests accepted earlier are processed or not. This condition serves as a soft limit.

The `MaximumIncomingRequestRate` property by default is zero (disabled).

## MaximumOutstandingRequests

You can use the `MaximumOutstandingRequests` property to limit incoming traffic in terms of “requests processed”.

For example, if you set the `MaximumOutstandingRequests` to  $n$ ,  $n$  requests are accepted for processing. Further requests are accepted only after processing some of these requests and sending the replies back. This condition serves as a hard limit.

The `MaximumOutstandingRequests` property by default is zero (disabled).



### Note

You can enable either of these properties independent of the other.

You must follow the steps outlined below to configure the `MaximumIncomingRequestRate` or `MaximumOutstandingRequests` property:

- 
- Step 1** Log in to `aregcmd`.
- Step 2** Change directory to `/Radius/Advanced`.
- Step 3** Set the `MaximumIncomingRequestRate` or `MaximumOutstandingRequests` property to non-zero values.
- ```
set MaximumIncomingRequestRate n
```
- or
- ```
set MaximumOutstandingRequests n
```
- where  $n$  is any nonzero value.
- Step 4** Save the configuration; enter:
- ```
save
```
- Step 5** Reload the server; enter:
- ```
reload
```
-

# Backing Store Parsing Tool

CAR tool, **carbs.pl**, helps to analyze the session backing store files. You will find this tool under **/cisco-ar/bin** directory.

Using **carbs.pl**, you can:

- Get information about the active, stopped, and stale Radius sessions.
- Clear phantom sessions manually.
- Process the binary log files and get information in a user-readable format.

The syntax is:

**carbs.pl [-a] [-d <dir>] [-f <logfile>] [-v] [p] [-o <output>] [-h]**

-a—All session statistics (active, stale, stopped)

-d—<Directory> Default: .

-f—<Filename> Default: 00\*.log

-v—verbose Default: off

-p—Clear phantom sessions

-o—<Filename> Output log to TEXT

-h—Help, usage

Table 16-1 lists the options available with **carbs.pl** and their description.

**Table 16-1 Carbs.pl Options and Description**

Option	Description
-d<directory>	Optional. Accepts a directory as parameter with no trailing slash. You can use this option to change the default directory to scan for BackingStore log files. Default is current directory.
-f<logfile>	Optional. Accepts a logfile as parameter with no leading or trailing slashes. You can use this option to change the default log files. Allows you to enter individual logfile name as well as wildcard characters surrounded by single quotes.
-v	Optional. No parameters. You can use this option to get total session count and phantom session count.
-p	Optional. No parameters. Generates a list of phantom sessions. You can use this option to clear the stale sessions.
-o	Optional. Accepts <output file> as parameter. You can use this option to convert BackingStore log files to readable files and write the results to the output file specified.

Table 16-1 Carbs.pl Options and Description (continued)

Option	Description
-a	Optional. No parameters. You can use this option to print all session statistics, such as per-NAS stale session count, total active sessions, and total stale sessions.
-h	You can use this option to get help with usage of carbs.pl.

## Configurable Worker Threads Enhancement

CAR provides a configurable variable you can use to increase the number of worker threads to handle a greater number of RADIUS packets during peak operating periods. This variable controls the processing of greater number of RADIUS packets than expected during peak operating periods.

The variable, RADIUS\_WORKER\_THREAD\_COUNT, is found in the **arserver** file under **/cisco-ar/bin/arserver** and controls the number of worker threads the CAR server creates. You can increase the number of worker threads to help make more efficient use of the server's CPU.



### Note

Before you increase the setting for RADIUS\_WORKER\_THREAD\_COUNT, you should be certain that you are running into a worker thread starvation issue. If you use scripts that consume a lot of processing and memory, you might run out of memory if you create too many worker threads.

Increasing the number of worker threads also increases memory utilization.

The default value of RADIUS\_WORKER\_THREAD\_COUNT for servers running a Solaris operating system is 256. The default value for servers running Red Hat Enterprise Linux (RHEL) is 64.

The purpose of this enhancement is to take advantage of spare CPU bandwidth which was not being used in earlier releases of CAR due to a lower number of worker threads. At times, the worker threads would be stuck doing work that took a long time to complete, like running a script. Having more threads will help mitigate these situations and will help improve on the latency created due to lack of free worker threads.



### Note

Before modifying the RADIUS\_WORKER\_THREAD\_COUNT variable, consult with a TAC representative to ensure that modifying the RADIUS\_WORKER\_THREAD\_COUNT is warranted.

To modify the RADIUS\_WORKER\_THREAD\_COUNT variable:

- Step 1** Log in to the CAR server as a root user and change directory to **/cisco-ar/bin**.
- Step 2** Use a text editor and open the **arserver** file.
- Step 3** Locate the line with the RADIUS\_WORKER\_THREAD\_COUNT variable.
 

```
#change this to configure number of worker threads
RADIUS_WORKER_THREAD_COUNT=256
```
- Step 4** Modify the number of RADIUS worker threads to the number you choose.

**Note**

There is no upper limit to the number of RADIUS worker threads you can enable in your CAR server, but you should take care not to exceed your server's memory capacity.

**Step 5** Save the file and restart the CAR server.

## Session-Key Lookup

The Session-Key Lookup feature enables you to identify the Session Manager and Session Key of an existing session based on certain attributes associated with that session, such as the Mobile Station Integrated Services Digital Network (MSISDN) number.

The Session-Key Lookup feature required the following enhancements to CAR software:

- Enabling a query service to be invoked for Ascend-IP-Allocate packets
- Enabling the setting of the Session-Key and Session-Manager environment variables by a query operation
- Performing session management after the query operation
- A new environment variable, `Set-Session-Mgr-And-Key-Upon-Lookup`, which when set to TRUE causes a session-cache Resource Manager to set the Session-Manager and Session-Key environment variables during the query lookup.

The Session-Key Lookup feature is useful in a scenario where an existing session requires an update from an incoming Ascend-IPA-Allocate packet (from a different NAS or device) with modified authorization attributes. Note that this Ascend-IPA-Packet might not have the exact set of attributes as the original packet that created the session. However, the Ascend-IPA-Allocate packet must contain at least one attribute that can uniquely identify the session (such as the MSISDN number) and should contain the same Username of the original session.

The Session-Key Lookup feature works in tandem with the Radius Query feature, where a Radius Query service is defined with the unique attribute (such as the MSISDN number) as the query-key and is configured to query all session managers. The Query-Service environment variable is set to the defined Radius Query service and the new environment variable (`Set-Session-Mgr-And-Key-Upon-Lookup`) is set to TRUE for this Ascend-IPA-Allocate packet. This triggers a query operation on all the live sessions. If there is a match, the Session-Manager and Session-Key of that session is used for subsequent session management. During session management, the session cache is updated with the modified authorization attributes.

The Session-Manager OutgoingScript (or any outgoing script that executes after the Session-Manager Outgoing Script) should not reject the packet when doing a Session-Key lookup. Doing so causes the session to be deleted.

## Query-Notify

The Query-Notify feature enables you to store information about Wireless Application Protocol (WAP) gateways that have queried for User Identity-IP Address mapping and send appropriate messages to the WAP gateway when the subscriber logs out of the network.

CAR has been enhanced to update the session cache with the attribute-value pairs of an interim accounting update packet. This ensures that CAR server provides the most up-to-date information to the WAP gateway during the proxy of interim records or query of the session cache.

CAR has been enhanced to also notify the WAP gateways that have queried a session with Interim accounting update packets. If a WAP gateway does not respond to the Interim accounting update packets, the CAR server times out and retries by notifying the WAP gateways again. If there is no response after all the retries, the proxy packet is deleted and no change is made to the session or the WAP gateway's state in the CAR server. You can configure the number of retries under **/Radius/Clients/notificationproperties**.

The accounting response packet from the CAR server to the GPRS Gateway Support Node (GGSN) is independent of the proxy operation to the WAP gateways. The accounting response packet is sent back immediately without waiting for responses from the WAP gateways.

The Query-Notify feature also enables you to quarantine IP addresses for a configurable amount of time if a WAP gateway does not respond to Accounting-Stop packets sent by the CAR server.

The CAR server stores information about clients (usually the IP address) that queried for particular user information and send RADIUS Accounting-Stop packets to those clients when the CAR server receives the Accounting-Stop packet. There is no intermediate proxy server between the CAR server and the WAP gateway.

To support the Query-Notify feature, the CAR server's *radius-query* service has been modified to also store information like the IP address about the clients queried for cached information. The information is stored in the user session record along with the cached information so it is available after a server reload.

To use the Query-Notify feature, you must make the following configuration changes:

- 
- Step 1** Configure the Clients object under **/Radius/Clients**.
  - Step 2** Set the EnableNotifications property to TRUE.  
The EnableNotifications property indicates that a client can receive Accounting-Stop notifications from the CAR server. When EnableNotifications is set to TRUE, a sub-directory named NotificationProperties appears in client object configuration.
  - Step 3** Configure the properties under the client's NotificationProperties subdirectory.  
See [Clients, page 4-5](#), for information about how to configure these properties.
  - Step 4** Configure a list of attributes to store under **/Radius/Advanced/Attribute Groups/<Notification Group>** where *<notification group>* is the name of an Attribute Group containing a list of attributes to be stored.
- 

## Call Flow

This section describes the call flow of the Query-Notify feature.

1. The CAR server caches information from an from Accounting-Start.  
This information is usually from a GGSN when a subscriber enters into the network.
2. When a WAP gateway receives a request to authenticate a subscriber, it queries the CAR server using an Access-Request packet to retrieve the cached information for that subscriber.

3. The CAR server responds with Access-Accept if an entry is found for the subscriber in its cache; otherwise the server returns an Access-Reject.

The CAR server sends an Access-Accept packet to the WAP gateway. The list of attributes sent in this Access-Accept will depend on radius-query service configuration.




---

**Note** You use **aregcmd** to configure the attributes for the Access-Accept packet in the AttributesToBeReturned subdirectory under a radius-query service type.

---

4. If the CAR server finds a cache entry for the subscriber, it checks to see if the EnableNotifications property for that client. If EnableNotifications is set to TRUE, the CAR server stores the client IP address in the subscriber's cache.
5. If the CAR server receives an Accounting-Interim-Update packet from the GGSN, it responds by sending an Accounting-Response packet then sends the Accounting-Interim-Update packets to all the queried clients of the WAP Gateways.

