GNU Radius Reference Manual

GNU Radius Reference Manual

version 1.6, 6 December 2008

Sergey Poznyakoff

This manual documents GNU Radius (version 1.6, 6 December 2008).
Published by:

GNU Press
a division of the
Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301 USA

Website: www.gnupress.org
General: press@gnu.org
Orders: sales@gnu.org
Tel: 617-542-5942
Fax: 617-542-2652

Copyright © 1999, 2000, 2001, 2002, 2003 Free Software Foundation

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being "A GNU Manual", and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled "GNU Free Documentation License".

(a) The FSF's Back-Cover Text is: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

Cover art by Etienne Suvasa. Cover design by Jonathan Richard.

Introduction to Radius

GNU Radius is a software package that provides authentication and accounting services. The acronym RADIUS stands for *Remote Authentication Dial In User Service* and (in that form) usually denotes the underlying protocol name.

Historically, RADIUS servers were used as a means to authenticate the user coming from a dial-in connection, but GNU Radius is much more than an authentication system: it is an advanced, customizable, and extensible system for controlling access to the network.

GNU Radius has several built-in authentication and accounting methods. When these methods are not enough, it allows the administrator to implement any new method she deems convenient.

The GNU Radius package includes the server program, `radiusd`, which responds to authentication and accounting requests, and a set of accompanying programs designed to monitor the activity of the server and analyze the information it provides.

Overview

To illustrate what GNU Radius does, let's consider an imaginary internet service provider. Our provider has two *network access servers* (NASes for short)—i.e., two pieces of equipment which directly accept users' connections—and a core router that connects the ISP's internal network with the Internet backbone.

When a user connects to a NAS, the server must verify that the user is actually registered and that the credentials she has supplied are correct. This first step is called *authentication*.

Upon authenticating the user, the NAS must determine which services the user is permitted to use and to what extent the user may use them. This second step is called *authorization*.

When the first two stages have been successfully completed, the NAS takes the third step and establishes the connection between the user and the main server. This connection is called a *user session*. For the purposes of *accounting*, the NAS remembers the exact time of the start of the session. When the session is terminated, the duration of the session and the number of bytes transferred are recorded as well.

All three tasks can be accomplished by the use of user and accounting databases on each terminal server. However, this is not convenient, and it is error-prone in that the maintenance of separate databases for the same users is not a trivial task. What is worse, as the number of terminal servers grows, this maintenance problem becomes more difficult.

How Does RADIUS Perform These Tasks?

RADIUS allows an administrator to keep authentication and accounting data in a single place, no matter how many network access servers are actually present. Using RADIUS, NASes instead communicate with this central server to perform authentication and accounting, thus easing the burden on the system administrator.

Let's return to our imaginary ISP. Suppose it runs a RADIUS daemon on its central server. Each NAS runs *client software* to communicate with the RADIUS server by sending *radius packets*.

An average user session life cycle looks as follows.

A user connects to the nearest NAS and supplies his login and password. The NAS forms an authentication request and sends it to the RADIUS server.

The RADIUS server verifies the user's credentials and finds them sufficient. It then retrieves the user's authorization information from its database, packages it into an *acknowledgement packet*, and then sends it back to the NAS

The NAS receives the acknowledgement packet and starts the user session. The information brought with the packet tells the NAS to establish a connection between the core router and the user, and to assign the user a certain IP address. Having established the session, the NAS informs the RADIUS server by sending it an *accounting start packet*. The server acknowledges the receipt of the accounting packet.

Now suppose that after some time the user decides to break the connection. The NAS notices this and terminates the user's session. The NAS then sends an *accounting stop packet* to the RADIUS server to mark this event. Again, the server acknowledges the receipt of the packet.

RADIUS Attributes

Attributes are means of passing the information between the NAS and the server. Basically, an attribute is an integer number that identifies some piece of information. A set of *properties* are associated with each attribute, specifying the way to interpret the attribute. The most important property is the *data type*, which declares the type of data that the attribute identifies (*character string*, *integer number*, *IP address*, or *raw binary data*).

The information to be transmitted with the request is packaged in a set of *attribute-value pairs* (or A/V pairs for short). Such pairs consist of attribute numbers and the associated data.

RADIUS Packets

There exist two basic kinds of RADIUS packets: authentication and accounting packets. Each of them is subdivided into *requests* and *replies*.