If the WAP gateway queried clients do not respond to the Accounting-Interim-Update packets, the CAR server times out and retries by notifying the WAP gateways again. If there is no response after all the retries, the proxy packet is deleted and no change is made to the session or the WAP gateway's state in the CAR server. The StaleSessionTimeout property under **/Radius/Advanced** is not applicable for Accounting-Interim-Update packets.

6. When the subscriber logs out of the network, the CAR server receives an Accounting-Stop packet and responds by sending an Accounting-Response back to the client.

Before releasing the subscriber's session, the CAR server looks for any client IP addresses in the subscriber's cache. If it finds any, the CAR server sends Accounting-Stop packets to those clients with the attributes configured in the NotificationAttributeGroup subdirectory for each client.

The CAR server forms the attributes with those attributes in the session cache and from the Accounting-Stop packet. The CAR server uses the value configured for the Port property in the NotificationProperties subdirectory as the destination port for the Accounting-Stop packet and uses the client's shared secret.

The CAR server then waits for Accounting-Response packets from each client to which it has sent Accounting-Stop packets. The CAR server waits for the time interval configured in the InitialTimeout property configured in the NotificationProperties subdirectory before sending another Accounting-Stop packet. If it does not receive an Accounting-Response packet, the CAR server sends additional Accounting-Stop packets until the number of attempts reaches the value configured in the MaxTries property in the NotificationProperties subdirectory.

7. When the CAR server receives an Accounting-Response packet from each client, the server releases the subscriber session.

If the CAR server does not receive Accounting-Response packets from all clients after the configured time and attempts, the server maintains the subscriber session for the time interval configured in the StaleSessionTimeout property in **/Radius/Advanced** then releases the subscriber session.

The CAR server maintains the subscriber session to address the quarantine IP address requirement. The CAR server must quarantine IP addresses if a WAP gateway does not respond to Accounting-Stop sent by the CAR server. The length of time an IP address is quarantined depends on the value of the InitialTimeOut property under the **NotificationProperties** subdirectory of **/Radius/Clients/wap\_gateway**.

8. If the `StaleSessionTimeout` property is `TRUE` for a subscriber session, the CAR server rejects any query requests from clients for this session cache. After the `StaleSessionTimeout` expires, the CAR server will again send `Accounting-Stop` to all the clients listed in the session and proceeds to delete this subscriber session regardless of the status of the `Accounting-Stop`.

## Configuration Examples



### Note

In addition to the following configuration, the `StaleSessionTimeout` property must be set in **/Radius/Advanced**. This property has a default value of 1 hour.

The following shows an example configuration for a Query-Notify client:

```
[ //localhost/Radius/Clients/wap-gateway1 ]
Name = wap-gateway1
Description =
IPAddress = 10.100.10.1
SharedSecret = secret
Type = NAS
Vendor =
IncomingScript~ =
OutgoingScript~ =
EnableDynamicAuthorization = FALSE
NetMask =
EnableNotifications = TRUE
NotificationProperties/
  Port = 1813
  InitialTimeout = 5000
  MaxTries = 3
  NotificationAttributeGroup = notifyGroup
```

The following shows an example configuration for a Query-Notify AttributeGroup:

```
[ //localhost/Radius/Advanced/AttributeGroups/notifyGroup ]
Name = notifyGroup
Description =
Attributes/
  1. User-Name
  2. Acct-Session-Id
  3. NAS-Identifier
  4. NAS-Port
```

## Memory and Performance Impact

Using the Query-Notify feature will have the following effects:

- There will be a memory impact because the CAR server caches IP addresses of clients queried in the session record.
- There will be an impact on performance because the CAR server has to persist the cached IP address information before responding to **radius-query** requests.

# Support for Windows Provisioning Service

CAR supports Microsoft's Windows Provisioning Service (WPS). WPS provides hotspot users with seamless service to public WLAN hotspots by using Microsoft Windows-based clients. The Microsoft WPS solution requires Microsoft-based software in the data center for the RADIUS server and the provisioning server.

## Call Flow

The following is the WPS process and Wireless Internet Service Provider (WISP) packet sequence for a new wireless client login at a Wi-Fi hotspot location.

1. The client discovers the WISP network at a Wi-Fi hotspot.
2. The client authenticates as guest (with null username and credentials) to the CAR server.
3. The client is provisioned and a new account is created.
4. The client is authenticated using the new account credentials and accesses the Internet.

The CAR server performs the following functions during WPS:

1. Detects the guest subscriber login from the null username and null credentials during PEAPv0 (MS-PEAP) authentication.
2. Grants a successful login and returns a *sign-up* URL of the provisioning server as a PEAP-Type-Length-Value (TLV) in the next Access-Challenge Packet.

The following is an example value for the URL PEAP-TLV:

```
http://www.example.com/provisioning/master.xml#sign up
```

Where *#sign up* is the parameter for this action and is a required element of the value.

The sign-up URL value is passed when the user authenticates as guest. The sign-up URL is a fragment within the Master URL. You can also configure other fragments to be returned in the Master URL. See [Master URL Fragments, page 16-10](#), for more information about the different fragments.

3. Sends a VLAN-ID or IP filter (or both) in the final Access-Accept packet to restrict the guest user's accessibility to only the Provisioning server.
4. Authenticates using the user configuration in the user database after the client is provisioned and a new account is created.

## Example Configuration

The following shows an example configuration for the WPS feature:

```
[ //localhost/Radius/Services/peapv0 ]
  Name = peapv0
  Description =
  Type = peap-v0
  IncomingScript~ =
  OutgoingScript~ =
  MaximumMessageSize = 1024
  PrivateKeyPassword = <password>
  ServerCertificateFile = <path_to_ServerCertificateFile>
  ServerRSAKeyFile = <path_to_ServerRSAKeyFile>
  CACertificateFile = <path_to_CACertificateFile>
```

```

CACertificatePath =<path_to_CACertificatePath>
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = True
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120
TunnelService = eap-mschapv2
EnableWPS = True
MasterURL = http://www.example.com/provisioning/master.xml
WPSGuestUserProfile = WPS-Guest-User-Profile

```

When you set the EnableWPS property to TRUE, you must provide values for the properties MasterURL and WPSGuestUserProfile. See [Environment Variables](#) for more information.

## Environment Variables

The two environment variables that is used to support WPS:

- [Send-PEAP-URI-TLV](#)
- [Master-URL-Fragment](#)

### Send-PEAP-URI-TLV

Send-PEAP-URI-TLV is a Boolean value and provides a way in which the authenticating user service can let the PEAP-V0 service know to send the URI PEAP-TLV along. Under different circumstances CAR might send back different fragments within the MasterURL to the client, as described above.

The conditions under which this has to be sent is best known to the user authentication service (the service that is specified within the eap-mschapv2 service, which in turn is the tunnel service for PEAP-V0 service). So when it decides that it needs to send back the URL it can set this variable to TRUE. The default value for this is FALSE.

### Master-URL-Fragment

The CAR authenticating user service uses Master-URL-Fragment to set the fragment within the Master URL that needs to be sent back. The CAR user authentication service sets the fragment to different values under different circumstances. While the Send-PEAP-URL-TLV indicates whether to send the URL or not, Master-URL-Fragment is used to intimate which fragment within the URL needs to be sent. If this variable is not set and if it is required to send the URL, '#signup' will be sent by default.

## Master URL Fragments

This section describes the different fragments the RADIUS server might send to the AP in the Master URL.

### Sign up

This value is passed when the user authenticates as guest. The following is an example value for the URL PEAP-TLV:

```
http://www.example.com/provisioning/master.xml#sign up
```

where #sign up is the parameter for this action and a required element of the value.

## Renewal

This value is passed when the user's account is expired and needs renewal before network access can be granted. The following is an example value for the URL PEAP-TLV:

```
http://www.example.com/provisioning/master.xml#renewal
```

where #renewal is the parameter for this action and a required element of the value.

## Password change

This value is passed when the user is required to change the account password. An example value for the URL PEAP-TLV is:

```
http://www.example.com/provisioning/master.xml#passwordchange
```

where #passwordchange is the parameter for this action and a required element of the value.

## Force update

This value is passed when the WISP requires the Wireless Provisioning Services on the client to download an updated XML master file. This method of updating the XML master file on the client should be used only to correct errors; otherwise, the TTL expiry time in the XML master file is used to provide background updates. The following is an example value for the URL PEAP-TLV:

```
http://www.example.com/provisioning/master.xml#forceupdate
```

where #forceupdate is the parameter for this action and a required element of the value.

# Unsupported Features

The following features are part of the Microsoft WPS functionality, but are not supported in the CAR.

## Account Expiration and Renewal

When the user creates an account and logs in with that account, the RADIUS server authenticates and authorizes the request and sends back an Access-Accept with a Session-Timeout attribute. The Access Point (AP) then forces the wireless client to reauthenticate for every timeout value. When there is one timeout duration left in the user account, the RADIUS server needs to send back a *renewal* URL (a URL fragment within the master URL) to the client for the user to renew the account.

CAR does not support this feature because the interface the CAR server has with the AD (through the CiscoSecure Remote Agent) does not have provisions to get the expiration information of user account. However, this release does provide an environment variable to copy the URL fragment and to control whether or not to send the URL using another environment variable. This can be used to send the renewal URL. There are some limitations, however.

## Password Changing and Force Update

The Password Changing option is passed when the user is required to change the account password. Force Update option is passed when the WISP requires the Wireless Provisioning Services on the client to download an updated XML master file.

These functions are not possible in this release for the same reason mentioned above, the loose coupling between CAR and the AD. Additionally, there is no known use case for this. As mentioned above, you can use the newly added environment variables to trigger these options.

# Command Completion

CAR's command completion feature provides online help by listing possible entries to the current command line when you press the Tab key after entering a partial command. The Cisco AR 4.1 server responds based on:

- The location of the cursor including the current directory
- Any data you have entered on the command line prior to pressing the Tab key

The command completion feature emulates the behavior of Cisco IOS and Kermit. When you press the Tab key after entering part of a command, the CAR server provides any identifiable object and property names. For example, after you first issue **aregcmd** and log in to CAR, enter the following:

```
cd <Tab>
```

```
Administrators/ Radius/
```

Pressing the Tab key consecutively displays possible context-sensitive choices.

In the following example, after changing directory to **/Radius/services/local-file** an administrator wants to see the possible types of authentication services that can set.

```
cd /Radius/services/local-file
```

```
//localhost/Radius/Services/local-file ]
Name = local-file
Description =
Type = file
IncomingScript~ =
OutgoingScript~ =
OutagePolicy~ = RejectAll
OutageScript~ =
FilenamePrefix = accounting
MaxFileSize = "10 Megabytes"
MaxFileAge = "1 Day"
RolloverSchedule =
```

```
set type <Tab>
```

```
eap-leap      file          local          radius-session
eap-md5       group         odbc           rex
eap-sim       ldap          radius         tacacs-udp
```

Values can also be tab-completed. For example, if you decide to set the local-file service's type to file, you can do the following:

```
set type f<Tab>
```

and the command line completes to:

```
set type file
```

# Service Grouping Feature

The Service Grouping feature enables you to specify multiple services (called *subservices*) to be used with authentication, authorization, or accounting requests. The general purpose is to enable multiple Remote Servers to process requests.

Perhaps the most common use of this feature will be to send accounting requests to multiple Remote Servers thus creating multiple accounting logs. Another common use might be to authenticate from more than one Remote Server where, perhaps the first attempt is rejected, other Remote Servers can be attempted and an Access-Accept obtained.

Clearly, in the accounting request example, each request must be successfully processed by each subservice in order for the originator of the accounting request to receive a response. This is known as a *logical AND* of each of the subservice results. In the authenticate example, the first subservice which responds with an accept is returned to the client or if all subservices respond with *reject*, then a reject is returned to the client. This is known as a *logical OR* of each of the subservice results.

A Service is specified as a Group Service by setting its type to *group*, specifying the ResultRule (AND or OR) and specifying one or more subservices in the GroupServices subdirectory. The subservices are called in numbered order and as such are in an indexed list similar to Remote Server specification in a radius Service. Incoming and outgoing scripts for the Group Service can be optionally specified.

A subservice is any configured non-Group Service. When a Group Service is used, each subservice is called in exactly the same manner as when used alone (such as if specified as the DefaultAuthenticationService). Incoming and Outgoing scripts are executed if configured and Outage Policies are honored.

## Configuration Example - AccountingGroupService

The following example shows how to configure an accounting Group Service to deliver accounting requests to multiple Remote Servers.

**Step 1** The first task is to set up the subservices which are to be part of the AccountingGroupService. Since subservices are merely configured Services which have been included in a service group, you need only define two new Services.

For this example, we will define two new radius Services called *OurAccountingService* and *TheirAccountingService*. A provider might want to maintain duplicate accounting logs in parallel with their bulk customer's accounting logs.

**Step 2** Change directory to */radius/services*. At the command line, enter the following:

```
cd /radius/services
```

```
[ //localhost/Radius/Services ]
Entries 1 to 2 from 2 total entries
Current filter: <all>
local-file/
local-users/
```

**Step 3** At the command line, enter the following:

```
add OurAccountingService
add TheirAccountingService
```

The configuration of these Services is very similar to stand-alone Radius accounting service. Step-by-step configuration instructions are not provided, but the complete configuration is shown below:

```
[ //localhost/Radius/Services/OurAccountingService ]
Name = OurAccountingService
Description =
Type = radius
IncomingScript = OurAccountingInScript
OutgoingScript = OurAccountingOutScript
OutagePolicy = RejectAll
OutageScript =
MultipleServersPolicy = Failover
RemoteServers/
  1. OurPrimaryServer
  2. OurSecondaryServer

[ //localhost/Radius/Services/TheirAccountingService ]
Name = TheirAccountingService
Description =
Type = radius
IncomingScript = TheirAccountingInScript
OutgoingScript = TheirAccountingOutScript
OutagePolicy = RejectAll
OutageScript =
MultipleServersPolicy = Failover
RemoteServers/
  1. TheirPrimaryServer
  2. TheirSecondaryServer
```

The next step is to create the new **AccountingGroupService**. The purpose of this Service is to process Accounting requests through both OurAccountingService and TheirAccountingService.

**Step 4** At the command line, enter the following:

```
add AccountingGroupService
```

```
Added AccountingGroupService
```

```
cd AccountingGroupService
```

```
[ //localhost/Radius/Services/AccountingGroupService ]
Name = AccountingGroupService
Description =
Type =
IncomingScript =
OutgoingScript =
```

```
set type group
```

```
Set Type group
```

**Step 5** Set the ResultRule to **AND** to ensure that both services process the accounting request successfully.

```
set ResultRule AND
```

```
Set ResultRule AND
```

```
Is
```

```
[ //localhost/Radius/Services/AccountingGroupService ]
  Name = AccountingGroupService
  Description =
  Type = group
  IncomingScript =
  OutgoingScript =
  ResultRule = AND
  GroupServices/
```

**set IncomingScript AcctGroupSvcInScript**

**set OutgoingScript AcctGroupSvcOutScript**

Now we must add the Services we created OurAccountingService and TheirAccountingService as subservices of the Group Service.

**Step 6** At the command line, enter the following:

**cd GroupServices**

```
[ //localhost/Radius/Services/AccountingGroupService/GroupServices ]
```

**set 1 OurAccountingService**

```
Set 1 OurAccountingService
```

**Set 2 TheirAccountingService**

```
Set 2 TheirAccountingService
```

**ls**

```
[ //localhost/Radius/Services/AccountingGroupService ]
  Name = AccountingGroupService
  Description =
  Type = group
  IncomingScript = AcctGroupSvcInScript
  OutgoingScript = AcctGroupSvcOutScript
  ResultRule = AND
  GroupServices/
    1. OurAccountingService
    2. TheirAccountingService
```

---

This completes the setup of the AccountingGroupService. To use this Service simply set it as the DefaultAccountingService and/or configure a policy/rule set which will select this Service. Essentially, this can be used in the same manner as any other stand-alone service.

## Summary of Events

The following describes the flow of what happens when a client sends an accounting request which is processed by the AccountingGroupService:

1. ActGroupSvcInScript is executed.

2. OurAccountingService is called.
3. OurAccountingService's Incoming Script, OurAccountingInScript is called.
4. The request is sent to the Remote Server OurPrimaryServer and/or OurSecondaryServer, if necessary.
5. If a response is not received, because we used the **AND** ResultRule, the request failed and no response is sent to the client and the request is dropped. If a response is received, then the process continues.
6. OurAccountingService's Outgoing Script, OurAccountingOutScript is called.
7. TheirAccountingService is called.
8. TheirAccountingService's Incoming Script, TheirAccountingInScript is called.
9. The request is sent to the Remote Server TheirPrimaryServer and/or TheirSecondaryServer, if necessary.
10. If a response is not received, because we used the **AND** ResultRule, the request failed and no response is sent to the client and the request is dropped. If a response is received, then the process continues.
11. TheirAccountingService's Outgoing Script, TheirAccountingOutScript is called.
12. AcctGroupSvcOutScript is executed.
13. Standard processing continues.

## Configuration Example 2 - AuthenticationGroupService

In this example, we will configure a Group Service for the purposes of providing alternate Remote Servers for a single authentication. Simply put, if Service A rejects the request, try Service B.

**Step 1** The first task is to set up the subservices which are to be part of the AuthenticationGroupService. Since subservices are merely configured Services which have been included in a service group, we will simply define two new Services. For simplicity, we will define two new radius Services called AuthenticationServiceA and AuthenticationServiceB.

**Step 2** At the command line, enter the following:

```
cd /radius/services

[ //localhost/Radius/Services ]
  Entries 1 to 2 from 2 total entries
  Current filter: <all>
  local-file/
  local-users/
```

```
add AuthenticationServiceA
```

```
add AuthenticationServiceB
```

**Step 3** The configuration of these Services is very similar to stand-alone Radius authentication service. Step-by-step configuration instructions are not provided, but the complete configuration is shown below:

```
[ //localhost/Radius/Services/AuthenticationServiceA ]
  Name = AuthentictionServiceA
  Description =
  Type = radius
  IncomingScript = AuthAInScript
  OutgoingScript = AuthAOutScript
  OutagePolicy = RejectAll
  OutageScript = AuthAOutageScript
  MultipleServersPolicy = Failover
  RemoteServers/
    1. PrimaryServerA
    2. SecondaryServerA

[ //localhost/Radius/Services/AuthenticationServiceB ]
  Name = AuthentictionServiceB
  Description =
  Type = radius
  IncomingScript = AuthBInScript
  OutgoingScript = AuthBOutScript
  OutagePolicy = RejectAll
  OutageScript = AuthBOutageScript
  MultipleServersPolicy = Failover
  RemoteServers/
    1. PrimaryServerB
    2. SecondaryServerB
```

The next step is to create the new "AuthenticationGroupService". The purpose of this Service is to process authentication requests through both AuthenticationServiceA and AuthenticationServiceB if AuthenticationServiceA rejects the request.

**Step 4** At the command line, enter the following:

#### **add AuthenticationGroupService**

```
Added AuthenticationGroupService
```

#### **cd AuthenticationGroupService**

```
[ //localhost/Radius/Services/AuthenticationGroupService ]
  Name = AuthenticationGroupService
  Description =
```

```
Type =
IncomingScript =
OutgoingScript =
```

### set type group

```
Set Type group
```

Next set the ResultRule to **OR** because we want to ensure that if the first subservice rejects the request, we then try the second subservice. If the second subservice rejects the request, then the response to the client is a reject.

**Step 5** At the command line, enter the following:

### set ResultRule OR

```
Set ResultRule OR
```

### Set IncomingScript AuthGroupSvcInScript

```
Set OutgoingScript AuthGroupSvcOutScript
```

### Set IncomingScript AuthGroupSvcInScript

```
Set OutgoingScript AuthGroupSvcOutScript
```

### ls

```
[ //localhost/Radius/Services/AuthenticationGroupService ]
Name = AuthenticationGroupService
Description =
Type = group
IncomingScript = AuthGroupSvcInScript
OutgoingScript = AuthGroupSvcOutScript
ResultRule = OR
GroupServices/
```

Now we must add the services we created "AuthenticationServiceA" and "AuthenticationServiceB" as subservices of the Group Service.