Authentication requests are sent from the NAS to the RADIUS server and contain the information necessary to check the identity of the user. The minimum set of data in such packets consists of the user login name, user password, and NAS IP or identifier.

Authentication replies are sent by the RADIUS server and contain the reply code and a set of additional attributes. According to their reply code the authentication replies are subdivided into *authentication acknowledgements*, *authentication rejections*, and *authentication challenges*.

An authentication acknowledgement packet is sent to the NAS if the credentials supplied with the authentication request were correct. This kind of packet tells the NAS to establish a normal user session. The additional attributes in such packets carry the *authorization data*, i.e., they determine which kind of service the user is to be provided.

An authentication rejection is sent to the NAS if the authentication has failed. This packet forbids the NAS to provide any service to the user. The additional attributes may carry descriptive text to be displayed as an explanation to the user for the failure of his request.

Finally, an authentication challenge packet is sent to the NAS if the supplied credentials did not suffice to establish the authenticity of the user. This means that the dialog between the NAS and the RADIUS server continues. As the RADIUS server asks for additional authentication credentials, the NAS acts as a liaison, passing server requests to the user and sending user replies back to the server. Such a dialog ends when the RADIUS server sends either an acknowledgement packet or a rejection packet.

An *accounting request* is sent to the server when the NAS wishes to report some event in the user session: the start of the session, session termination, etc. The attributes carry the actual information about the event.

For each accounting request that has been received and successfully processed, the RADIUS server sends back an *accounting acknowledgement*. This packet carries no attributes, but simply informs the NAS that the information it had sent was received.

Occasionally, a RADIUS server may fail to receive incoming requests or may fail to process them due to high server load. In order to prevent such requests from being lost, the NAS retransmits the request if no response from the server is received within a predefined interval of time (a *timeout* interval). Usually the NAS is configured in such a way that it continues retransmitting failed requests until either it receives a reply from the server or a predefined number of *retries* are exhausted, whichever occurs first. Furthermore, a NAS may be configured to communicate with a set of *backup* RADIUS servers. In this case it applies the described process to each server from the set, until one of them responds or the set is exhausted.

1 Naming Conventions

This chapter describes file naming conventions used throughout this document.

Programs from the GNU Radius package use the following directories to store various configuration and log files:

Configuration or database directory
: A directory where all configuration files are stored.

Log directory
: A directory where `radiusd` stores its log files.

Accounting directory
: A directory where `radiusd` stores accounting detail files (see Section 7.2 [Detailed Request Accounting], page 81).

Data directory
: A directory where shared data files are stored, such as `Rewrite` or `Scheme` source files.

The default locations of these directories are determined at compile time. Usually these are:

Directory	Short name	Default location
Configuration directory	'`raddb`'	/usr/local/etc/raddb
Log directory	'`radlog`'	/var/log
Accounting directory	'`radacct`'	/var/log/radacct
Data directory	'`datadir`'	/usr/local/share/radius/1.6

These locations may differ depending on your local site configuration.

Throughout this document we will refer to these directories by their short names. For example, when we say:

```
... this information is contained in file 'raddb/sqlserver'
```

we actually mean '`/usr/local/etc/raddb/sqlserver`'.

To get the default directory names that your version of Radius was compiled with, run `radiusd --version`.

Locations of these directories may be overridden by specifying the appropriate command line options. For example, any program from the GNU Radius package accepts the command line option '`-d`' or '`--directory`', which introduces the configuration directory path.

2 How Radius Operates

The main purpose of GNU Radius is to centralize authentication of users coming from various network stations, pursuant to the RADIUS specification. Its primary usage is for dial-in users, though it can be used for any kind of network connection.

2.1 Attributes

Information carried by RADIUS requests is stored as a list of *attribute-value pairs*. Each pair consists of an *attribute number* and an *attribute value*. The *attribute number* identifies the type of information the pair carries, and the *attribute value* keeps the actual data.

The value part of an attribute can contain data of one of the following types:

Integer A 32-bit unsigned integer value.

IP-number
An IPv4 IP-number.

String A character string up to 253 characters long.

For convenience, the attributes and the values of some frequently used integer attributes are given symbolic names. These names are assigned to attributes and values in the dictionary file (see Section 4.2 [dictionary file], page 40).

Attribute numbers range from 1 to 255. Attributes with numbers greater than 255 are used internally by the server and cannot be sent to the NAS.

The *vendor-specific* attribute number 26 is special, allowing vendors of the NAS hardware or software to support their own extended attributes. Section 13.1.26 [Vendor-Specific], page 175.

Each attribute has a set of properties associated with it. The properties are:

Usage flags
These flags determine the usage of the attribute in the configuration files '`huntgroups`', '`hints`', and '`users`'.

Propagation
When a RADIUS server functions in proxy mode, it uses the *propagation flag* to determine which attributes from the reply packet should be passed back to the requesting NAS (see Section 2.4.2.1 [Proxy Service], page 13).

additivity Some configuration rules may cause the addition of new A/V pairs to the incoming request. Before the addition of a new pair, `radiusd` scans the request to see if it already contains a pair with the same attribute. If it does, the value of the *additivity* determines the following additional actions:

None — The old pair is retained in the request; the new pair is not added to it.

Replace — The old pair is retained in the request, but its value is replaced with that of the new pair.

Append — The new pair is appended to the end of the pair list.

Attributes are declared in the '`raddb/dictionary`' file. For a detailed description, see Section 4.2.4 [ATTRIBUTE], page 41. For information about particular attributes, see Chapter 13 [Attribute List], page 165.

2.2 RADIUS Requests

The term *request* refers to both the authentication/accounting request packet from a NAS to a RADIUS server and the response packet that the server sends back to the NAS.

Each request contains the following fields:

'`Code`' The code field identifies the type of the request.

'`Identifier`'
The number in the range 0–255 used to match the request with the reply.

'`Length`' The length of the request packet.

'`Authenticator`'
The 16-byte hash value used to authenticate the packet.

'`Attributes`'
The list of attribute-value pairs carrying actual information about the request.

2.2.1 Authentication Requests

A NAS sends authentication requests (packets with code field set to Access-Request) to a RADIUS server when a user is trying to connect to that NAS. Such requests convey information used to determine whether a user is allowed access to the NAS, and whether any special services are requested for that user.

An Access-Request must contain a `User-Name` attribute Section 13.1.24 [User-Name], page 174. This packet should contain a `NAS-IP-Address` attribute, a `NAS-Identifier` attribute, or both. It also must contain either a `User-Password` attribute or a `CHAP-Password` attribute. These attributes are passed after being encoded using a method based on the RSA Message Digest Algorithm MD5.

The Access-Request should contain a `NAS-Port` or `NAS-Port-Type` attribute or both, unless the type of access being requested does not involve a port or the NAS does not distinguish among its ports.

Upon receiving an Access-Request packet for a particular user and authenticating that user, the RADIUS server replies to the NAS that has sent the packet with any one of the following packets:

- Access-Accept
- Access-Reject
- Access-Challenge

GNU Radius replies with an Access-Accept packet when it has successfully authenticated the user. Such a reply packet provides the configuration information necessary to begin delivery of service to the user.

GNU Radius replies with an Access-Reject packet when it is unable to authenticate the user. Such a packet may contain a descriptive text encapsulated in one or more `Reply-Message` attributes. The NAS may display this text along with its response to the user.

GNU Radius replies with an Access-Challenge packet when it needs to obtain more information from the user in order to determine the user's authenticity or to determine the kind of service to be provided to the user.

An Access-Challenge packet may include one or more `Reply-Message` attributes, and it may or may not include a single `State` attribute. No other attributes are permitted in an Access-Challenge packet.

Upon receipt of an Access-Challenge, the Identifier field is matched with a pending Access-Request. Additionally, the Response Authenticator field must contain the correct response for the pending Access-Request. In the event of an invalid packet, GNU Radius discards the offending packet and issues the appropriate log message.

If the NAS does not support challenge/response, it treats an Access-Challenge as though it had received an Access-Reject instead. Otherwise, upon receipt of a valid Access-Challenge the NAS prompts the user for a response, possibly displaying the text message provided in the `Reply-Message` attributes of the request. It then sends its original Access-Request with a new request ID and request authenticator, along with the `User-Password` attribute replaced by the encrypted user's response, and including the `State` attribute from the Access-Challenge, if any.

2.2.2 Accounting Requests

Accounting-Request packets are sent from a NAS to a RADIUS server to allow for accounting of a service provided to a user.