**Step 6** At the command line, enter the following:

### cd GroupServices

```
[ //localhost/Radius/Services/AuthenticationGroupService/GroupServices ]
```

**set 1 AuthenticationServiceA**

```
Set 1 AuthenticationServiceA
```

**Set 2 AuthenticationServiceB**

```
Set 2 AuthenticationServiceB
```

**Is**

```
[ //localhost/Radius/Services/AuthenticationGroupService ]
Name = AuthenticationGroupService
Description =
Type = group
IncomingScript = AuthGroupSvcInScript
OutgoingScript = AuthGroupSvcOutScript
ResultRule = OR
GroupServices/
    1. AuthenticationServiceA
    2. AuthenticationServiceB
```

---

This completes the setup of the AuthenticationGroupService. To use this Service simply set it as the DefaultAuthenticationService and/or configure a policy/rule set which will select this Service. Essentially, this can be used in the same manner as any other stand-alone Service.

## Summary of Events

The following describes the flow of what happens when a client sends an Authentication request which is processed by the AuthenticationGroupService:

1. AuthGroupSvcInScript is executed.
2. AuthenticationServiceA is called.
3. AuthenticationServiceA's Incoming Script, AuthAInScript is called.
4. If the response is a reject or the request is dropped (due to an Outage Policy):
  - a. AuthenticationServiceA's Outgoing Script, AuthAOutScript is called.
  - b. Processing continues with the next service.
5. If the response is an Accept:
  - a. AuthenticationServiceA's Outgoing Script, AuthAOutScript is called.
  - b. Skip to step 9.
6. AuthenticationServiceB is called.
7. AuthenticationServiceB's Incoming Script, AuthBInScript is called.

8. Since this is the last subservice in our Group Service:
  - a. AuthenticationServiceB's Outgoing Script, AuthBOutScript is called.
  - b. Regardless of whether the request is Accepted or Rejected, processing will continue at step 9.
9. AuthGroupSvcOutScript is executed.
10. Standard processing continues.

## SHA-1 Support for LDAP-Based Authentication

The CAR server supports secure hash algorithm (SHA-1) for LDAP-based authentication. This feature enables the CAR server to authenticate users whose passwords are stored in LDAP servers and hashed using the SHA-1 encoding scheme.

SHA-1 support actually adds functionality for the following three features to CAR:

- Authentication of PAP access requests against an LDAP user entry that uses the SHA-algorithm to the hash password attribute
- Authentication of PAP access requests against an LDAP user entry that uses the SSHA algorithm to hash the password attribute
- Configuration of the CAR server to dynamically determine how password attributes retrieved from LDAP are encrypted and process them accordingly

This enhancement is 100% backwards compatible. All previously supported values for the PasswordEncryptionStyle property are still supported and still provide the same behavior. The only noticeable change is that **dynamic** is now the default value for the PasswordEncryptionStyle property.

## Remote LDAP Server Password Encryption

Apart from the two values, none and crypt, of the PasswordEncryptionStyle property on a Remote LDAP Server, SHA-1 supports adds three additional values for the PasswordEncryptionStyle property. [Table 16-2](#) lists the valid values for this property and describes the corresponding behavior.

**Table 16-2 Remote LDAP Server Password Encryption Style Values**

PasswordEncryptionStyle	Access Registrar Behavior
none	All passwords retrieved from this LDAP server are assumed to be returned to CAR as clear text. (There is no change in this functionality.)
crypt	All passwords retrieved from this LDAP server are assumed to be returned to CAR as passwords encrypted using the UNIX <i>crypt</i> algorithm. (There is no change in this functionality.)  Passwords can be preceded by the {crypt} prefix, which is stripped before comparing passwords.

**Table 16-2 Remote LDAP Server Password Encryption Style Values (continued)**

PasswordEncryptionStyle	Access Registrar Behavior
SHA-1	All passwords retrieved from this LDAP server are assumed to be returned to CAR as a Base64-encoded version of the user's password after it has been hashes using the SHA-1 mechanism (as defined by Netscape).  Passwords can be preceded by the {sha} prefix, which is stripped before comparing passwords.
SSHA-1	All passwords retrieved from this LDAP server are assumed to be encrypted/hashed using the SSHA mechanism (as defined by Netscape). Passwords can be preceded by the {ssha} prefix, which is stripped before comparing passwords.  <b>Note</b> This is a Netscape/iPlanet-specific mechanism.
dynamic	The value instructs CAR to choose the encryption mechanism on a case-by-case basis after it determines the presence of a known prefix, which the LDAP server prepends to the value of the password attribute.  For example, if the following was returned from an LDAP server as a password attribute: {SHA}qZk+NkcGgWq6PiVxeFDCbJzQ2J0=, the password would be processed using the SHA-1 mechanism. This value will be the new default for the PasswordEncryptionStyle property.

## Dynamic Password Encryption

When using the dynamic setting for the PasswordEncryptionStyle property on a Remote LDAP Server, the CAR server looks for the prefixes listed in [Table 16-3](#) to determine if encryption or a hash algorithm should be used during password comparison.


**Note**

Password prefixes are not case sensitive.

**Table 16-3 Remote LDAP Server Password Prefix Values**

Password Prefix	Encryption/Hash Algorithm Used
none	None; when no known prefix is found, the password attribute is assumed to be in clear text.
{crypt}	UNIX crypt algorithm
{sha}	Secure Hash Algorithm, version 1 (SHA-1)
{ssha}	SSHA-1, as defined by Netscape.

The default value for the PasswordEncryptionStyle property on a Remote LDAP Server is **dynamic**.


**Note**

Using the *dynamic* setting for the PasswordEncryptionStyle property will require a bit more processing for each password comparison. When using dynamic, the CAR server must examine each password for a known prefix. This should have no visible impact on performance.

## Logs

Turn on **trace** to level 4 to indicate (via the trace log) which password comparison method is being used.

## Dynamic Attributes

CAR supports dynamic values for the configuration object properties listed below. Dynamic attributes are similar to UNIX shell variables. With dynamic attributes, the value is evaluated at run time. All of the objects that support dynamic attributes will have validation turned off in **aregcmd**.

## Object Properties with Dynamic Support

The following object properties support dynamic values:

Radius

DefaultAuthenticationService

DefaultAuthorizationService

DefaultAccountingService

DefaultSessionManager

IncomingScript

OutgoingScript



**Note**

Do not use the following environment variables:

Accounting-Service for the **/Radius/DefaultAccountingService**, Authentication-Service for the **/Radius/DefaultAuthenticationService**, or Authorization-Service for the **/Radius/DefaultAuthorizationService**

User-Profile for the **BaseProfile**, User-Group for the **Group**, User-Authorization for the **AuthorizationScript**, Session-Manager for the **DefaultSessionManager**, or Session-Service for the **DefaultSessionService**.

/Radius/Clients

client1/

IncomingScript

OutgoingScript

/Radius/Userlist/Default

user1/

Group

BaseProfile

AuthenticationScript

AuthorizationScript

/Radius/UserGroup

Group1/

```

    BaseProfile
    AuthenticationScript
    AuthorizationScript
/Radius/Vendor
  Vendor1/
    IncomingScript
    OutgoingScript
/Radius/Service
  Service1/
    IncomingScript
    OutgoingScript
    OutageScript
    OutagePolicy
/Radius/RemoteServers
  remoteserver1/
    IncomingScript
    OutgoingScript
  Remoteldapservice1/
    Searchpath
    Filter

```




---

**Note** To differentiate the properties that support dynamic attributes, we place a tilde (~) after each property, as in `IncomingScript~`. However, when the CAR administrator is required to set values for those properties, continue to use the original property name, such as `set IncomingScript ${elrealm}{Test}`. The tilde is only for visual effect, and including the tilde will generate an error (“310 command Failed.”)

---

## Dynamic Attribute Format

The format of the dynamic attribute is:

```

${eq|attribute-name}{default-name}

```

where **e** stands for environment dictionary, **q** stands for request dictionary and **p** stands for response dictionary. You can use e, q and p in any order. The attribute name is the name for the attribute from environment dictionary, request dictionary, or response dictionary.

For example,

```

/Radius
DefaultAuthenticationService = ${eq|realm}{local-users}

```

The default Authentication Service is determined at run time. CAR first checks to see if there is one value of *realm* in the environment dictionary. If there is, it becomes the value of `DefaultAuthenticationService`. If there is not, check the value of *realm* in the request dictionary. If there

is one value, it becomes the value of DefaultAuthenticationService. Otherwise, local-users is the DefaultAuthenticationService. If we don't set local-users as the default value, the DefaultAuthenticationService is *null*. The same concept applies to all other attribute properties.

The validation for the dynamic values of the object property will only validate the default value. In the above example, CAR will do validation to check whether local-users is one of services defined in the service subdirectory.

**Note**

When setting specific property values, do not use the tilde (~) in the property name. Doing so generates a *310 Command Failed* error.

## Tunneling Support Feature

Tunneling support is strictly based upon the IETF RFC: “RADIUS Attributes for Tunnel Protocol Support” (<http://www.ietf.org/rfc/rfc2868.txt>).

Table 16-4 lists the tunneling attributes supported in this CAR release.

**Table 16-4 Tunneling Attributes Supported by CAR**

Attribute Number	Attribute
64	Tunnel-Type
65	Tunnel-Medium-Type
66	Tunnel-Client-Endpoint
67	Tunnel-Server-Endpoint
69	Tunnel-Password
81	Tunnel-Private-Group-ID
82	Tunnel-Assignment-ID
83	Tunnel-Preference
90	Tunnel-Client-Auth-ID
91	Tunnel-Server-Auth-ID

The tunneling attribute has the following format:

(1 byte)	(1 byte)	(1 byte)	(variable number of bytes)
Type	Length	Tag	Value

## Configuration

1. Configure the tag attributes as untagged attributes under the **/Radius/Advanced/Attribute Dictionary** directory (for example, **Tunnel-Type**).
2. Attach the “\_tag” tag to these attributes when configuring the attributes under all of the other directories as tagged attributes (for example, **Tunnel-Type\_tag10** under the **/Radius/Profiles/test** directory). Without the tag number, the default value is (**\_tag = \_tag0**).

## Example

```
/Radius/Advanced/Attribute Dictionary
  /Tunnel-Client-ID
    Name = Tunnel-Client-Endpoint
    Description =
    Attribute = 66
    Type = STRING
    Min = 0
    Max = 253

/Radius/Profiles/test
  Name = test
  Description =
  /Attributes
    Tunnel-Client-Endpoint_tag3 = "129.56.123.1"
```

## Notes

1. “\_tag” is reserved for the tunneling attributes. No other attributes should include this suffix.
2. The tag number value can range from 0 through 31.