Upon receipt of an Accounting-Request packet, the server attempts to record it (see Chapter 7 [Accounting], page 81), and if it succeeds in doing so, it replies with an Accounting-Response packet. Otherwise, it sends no reply, which then causes the NAS to retransmit its request within a preconfigured interval of time. Such retransmits will continue until either the server responds with an Accounting-Response packet or a preconfigured number of retransmits is reached, whichever occurs first.

Any attribute valid in an Access-Request or Access-Accept packet is also valid in an Accounting-Request packet, except the following attributes, which are never present in any Accounting-Request packet:

- `User-Password`
- `CHAP-Password`
- `Reply-Message`
- `State`

Either a `NAS-IP-Address` or a `NAS-Identifier` must be present in an Accounting-Request packet. It should contain either a `NAS-Port` or a `NAS-Port-Type` attribute (or both), unless the service does not involve a port or the NAS does not distinguish among its ports.

If the Accounting-Request packet includes a `Framed-IP-Address`, that attribute *must* contain the actual IP of the user.

There are five types of accounting packets, differentiated by the value of the `Acct-Status-Type` attribute. These are:

Session Start Packet

The session start packet is sent after the user has successfully passed the authentication and has started to receive the requested service. It must contain at least following attributes:

- `Acct-Status-Type = Start`
- `User-Name`
- `Acct-Session-Id`
- `NAS-IP-Address`
- `NAS-Port-Id`

Session Stop Packet

The session stop packet is sent after the user has disconnected. It conveys the information about the duration of the session, number of octets transferred, etc. It must contain at least the following attributes:

- `Acct-Status-Type = Stop`
- `User-Name`
- `NAS-IP-Address`
- `Acct-Session-Id`

The last three of them are used to find the corresponding session start packet.

Keepalive Packet

The keepalive packet is sent by the NAS when it obtains some new information about the user's session, e.g. it has determined its IP or has changed the connection speed. The packet must contain at least the following attributes:

- `Acct-Status-Type = Alive`
- `User-Name`
- `NAS-IP-Address`
- `Acct-Session-Id`

Accounting-Off Packet
: By sending this packet, the NAS requests that `radiusd` mark all sessions registered from this particular NAS as finished. Receiving this packet usually means that the NAS is to be shut down, or is about to change its configuration in a way that requires all currently opened sessions to be closed. The packet must contain at least the following attributes:

  - `Acct-Status-Type = Accounting-Off`
  - `NAS-IP-Address`

Accounting-On Packet
: By sending this packet, the NAS informs `radiusd` that it is ready to accept the incoming connections. Usually this packet is sent after startup, or after a major reconfiguration of the NAS. It must contain at least the following attributes:

  - `Acct-Status-Type = Accounting-On`
  - `NAS-IP-Address`

2.3 Matching Rule

A record in the GNU Radius database describing a particular rule for matching an incoming request is called a *matching rule*. Each such rule defines an action to be taken when the match occurs.

The matching rule consists of three distinct parts:

Label
: This is used to identify the rule. The special usernames `DEFAULT` and `BEGIN` are reserved. These will be described in detail below.

*Left-Hand Side (*LHS*)*
: The list of attribute-value pairs used for matching the profile against an incoming request.

*Right-Hand Side (*RHS*)*
: The list of attribute-value pairs that define the action to be taken if the request matches LHS.

The following GNU Radius configuration files keep data in a matching rule format: '`hints`', '`huntgroups`', and '`users`'. Although they keep data in a similar format, the rules that are used to match incoming requests against the contents of these files differ from file to file. The following section describes these rules in detail.

2.4 Processing Requests

Upon receiving a request, `radiusd` applies to it a number of checks to determine whether the request comes from an authorized source. If these checks succeed, the request is processed and answered. Otherwise, the request is dropped and corresponding error message is issued (see Chapter 8 [Logging], page 85).

The following checks are performed:

Check if the username is supplied.
: If the packet lacks the `User-Name` attribute, it is not processed.

Check if the NAS is allowed to speak.
: The source IP of the machine that sent the packet is looked up in the '`clients`' file (see Section 4.3 [clients file], page 45). If no match is found, the request is rejected.

Compute the encryption key.
: Using the data from the packet and the shared key value from the '`clients`' file, Radius computes the MD5 encryption key that will be used to decrypt the value of the `User-Password` attribute.

Process user-name hints.
: *User-name hints* are special rules that modify the request depending on the user's name and her credentials. These rules allow an administrator to divide users into distinct groups, each group having its own authentication and/or accounting methods. The user-name hints are stored in '`raddb/hints`' (see Section 4.6 [hints file], page 50).

Process huntgroup rules.
: *Huntgroup rules* allow an administrator to segregate incoming requests depending on the NAS and/or port number they came from. These rules are stored in '`raddb/huntgroups`' (see Section 4.7 [huntgroups file], page 50).

Determine whether the request must be proxied to another RADIUS server.
: The requests pertaining to another realm are immediately forwarded to the remote RADIUS server for further processing. See Section 2.4.2 [Proxying], page 13, for the description of this process.

Process individual user profiles
: This step applies only to authentication requests.

2.4.1 Checking for Duplicate Requests

As described above (see Chapter 2 [Operation], page 7), a NAS may decide to retransmit the request under certain circumstances. This behavior ensures that no requests are lost. For example, consider the following scenario:

1. The NAS sends a request to the server.
2. The server processes it and sends back the reply.
3. The reply is lost due to a network outage, or the load average of the NAS is too high and it drops the response.
4. The NAS retransmits the request.

Thus the RADIUS server will receive and process the same request twice. This probably won't do any harm if the request in question is an authentication one, but for accounting requests it will lead to duplicate accounting. To avoid such an undesirable effect, `radiusd` keeps a queue of received requests. When an incoming request arrives, `radiusd` first scans the request queue to see if the request is a duplicate. If so, it drops the request; otherwise, it inserts the request into the queue for processing. After the request is completed, it will still reside in the queue for a preconfigured interval of time (see Section 4.1.3 [auth], page 28, parameter `request-cleanup-delay`).

By default, `radiusd` considers two requests to be equal if the following conditions are met:

1. Both requests come from the same NAS.
2. They are of the same type.
3. The request identifier is the same for both requests.
4. The request authenticator is the same for both requests.

Additionally, `radiusd` may be configured to take into account the contents of both requests. This may be necessary, since some NASes modify the request authenticator or request identifier before retransmitting the request, so the method described above fails to recognize the request as a duplicate. This *extended comparison* is described in detail in Section 5.1 [Extended Comparison], page 67.

2.4.2 Proxying

Proxying is a mode of operation where a RADIUS server forwards incoming requests from a NAS to another RADIUS server, waits for the latter to reply, and then forwards the reply back to the requesting NAS. A common use for such operation mode is to provide *roaming* between several internet service providers (ISPs). Roaming permits ISPs to share their resources, allowing each party's users to connect to other party's equipment. Thus, users traveling outside the area of one ISP's coverage are still able to access their services through another ISP.

2.4.2.1 Proxy Service

Suppose the ISP '`Local`' has a roaming arrangement with the ISP '`Remote`'. When the user of '`Remote`' dials in to the NAS of '`Local`', the NAS sends the authentication request to the '`Local`' RADIUS server. The server then determines that this is a roaming user, stores a copy of the request in its internal queue, and forwards the request to the '`Remote`' RADIUS server for

processing. Thus, the '`Local`' RADIUS server acts as a client for the '`Remote`' RADIUS server.

When the '`Remote`' RADIUS server responds, the '`Local`' RADIUS server receives the response, and passes it back to the NAS. The copy of the request from the server's queue determines which NAS originated the request. Before passing the request back to the NAS, the server removes information specific to the '`Remote`' site, such as `Framed-IP-Address`, `Framed-Netmask`, etc. Only the attributes marked with a '`propagation`' flag (see Section 2.1 [Attributes], page 7) are passed back to the NAS. After removing site-specific attributes, the '`Local`' RADIUS server passes the request through its user profiles (see Section 2.4.5 [User Profiles], page 16) to insert any local, site-specific information that might be needed. Finally, it passes the reply back to the NAS.

Proxied accounting requests are processed in a similar manner, except that no attribute filtering takes place, as accounting responses do not carry any A/V pairs.

This example illustrates only the simplest *proxy chain*, consisting of two servers; real-life proxy chains may consist of several servers. For example, our '`Remote`' RADIUS server might also act as a proxy, forwarding the request to yet another RADIUS server, and so on.