## Validation

The CAR server checks whether the tag attributes are defined under the **/Radius/Advanced/Attribute Dictionary** directory. The server also checks whether the tag number falls within the range (0-31).

## xDSL VPI/VCI Support for Cisco 6400

To provide this support, a distinction must be made between device authentication packets and regular user authentication packets.

## Using User-Name/User-Password for Each Cisco 6400 Device

This approach assumes that for every 6400 NAS, a device-name/device-password is created for each. Following are the required changes:

For each NAS in CAR:

```
Name = test6400-1
Description =
IPAddress = 209.165.200.224
SharedSecret = secret
Type = NAS
Vendor =
IncomingScript =
OutgoingScript =
Device-Name = theDevice
Device-Password = thePassword
```

When the 6400 sends out the device authentication packet, it might have different **User-Name/User-Password** attributes for each 6400 NAS. When CAR receives the packet, it tries to obtain the **Device-Name/Device-Password** attributes from the NAS entry in the CAR configuration database. When the **User-Name/User-Password** in the packet match the configured **Device-Name/Device-Password** attribute values, CAR assumes that it must get the device. The next step is to replace the **User-Name** attribute with the concatenated `<module>/<slot>/<port>` string. From this point, the packet is treated as a regular packet.

**Note**

A user record with the name of the concatenated string must be created.

## Format of the New User-Name Attribute

After the device is identified, the **User-Name** attribute is replaced with the new value. This new value is the concatenation of 6400 `<module>/<slot>/<port>` information from the **NAS-Port** attribute and the packet is treated as a regular user authentication from this point on.

**Note**

This format only supports NAS Port Format D. See Cisco IOS documentation for more information about NAS port formats.

The format of the new **User-Name** attribute is the **printf** of “%s-%d-%d-%d-%d-%d” for the following values:

**NAS-IP**—in dot format of the **NAS-IP-Address** attribute. For example, 10.10.10.10.

**slot**—apply mask 0xF0000000 on **NAS-Port** attribute and shift right 28 bits. For example, **NAS-Port** is 0x10000000, the slot value is 1.

**module**—apply mask 0x08000000 on **NAS-Port** attribute and shift right 27 bits. For example, **NAS-Port** is 0x08000000, the module value is 1.

**port**—apply mask 0x07000000 on **NAS-Port** attribute and shift right 24 bits. For example, **NAS-Port** is 0x06000000, the port value is 6.

**VPI**—apply mask 0x00FF0000 on **NAS-Port** attribute and shift right 16 bits. For example, **NAS-Port** is 0x00110000, the VPI value is 3.

**VCI**—apply mask 0x0000FFFF on **NAS-Port** attribute. For example, **NAS-Port** is 0x00001001, the VCI value is 9.

## Apply Profile in Cisco Access Registrar Database to Directory Users

You can define the **User-Profile** and **User-Group** environment variables in the directory mapping and CAR will apply the profiles defined in the CAR database to each directory user having any of these two variables set.

### User-Profile

This attribute is of type string with the format:

<Value1>::*<Value2>* ...

The **User-Profile** attribute is intended to hold a list of profile names. <Value1> and <Value2> represent the names of the profiles. They are separated by the “::” character, therefore, the “::” can not be part of the profile name. The order of values in the string has significance, as the profiles are evaluated from left to right. In this example, profile <Value2> is applied after profile <Value1>.

Assume the user record has a field called `UserProfile` that holds the name of the profile that applies to this user. This field is mapped to the environment attribute **User-Profile**. Following is how the mapping is done with **aregcmd**:

```
QuickExample/
  Name = QuickExample
  Description =
  Protocol = ldap
  IPAddress = 209.165.200.224
  Port = 389
  ReactivateTimerInterval = 300000
  Timeout = 15
  HostName = QuickExample.company.com
  BindName =
  BindPassword =
  UseSSL = FALSE
  SearchPath = "o=Ace Industry, c=US"
  Filter = (uid=%s)
  UserPasswordAttribute = password
  LimitOutstandingRequests = FALSE
  MaxOutstandingRequests = 0
  MaxReferrals = 0
  ReferralAttribute =
  ReferralFilter =
  PasswordEncryptionStyle = None
  LDAPToEnvironmentMappings/
    UserProfile = User-Profile
  LDAPToRadiusMappings/
```

After CAR authenticates the user, it checks whether **User-Profile** exists in the environment dictionary. If it finds **User-Profile**, for each value in **User-Profile**, CAR looks up the profile object defined in the configuration database and adds all of the attributes in the profile object to the response dictionary. If any attribute is included in more than one profile, the newly applied profile overrides the attribute in the previous profile.

## User-Group

You can use the **User-Group** environment variable to apply the user profile as well. In CAR, a user can belong to a user group, and that user group can have a pointer to a user profile. When CAR finds that a packet has **User-Group** set, it obtains the value of the **User-Profile** within the user group, and if the **User-Profile** exists, it applies the attributes defined in the user profile to that user.

Note that in CAR, every user can also directly have a pointer to a user profile. CAR applies profiles in the following order:

1. If the user profile defined in the user group exists, apply it.
2. If the user profile defined in the user record exists, apply it.

The profile in **User-Group** is more generic than in **User-Profile**. Therefore, CAR applies the profile from generic to more specific.

## Example User-Profile and User-Group Attributes in Directory User Record

You can use an existing user attribute in the user record to store profile info. When this is a new attribute, we suggest you create a new auxiliary class **AR\_UserRecord** for whichever user class is used.

**AR\_User\_Profile** and **AR\_User\_Group** are two optional members in this class. They are of type string. The mapping is as follows:

```
LDAPToEnvironmentMappings/
  AR_User_Profile = User-Profile
  AR_User_Group = User-Group
```

## Directory Multi-Value Attributes Support

If any attributes mapped from the LDAP directory to the CAR response dictionary are multivalued, the attributes are mapped to multiple RADIUS attributes in the packet.

## MultiLink-PPP (ML-PPP)

CAR supports MultiLink-PPP (ML-PPP). ML-PPP is an IETF standard, specified by RFC 1717. It describes a Layer 2 software implementation that opens multiple, simultaneous channels between systems, providing additional bandwidth-on-demand, for additional cost. The ML-PPP standard describes how to split, recombine, and sequence datagrams across multiple B channels to create a single logical connection. The multiple channels are the ports being used by the Network Access Server (NAS).

During the AA process, CAR authenticates the user connection for each of its channels, even though they belong to the same logical connection. The Authentication process treats the multilink connection as if it is multiple, single link connections. For each connection, CAR creates a session dedicated for management purposes. The session stays active until you logout, which subsequently frees up all of the ports in the NAS assigned to each individual session, or until the traffic is lower than a certain threshold so that the secondary B channels are destroyed thereafter. CAR has the responsibility of maintaining the active session list and discards any session that is no longer valid in the system, by using the accounting stop packet issued from NAS. The multiple sessions that were established for a single logical connection must be destroyed upon the user logging out.

In addition, the accounting information that was gathered for the sessions must be aggregated for the corresponding logical connection by the accounting software. CAR is only responsible for logging the accounting start and accounting stop times for each session. As those sessions belong to the same bundle, IETF provides two standard RADIUS attributes to identify the related multilink sessions. The attributes are **Acct-Multi-Session-Id** (attribute **50**) and **Acct-Link-Count** (attribute **51**), where **Acct-Multi-Session-Id** is a unique Accounting identifier used to link multiple related sessions in a log file, and **Acct-Link-Count** provides the number of links known to have existed in a given multilink session at the time the Accounting record was generated. The Accounting software is responsible for calculating the amount of the secondary B channel's connection time.

The secondary B channel can go up and down frequently, based upon traffic. The Ascend NAS supports the **Target-Util** attribute, which sets up the threshold for the secondary channel. When the traffic is above that threshold the secondary channel is up, and when the traffic is below that threshold, the secondary B channel is brought down by issuing an Accounting stop packet to CAR. On the other hand, if you bring down the primary channel (that is, log out), the secondary B channel is also destroyed by issuing another Accounting stop packet to CAR.

[Table 16-5](#) lists ML-PPP related attributes.

**Table 16-5 ML-PPP Attributes**

Number	Attribute	Cisco NAS (IOS 11.3 Release)	Ascend NAS
44	Acct-Session-Id	Supported	Supported
50	Acct-Multi-Session-Id	Supported	Supported
51	Acct-Link-Count	Supported	Supported
62	Port-Limit	Supported	Supported
234	Target-Util	Not Supported	Supported
235	Maximum-Channels	Supported	Supported

Following are sample configurations for ML-PPP:

```

/Radius
  /Profile
    /Default-ISDN-Users
      Name = Default-ISDN-Users
      Description =
      Attributes/
        Port-Limit = 2
        Target-Util = 70
        Session-Timeout = 70

/Radius
  /UserGroups
    /ISDN-Users
      Name = ISDN-Users
      Description = " Users who always use ISDN"
      BaseProfile = Default-ISDN-Users
      Authentication-Script =
      Authorization-Script =

```

The **Port-Limit** attribute controls the number of concurrent sessions a user can have. The **Target-Util** attribute controls the threshold level at which the second B channel should be brought up.

## Dynamic Updates Feature

The Dynamic Updates feature enables changes to server configurations made using **aregcmd** to take effect in the CAR server after issuing the **save** command, eliminating the need for a server **reload** after making changes.

**Table 16-6** lists the Radius object and its child objects. For each object listed, the **Add** and **Modify or Delete** columns indicate whether a dynamic update occurs after adding, modifying, or deleting an object or attribute. Entries in the **Add** and **Modify or Delete** columns also apply to child objects and child attributes of the objects listed, unless the child object is explicitly listed below the object, such as **/Radius/Advanced/Ports** or **/Radius/Advanced/Interfaces**.

**Table 16-6 Dynamic Updates Effect on Radius Server Objects**

Object	Add	Modify or Delete
Radius	Yes	Yes
UserLists	Yes	Yes

**Table 16-6** Dynamic Updates Effect on Radius Server Objects (continued)

Object	Add	Modify or Delete
UserGroups	Yes	Yes
Policies	Yes	Yes
Clients	Yes	Yes
Vendors	Yes	Yes
Scripts	Yes	Yes
Services	Yes	Yes
SessionManagers	Yes	No
ResourceManagers	Yes	No
Profiles	Yes	Yes
Rules	Yes	Yes
Translations	Yes	Yes
TranslationGroups	Yes	Yes
RemoteServers	Yes	No
Replication	No	No
Advanced	Yes	Yes
SNMP	No	No
Ports	No	No
Interfaces	No	No

The Dynamic Updates feature is subject to the following limitations:

- Changes to the Ports or Interfaces objects are not dynamically updated. An **aregcmd reload** command must be issued for these changes to be propagated to the CAR server.
- Changes (modifications and deletions) to existing Session Manager and Resource Manager objects are not dynamically updated. An **aregcmd reload** command must be issued for these changes to be propagated to the CAR server. However, additions of new Session Manager and Resource Manager objects are dynamically updated. Active sessions and allocated resources are preserved in this case.
- Changes to the CAR configuration might not be immediately propagated to the server. Dynamic updates are only carried out in a *safe* environment (that is, when packets are not being processed and when packet processing can be delayed until the changes can be made on the server safely). Dynamic updates will yield to packet processing when appropriate, thus not significantly impacting server performance.
- Changes to SNMP require the CAR server to be restarted (**/etc/init.d/arservagt restart**)

# NAS Monitor

The ability to monitor when a NAS is *down* (really only unreachable from CAR) is provided by **nasmonitor**. This program will repeatedly query a TCP port at the specified IP address until the device (NAS) is reachable. If the NAS is not reachable after a period of time, a warning E-mail is sent; if the NAS is still not reachable after another period of time, a message is sent to CAR to release all sessions associated with that NAS. The port to query, the query frequency, the first time interval, the back-off time interval, and the E-mail address to send to are all configurable (with defaults); the only required parameter is the NAS IP address. This program will work for any device that has a TCP port open; it can either be run by hand, when desired, or put in a **cron** job. See **nasmonitor -h** for details.

**Note**

You must have **telsh** installed in **/usr/local/bin** to use **nasmonitor**. **telsh** is part of the standard Tcl installation that can be downloaded from <http://www.scriptics.com>.

## Automatic Information Collection (arbug)

You can use the script **arbug** to collect information about your CAR server. The results are collected into a tarball that can be E-mailed or **ftped** to Cisco as requested.

**arbug** collects all the relevant information needed to report a problem to CAR support. The goal of the **arbug** script is to make sure all the necessary information is collected.

**Note**

The **arbug** script neither updates nor replaces any system or CAR-related configuration.

## Running arbug

To run the **arbug** script, change directory to **/cisco-ar/bin** and enter the following:

```
./arbug
```

The following is a typical sequence.

```
Looking around...
Cluster:
User: admin
Password:
The report /tmp/arbug.10085/arbug.tar is ready to send; you
may want to compress it first using gzip or compress.
hostname user_name bin>
```

## Files Generated

The **arbug** script generates five files that are compressed into a tarball. [Table 16-7](#) provides a summary of the information found in each of the files.

Table 16-7 Files Generated by arbug

File	Description
<b>car.debug.tar.*</b>	Machine-specific information including OS type, RAM details, disk space information, swap space information, patch information and open file details.
<b>car.config.tar.*</b>	CAR server configuration, server statistics, database dump by taking the administrator username and password as the input.
<b>car.confini.tar.*</b>	Information about ODBC .ini files and SNMP configuration
<b>car.core.tar.*</b>	Core files if any are present
<b>car.logcscr.tar.*</b>	Information from scripts directory, certificate directory, license directory

## Simultaneous Terminals for Remote Demonstration

Multiple people can view and interact in a single demonstration by using the *share-access* program, a standard GNU release with a special configuration for use with CAR. To run **screen**, a technical support specialist (CSE or DE) will **telnet** to your server and log in as *cisco*. While you run **/opt/CSCOar/bin/share-access** (assuming **/opt/CSCOar** is the CAR path) as *root*, the CSE or DE runs **/opt/CSCOar/bin/share-access -r root**. Now both people (or more) can see what the other types, as well as the results of the commands entered. The special CAR configuration only allows *root* and *cisco* to run **screen**. To end a *share-access* session, type Control-D.

## Support for RADIUS Check Item Attributes

CAR supports RADIUS check item attributes configuration at the user and group levels. You can configure the CAR server to check for attributes that must be present or attributes that must not be present in the Access-Request packet for successful authentication.

When using check item attributes, the CAR server will reject Access-Requests if:

- Any of the configured check item attributes are not present in the Access-Request packet
- Any of the Access-Request packet's check item attribute values do not match with those configured check item attribute values

For remote servers using either LDAP or ODBC, CAR allows for mapping of certain LDAP or ODBC fields to check item attributes. The mapped attributes can be used as check item attributes while processing the Access-Request packets.

When you configure check item attributes at both the user and group levels, the Cisco AR 4.1 server first checks the attributes of the user level before those of the group level. The Cisco AR 4.1 server must first authenticate the user's password in the Access-Request before validating the check item attributes.

The Cisco AR 4.1 server logs details about any rejected Access-Requests as a result of check items processing.

## Configuring Check Items

You use **aregcmd** to configure check item attributes.

## Configuring User Check Items

The following example shows how to configure UserList check item attributes.

- 
- Step 1** Log in to the Cisco AR 4.1 server, and use **aregcmd** to navigate to **//localhost/Radius/UserLists/default/bob**.

```
[ //localhost/Radius/UserLists/Default/bob ]
  Name = bob
  Description =
  Password = <encrypted>
  AllowNullPassword = FALSE
  Enabled = TRUE
  Group~ = PPP-users
  BaseProfile~ =
  AuthenticationScript~ =
  AuthorizationScript~ =
  UserDefined1 =
  Attributes/
  CheckItems/
```

- Step 2** Change directory to CheckItems.

**cd CheckItems**

```
[ //localhost/Radius/UserLists/Default/bob/CheckItems ]
```

- Step 3** Use set to add any attributes to be used as check items.

**set calling-Station-Id 4085551212**

**save**

---

## Configuring Usergroup Check Items

The following example shows how to configure UserGroups check item attributes.

- 
- Step 1** Log in to the Cisco AR 4.1 server, and use **aregcmd** to navigate to **//localhost/Radius/UserGroups/Default**.

**cd /Radius/UserGroups/Default**

```
[ //localhost/Radius/UserGroups/Default ]
  Name = Default
  Description = "Users who sometimes connect using PPP and sometimes connect "
  BaseProfile~ =
  AuthenticationScript~ =
  AuthorizationScript~ = AuthorizeService
  Attributes/
  CheckItems/
```

- Step 2** Change directory to CheckItems.

**cd CheckItems**

```
[ //localhost/Radius/UserGroups/Default/CheckItems ]
```

**Step 3** Use set to add any attributes to be used as check items.

```
set NAS-IP-Address 10.10.10.10
```

```
save
```

---

## User-Specific Attributes

The CAR server supports user-specific attributes which enables the CAR server to return attributes on a per-user or per-group basis without having to use profiles.

The CAR server includes a property called HiddenAttributes to the User and UserGroup object. The HiddenAttributes property contains a concatenation of all user-level reply attributes. The HiddenAttributes property is not displayed, nor can the value be set or unset using the command-line interface.

The order of application of attributes is as follows:

1. UserGroup Base Profile
2. UserGroup Attributes
3. User Base Profile
4. User Attributes

The value of the HiddenAttributes property is used dynamically to construct and populate a virtual *attributes* directory in the User object. All values from the Attributes directory will go into the HiddenAttributes property. This occurs transparently when the administrator issues a save command.

## Packet of Disconnect

CAR supports the Packet of Disconnect (POD) feature that enables the CAR server to send disconnect requests (PODs) to a NAS so that all the session information and the resources associated with the user sessions can be released. CAR can also determine when to trigger and send the POD.

For example, when a PDSN handoff occurs during a mobile session, the new PDSN sends out a new access-request packet to CAR for the same user. CAR should detect this handoff by the change in NAS-Identifier in the new request and trigger sending a POD to the old PDSN if it supports POD. CAR also provides an option for administrator to initiate sending POD requests through the command-line interface (CLI) for any user session. CAR forwards POD requests from external servers to the destination NAS.

## Configuring Packet of Disconnect

This section describes how to configure the POD feature.

**Note**

Some of the properties used to configure POD in earlier releases of CAR have been renamed in Cisco AR 4.1.

---

## Configuring the Client Object

You should enable POD for each client object that might want to send disconnect requests to those clients. You enable POD in a client object using the `EnableDynamicAuthorization` property. This property is set to `FALSE` by default when you create a client object. The following example shows the default configuration for a new client object, `NAS1`.

```
[ //localhost/Radius/Clients/NAS1 ]
  Name = nas1
  Description =
  IPAddress =
  SharedSecret =
  Type = NAS
  Vendor =
  IncomingScript~ =
  OutgoingScript~ =
  EnableDynamicAuthorization = FALSE
```

If the CAR server might send a POD to this client, set the `EnableDynamicAuthorization` property to `TRUE`. When you set this property to `TRUE`, the CAR server creates a `DynamicAuthorizationServer` subdirectory under the client object. The following example shows a newly created `DynamicAuthorizationServer` subdirectory:

```
[ //localhost/Radius/Clients/NAS1/DyanamicAuthorizationServer ]
  Port = 3799
  DynamicAuthSharedSecret =
  InitialTimeout = 5000
  MaxTries = 3
  PODAttributeGroup =
  COAAttributeGroup =
```

The default port is 3799. You can change the port, if desired.

The property `DynamicAuthSharedSecret` is initially set to the same as value as the client's `SharedSecret` property when you set `EnableDynamicAuthorization` to `TRUE`. You can chose to configure a different secret for POD in this subdirectory.

The `InitialTimeout` property represents the number of milliseconds used as a timeout for the first attempt to send a POD packet to a remote server. For each successive retry on the same packet, the previous timeout value used is doubled. You must specify a number greater than zero, and the default value is 5000 (or 5 seconds).

The `MaxTries` property represents the number of times to send a proxy request to a remote server before deciding the server is off-line. You must specify a number greater than zero, and the default is 3.

The `PODAttributeGroup` property points to a group of attributes to be included in a disconnect-request packet sent to this client.

You can create and configure the `PODAttributeGroup` in the `/Radius/Advanced/AttributeGroups/` directory. The default group contains commonly used POD attributes `NAS-Port` and `Acct-Session-Id`.

The `COAAttributeGroup` property is used with the Change of Authorization (CoA) feature, also known as hot-lining.

## Configuring a Resource Manager for POD

CAR provides a resource manager type called *session-cache*. When you set a resource manager to *session-cache*, the resource manager's configuration contains a subdirectory called *AttributesToBeCached*. The following is an example Resource Manager set to type *session-cache*:

```
[ //localhost/Radius/ResourceManagers/PODresourceMgr ]
```

```
Name = PODresourceMgr
Description =
Type = session-cache
OverwriteAttributes = FALSE
AttributesToBeCached/
QueryMappings/
```

The attributes you configure under the **AttributesToBeCached** directory are cached in the session record during session management. The cached attributes are then sent in the disconnect-request for this session.

The `OverwriteAttributes` property indicates whether to overwrite the existing attributes if there are any in the session record. Since this resource manager can be invoked during Access-Request as well as Accounting-Start processing, the `OverwriteAttributes` can be used to control if the attributes cached during Access-Request processing can be overwritten with the attributes available during Accounting-Start processing.

The following is an example of a typical session-cache resource manager:

```
[ //localhost/Radius/ResourceManagers/RM-New ]
Name = RM-New
Description =
Type = session-cache
OverwriteAttributes = TRUE
AttributesToBeCached/
  1. Framed-IP-Address
  2. CDMA-Correlation-ID
QueryMappings/
```

The attributes used in the example can be added as an indexed list using **add** or **set** commands (in any order).

## Proxying POD Requests from External Servers

CAR can also proxy the disconnect requests received from external servers. To make CAR listen for external POD requests, the `ListenForDynamicAuthorizationRequests` property under **/Radius/Advanced** should be set to TRUE. The default value for this is FALSE. The default POD listening port is 3799. However this can be changed by configuring a new port of type *pod* under **/Radius/Advanced/Ports** and setting the new port number accordingly.

For security reasons, the source of a POD request should be configured as a remote server in CAR and the remote server should be configured to accept PODs. Set the property `AcceptDynamicAuthorizationRequests` to TRUE to do this. The default for this is FALSE. POD requests from unauthorized sources are silently discarded.

## CLI Options for POD

CAR has options for the **query-sessions** and **release-sessions** CLI commands that enable querying or releasing sessions based on the session's age. Another option enables querying or releasing sessions based on any valid RADIUS attribute available in the user's session record.

### query-sessions

The syntax for using **query-sessions** *with-Age* option is the following:

**query-sessions <path> with-Age <value>**

Where <path> is the path to the server, session-manager or resource manager and <value> is the minimum age of the session specified in minutes or hours with options M, Minutes, H or Hours. This command returns all sessions that are older than the given age value.

The syntax for using **query-sessions** *with-Attribute* option is the following:

**query-sessions <path> with-Attribute <name> <value>**

Where <name> is the RADIUS attribute name and <value> is the value of the attribute to be matched. This command returns the sessions where a session record contains and matches the attribute value specified in <value> field.

## release-sessions

The syntax for using **release-sessions** *with-Age* option is:

**release-sessions <path> with-Age <value>**

Where, <path> is the path to the server, session-manager or resource manager and <value> is the minimum age of the session specified in minutes or hours with options M for Minutes, H for Hours. This command returns all sessions that are older than the given age value.

The syntax for using **release-sessions** *with-Attribute* option is:

**release-sessions <path> with-Attribute <name> <value>**

Where, <name> is the RADIUS attribute name and <value> is the value of the attribute to be matched. This command returns the sessions where a session record contains and matches the attribute value specified in <value> field.

A new option is also available for **release-sessions** command to enable an administrator to trigger sending a POD for a user after the session is released.

**release-sessions <path> with-<type> <value> [send-pod]**

Where, <path> is the path to the server, Session Manager, or Resource Manager and <type> is one of the following: NAS, User, IP-Address ID, or Age. The **release-sessions** command with an optional [send-pod] at the end results in CAR sending a POD request. The PoD requests are directed to port number configured in /radius/clients/<client name>/DynamicAuthorizationServer/port. By default it is set to 3799. To configure udp xxx, set the port value as:

**/radius/clients/<client name>/DynamicAuthorizationServer/port = xxx**

# Configuring Change of Authorization Requests

CAR supports Change of Authorization (CoA) requests as defined in Internet RFC 3576 that provides a way to change authorization status of users already logged on to the network. The CoA feature, also known as hot-lining, provides a wireless operator the ability to efficiently address issues with users that might otherwise be unauthorized to access packet data services. When a problem occurs that causes a user to be unauthorized to use the packet data service, a wireless operator can use the CoA feature to resolve the problem and return the user's packet data services.

When a user is hot-lined, their packet data service is redirected to a hot-line application that notifies the user of issues that might be blocking their access to normal packet data services. Hot-lining provides users with a way to address the issues blocking their access, such as billing issues, a prepaid account that has been depleted, or an expired credit card.

The CoA feature provides an option to the wireless operator administrator to send CoA packets to the client device when a user needs to be hot-lined. When to send a CoA request to a user depends on the wireless operator's site-specific policies.

## Configuring the Client Object

You should enable CoA for each client object that might want to send CoA requests to those clients. You enable CoA in a client object using the `EnableDynamicAuthorization` property. This property is set to `FALSE` by default when you create a client object. The following example shows the default configuration for a new client object, `NAS1`.

```
[ //localhost/Radius/Clients/NAS1 ]
  Name = nas1
  Description =
  IPAddress =
  SharedSecret =
  Type = NAS
  Vendor =
  IncomingScript~ =
  OutgoingScript~ =
  EnableDynamicAuthorization = FALSE
```

If the CAR server might send a CoA request to this client, set the `EnableDynamicAuthorization` property to `TRUE`. When you set this property to `TRUE`, the CAR server creates a `DynamicAuthorizationServer` subdirectory under the client object. The following example shows a newly created `DynamicAuthorizationServer` subdirectory:

```
[ //localhost/Radius/Clients/NAS1/COA ]
  Port = 3799
  DynamicAuthSharedSecret =
  InitialTimeout = 5000
  MaxTries = 3
  PODataAttributeGroup =
  COAAttributeGroup =
```

The default port is 3799. You can change the port, if desired.

The property `DynamicAuthSharedSecret` is initially set to the same as value as the client's `SharedSecret` property when you set `EnableDynamicAuthorization` to `TRUE`. You can chose to configure a different secret for CoA in this subdirectory.

The `InitialTimeout` property represents the number of milliseconds used as a timeout for the first attempt to send a CoA packet to a remote server. For each successive retry on the same packet, the previous timeout value used is doubled. You must specify a number greater than zero, and the default value is 5000 (or 5 seconds).

The `MaxTries` property represents the number of times to send a proxy request to a remote server before deciding the server is off-line. You must specify a number greater than zero, and the default is 3.

The `COAAttributeGroup` property points to a group of attributes to be included in a CoA request packet sent to this client.

You can create and configure the COAAttributeGroup in the **/Radius/Advanced/AttributeGroups/** directory. The default group is not set to any value by default. When an attribute group is configured, the CAR server includes the attributes in this group in a CoA request. The values for these attributes are fetched from the user's session record.

The CoA attribute group configuration can be used with a session-cache Resource Manager. For example, any new attributes that are to be sent in a CoA request can be configured for caching by the session-cache Resource Manager so they will be available in the session record when it is to be sent in the CoA request.

The CoA request might also contain AV pairs from the optional profile name in the **query-session** CLI command used to send the CoA request. In a 3GPP2 scenario, a profile containing the Filter-Id attribute set to a value "Hot-Line Active" can be included when a user is to be hot-lined. This can be used as a hot-line profile possibly containing other attributes as desired by the wireless operator. Another profile might be defined containing the Filter-Id attribute with the value "Hot-Line Normal." This profile can be used with the **query-session** CLI command to bring the user back to normal.

The CoA request packet sent by the CAR server conforms to internet RFC 3756. In response to a CoA request initiated by the CAR server, the client should respond with a COA-ACK if it is able to hot-line the user based on credentials available in the CoA request. If the client is unable to hot-line the user for any reason, the client can include an error-cause attribute with the appropriate reason in a COA-NAK packet.

The CAR server logs all CoA responses. If the CAR server does not receive a response to a CoA request within the timeout period, it will retransmit for the configured number of retries, then logs an error if no response is received.

The CAR server forwards proxied CoA requests sent by external servers to the destination NAS. The CoA requests are proxied based on the NAS-IP-Address in the incoming request. The proxied CoA requests from external servers are forwarded to the destination NAS only if the source IP address is configured to accept dynamic authorization requests. The responses received from the NAS (either COA-ACK or COA-NAK) are forwarded back to the source where the CAR server received the original proxy request.

## Dynamic DNS

CAR supports the Dynamic DNS protocol providing the ability to update DNS servers. The dynamic DNS updates contain the hostname/IP Address mapping for sessions managed by CAR.

You enable dynamic DNS updates by creating and configuring new Resource Managers and new Remote Servers, both of type *dynamic-dns*. The dynamic-dns Resource Managers specify which zones to use for the forward and reverse zones and which Remote Servers to use for those zones. The dynamic-dns Remote Servers specify how to access the DNS Servers.

## Configuring Dynamic DNS

Before you configure CAR you need to gather information about your DNS environment. For a given Resource Manager you must decide which forward zone you will be updating for sessions the resource manager will manage. Given that forward zone, you must determine the IP address of the primary DNS server for that zone. If the dynamic DNS updates will be protected with TSIG keys, you must find out the name and the base64 encoded value of the secret for the TSIG key. If the resource manager should also update the reverse zone (ip address to host mapping) for sessions, you will also need to determine the same information about the primary DNS server for the reverse zone (IP address and TSIG key).

If using TSIG keys, use **aregcmd** to create and configure the keys. You should set the key in the Remote Server or the Resource Manager, but not both. Set the key on the Remote Server if you want to use the same key for all of the zones accessed through that Remote Server. Otherwise, set the key on the Resource Manager. That key will be used only for the zone specified in the Resource Manager.

**Step 1** Launch **aregcmd**.

**Step 2** Create the dynamic-dns TSIG Keys:

```
cd /Radius/Advanced/DDNS/TSIGKeys
add foo.com
```

This example named the TSIG Key, **foo.com**, which is related to name of the example DNS server we use. You should choose a name for TSIG keys that reflects the DDNS client-server pair (for example, **foo.bar** if the client is **foo** and the server is **bar**), but you should use the name of the TSIG Key as defined in the DNS server.

**Step 3** Configure the TSIG Key:

```
cd foo.com
set Secret <base64-encoded string>
```

The Secret should be set to the same base64-encoded string as defined in the DNS server. If there is a second TSIG Key for the primary server of the reverse zone, follow these steps to add it, too.

**Step 4** Use **aregcmd** to create and configure one or more dynamic-dns Remote Servers.

**Step 5** Create the dynamic-dns remote server for the forward zone:

```
cd /Radius/RemoteServers
add ddns
```

This example named the remote server *ddns* which is the related to the remote server type. You can use any valid name for your remote server.

**Step 6** Configure the dynamic-dns remote server:

```
cd ddns
set Protocol dynamic-dns
set IPAddress 10.10.10.1 (ip address of primary dns server for zone)
set ForwardZoneTSIGKey foo.com
set ReverseZoneTSIGKey foo.com
```

If the reverse zone will be updated and if the primary server for the reverse zone is different than the primary server for the forward zone, you will need to add another Remote Server. Follow the previous two steps to do so. Note that the IP Address and the TSIG Key will be different.

You can now use **aregcmd** to create and configure a resource manager of type dynamic-dns.

**Step 7** Create the dynamic-dns resource manager:

```
cd /Radius/ResourceManagers
```

```
add ddns
```

This example named the service `ddns` which is related to the resource manager type but you can use any valid name for your resource manager.

**Step 8** Configure the dynamic-dns resource manager.

```
cd ddns
```

```
set Type dynamic-dns
```

```
set ForwardZone foo.com
```

```
set ForwardZoneServer DDNS
```

Finally, reference the new resource manager from a session manager. Assuming that the example configuration was installed, the following step will accomplish this. If you have a different session manager defined you can add it there if that is appropriate.

**Step 9** Reference the resource manager from a session manager:

```
cd /Radius/SessionManagers/session-mgr-1/ResourceManagers
```

```
set 5 DDNS
```

**Note**

The Property `AllowAccountingStartToCreateSession` must be set to `TRUE` for dynamic DNS to work.

**Step 10** Save the changes you have made.

## Testing Dynamic DNS with radclient

After the Resource Manager has been defined it must be referenced from the appropriate Session Manager. You can use **radclient** to confirm that dynamic DNS has been properly configured and is operational.

To test Dynamic DNS using `radclient`, follow these steps:

**Step 1** Launch `aregcmd` and set the trace to level 4.

```
aregcmd
```

Login to the Cisco AR 4.1 server as an administrative user.

```
trace 4
```

**Step 2** Launch `radclient`.

```
cd /opt/CSCOar/bin
```

```
radclient
```

**Step 3** Create an Accounting-Start packet

```
acct_request Start username
```

Example:

```
set p [ acct_request Start bob ]
```

**Step 4** Add a Framed-IP-Address attribute to the Accounting-Start packet

**Step 5** Send the Accounting-Start packet

```
$p send
```

**Step 6** Check the `aregcmd` trace log and the dns server to verify that the host entry was updated in both the forward and reverse zones.

---

## Dynamic Service Authorization Feature

Typically, CAR does not allow sending another Access-Request to remote server after the user is connected to the LDAP servers for user authentication. The Dynamic Service Authorization feature allows you to access external databases such as LDAP and Oracle first to know which remote servers authenticated services need to be relayed. This feature enables CAR to determine whether to send access-accept back to the client or to send another access-request to the remote server such as LDAP and Oracle. CAR is able to perform this activity multiple times in a single access-request.

## Configuring Dynamic Service Authorization Feature

Configuring the dynamic service authorization involves:

- [Setting up the Environment Variable](#)
- [Configuring the Script](#)

### Setting up the Environment Variable

Before configuring the dynamic service authorization feature, you must set the following three environment variables in CAR:

- **Re-Authentication-Service**  
When the Re-Authentication-Service is set, the server directs the request to the specified reauthentication service for processing.
- **Re-Authorization-Service**  
When the Re-Authorization-Service is set, the server directs the request to the specified reauthorization service for processing.
- **Re-Accounting-Service**  
When the Re-Accounting-Service is set, the server directs the request to the specified reaccounting service for processing.

You can set the environmental variable by using scripts. See [Writing the Script, page 10-2](#) for more information.

**Note**

When using the same service for re-authentication and re-authorization, a loop can occur in these services. The loop count, by default is 10. You can change the loop count using the **Dynamic-Service-Loop-Limit** environment variable.

Following is a sample procedure for setting the environment variable:

```
proc dynamicservice { request response environ } {
$environ put Re-Authentication-Service "local-users"
$environ put Re-Authorization-Service "local-users"
}
```

You can append this procedure by copying it into **tclscript.tcl** that is located in **/opt/CSCOAr/scripts/radius/tcl** directory, or to the location that you chose when you installed CAR. You can also use this procedure as a separate script file and configure the script accordingly. See [Adding the CheckEap.tcl Script, page 14-15](#) for more information on configuring the TCL script.

## Configuring the Script

To configure the script for the dynamic service authorization feature:

- 
- Step 1** Launch **aregcmd**.
- ```
aregcmd
```
- Step 2** Change directory to **/Radius/Scripts**.
- ```
cd /Radius/Scripts
```
- Step 3** Enter **dynamicservice**.
- Step 4** Change the directory to **dynamicservice**.
- ```
cd dynamicservice
```
- You get the following output:
- ```
[ //localhost/Radius/Scripts/dynamicservice ]
Name = dynamicservice
Description =
Language =
```
- Step 5** Set the Language property to TCL.
- ```
Set Language TCL
```
- Step 6** Set the filename property to **tclscript.tcl**.
- ```
Set Filename tclscript.tcl
```
- Step 7** Set the EntryPoint property to **dynamicservice**.
- ```
Set EntryPoint dynamicservice
```

The following is an example of the script configuration:

```
cd /Radius
set IncomingScript dynamicservice
[ //localhost/Radius ]
IncomingScript~ = dynamicservice
DefaultAuthenticationService~ = local-users
DefaultAuthorizationService~ = local-users
```

**Step 8** Enter **Save** to save the configuration.

The following shows a sample trace:

```

10/30/2008 12:32:02.258: P577: Packet received from 127.0.0.1
10/30/2008 12:32:02.259: P577: Packet successfully added
10/30/2008 12:32:02.259: P577: Trace of Access-Request packet
10/30/2008 12:32:02.259: P577:   identifier = 9
10/30/2008 12:32:02.259: P577:   length = 61
10/30/2008 12:32:02.259: P577:   reqauth =
b6:89:41:52:6e:d4:86:37:4a:aa:9b:27:1f:74:ff:05
10/30/2008 12:32:02.259: P577:   User-Name = bob
10/30/2008 12:32:02.259: P577:   User-Password =
2b:4a:f0:c8:95:f1:ad:e5:52:d4:83:0f:45:2b:2b:70
10/30/2008 12:32:02.259: P577:   NAS-Port = 2
10/30/2008 12:32:02.260: P577:   NAS-Identifier = localhost
10/30/2008 12:32:02.260: P577: Running Server's IncomingScript: dynamicservice
10/30/2008 12:32:02.261: P577:   Tcl: environ put Re-Authentication-Service local-users
-> OK
10/30/2008 12:32:02.261: P577:   Tcl: environ put Re-Authorization-Service local-users
-> OK
10/30/2008 12:32:02.261: P577: Using Client: localhost
10/30/2008 12:32:02.262: P577: Using NAS: localhost (127.0.0.1)
10/30/2008 12:32:02.262: P577: Request is directly from a NAS: TRUE
10/30/2008 12:32:02.262: P577: Authenticating and Authorizing with Service local-users
10/30/2008 12:32:02.262: P577: Getting User bob's UserRecord from UserList Default
10/30/2008 12:32:02.263: P577: user list user bob's password matches
10/30/2008 12:32:02.263: P577: Processing UserGroup PPP-users's check items
10/30/2008 12:32:02.263: P577: User bob is part of UserGroup PPP-users
10/30/2008 12:32:02.263: P577: Merging UserGroup PPP-users's BaseProfiles into response
dictionary
10/30/2008 12:32:02.264: P577: Merging BaseProfile default-PPP-users into response
dictionary
10/30/2008 12:32:02.264: P577: Merging attributes into the Response Dictionary:
10/30/2008 12:32:02.264: P577:   Adding attribute Service-Type, value = Framed
10/30/2008 12:32:02.264: P577:   Adding attribute Framed-Protocol, value = PPP
10/30/2008 12:32:02.264: P577:   Adding attribute Framed-Routing, value = None
10/30/2008 12:32:02.264: P577:   Adding attribute Framed-MTU, value = 1500
10/30/2008 12:32:02.264: P577:   Adding attribute Framed-Compression, value = VJ TCP/IP
header compression
10/30/2008 12:32:02.264: P577:   Adding attribute Ascend-Idle-Limit, value = 1800
10/30/2008 12:32:02.265: P577: Merging UserGroup PPP-users's Attributes into response
Dictionary
10/30/2008 12:32:02.265: P577: Merging attributes into the Response Dictionary:
10/30/2008 12:32:02.265: P577: Authenticating and Authorizing with Service local-users
10/30/2008 12:32:02.265: P577: Getting User bob's UserRecord from UserList Default
10/30/2008 12:32:02.266: P577: user list user bob's password matches
10/30/2008 12:32:02.266: P577: Processing UserGroup PPP-users's check items
10/30/2008 12:32:02.266: P577: User bob is part of UserGroup PPP-users
10/30/2008 12:32:02.266: P577: Merging UserGroup PPP-users's BaseProfiles into response
dictionary
10/30/2008 12:32:02.266: P577: Merging BaseProfile default-PPP-users into response
dictionary
10/30/2008 12:32:02.266: P577: Merging attributes into the Response Dictionary:
10/30/2008 12:32:02.266: P577:   Replacing attribute Service-Type, new value = Framed
10/30/2008 12:32:02.267: P577:   Replacing attribute Framed-Protocol, new value = PPP
10/30/2008 12:32:02.267: P577:   Replacing attribute Framed-Routing, new value = None
10/30/2008 12:32:02.267: P577:   Replacing attribute Framed-MTU, new value = 1500

```