Note that when the accounting request passes through a chain of forwarding servers, the accounting records are *stored on all servers in the chain*.

2.4.2.2 Realms

GNU Radius determines which server a request must be forwarded to by the request's *authentication realm*. There are three kinds of realms:

1. A *named realm* is the part of a user name following the at sign ('`@`'). For example, if the user name is '`jsmith@this.net`', then '`this.net`' is the realm. The named realms can be cascaded; e.g., a request with user name '`jsmith@this.net@remote.net`' will first be forwarded to the RADIUS server of the realm '`remote.net`', which in turn will forward it to '`this.net`'.
2. A *default realm* defines the server to which the requests for realms not mentioned explicitly in the configuration are forwarded.
3. An *empty realm* defines the server to which the requests *without* explicitly named realms are forwarded. If the configuration does not define an empty realm, such requests are processed by the server itself.

GNU Radius keeps the information about the realms it serves in the '`raddb/realms`' configuration file (see Section 4.8 [realms file], page 51).

2.4.3 Hints

User-name hints are special rules that modify the incoming request depending on the user name and its credentials. Hints are stored as a list of *matching*

rules (see Section 2.3 [Matching Rule], page 11). Upon receiving a request, `radiusd` scans the hint entries sequentially, comparing each rule's label with the value of the `User-Name` attribute from the request. If they coincide, then `radiusd` appends the contents of the rule's RHS to the request's pair list.

The two user names must match exactly in order for a hint to take effect, unless the hint's checklist contains either the `Prefix` or the `Suffix` attribute. The special name '`DEFAULT`' or '`DEFAULT%d`' (where *%d* denotes any decimal number), used as a hint's label, matches any user name.

Two special attributes, `Prefix` and `Suffix`, may be used in LHS to restrict the match to a specified part of a user name. Both are string attributes. The `Prefix` instructs `radiusd` to accept the hint only if the user name begins with the given prefix. Similarly, `Suffix` instructs `radiusd` to accept the hint only if the user name ends with the given suffix. A hint may contain both `Prefix` and `Suffix` attributes.

In addition to these two attributes, a hint's LHS may contain `User-ID` and `Group` attributes.

The following attributes, when used in a hint's RHS have special meaning. They are not appended to the request pair list. Instead, they are removed after completing their function:

`Fall-Through`
: If this attribute is present and is set to `Yes`, `radiusd` continues scanning the hints after processing the current entry. This allows `radiusd` to apply several hints to a single packet.

`Rewrite-Function`
: If this attribute is present, the specified rewrite function is invoked.

`Replace-User-Name`
: The value of this attribute is expanded (see Section 4.14 [Macro Substitution], page 64) and replaces the value of the `User-Name` attribute from the request.

Hint rules are defined in the '`raddb/hints`' file (see Section 4.6 [hints file], page 50).

2.4.4 Huntgroups

Huntgroups are special rules that allow an administrator to provide alternate processing of certain incoming requests depending on the NAS IP and port number they come from. These rules are stored as a list of matching rules (see Section 2.3 [Matching Rule], page 11).

Upon receiving a request, `radiusd` scans this list sequentially until it finds an entry such that the conditions set forth in its LHS are matched by the request. If such an entry is found, `radiusd` verifies that the request meets the conditions described by RHS. If it does not, the request is rejected. In

short, a huntgroup requires that any request matching its LHS must match also its RHS.

The label part of the rule is not used in comparisons; instead it is used to label huntgroups. All entries with the same label form a single huntgroup. The special attribute `Huntgroup-Name` can be used to request a match against a particular huntgroup (see Section 13.3.12 [Huntgroup-Name], page 186).

Huntgroup rules are defined in the '`raddb/huntgroups`' file (see Section 4.7 [huntgroups file], page 50).

2.4.5 User Profiles

User profiles are *per-user* matching rules (see Section 2.3 [Matching Rule], page 11). All incoming authentication requests are compared with the user profiles after they have passed both hints and huntgroups. `radiusd` selects the user profiles whose label matches the value of the `User-Name` attribute from the incoming request.

The selected profiles form the list of authentication rules for the request. In order for a profile to be selected, its label must either coincide literally with the `User-Name` value, or be one of the special labels, `DEFAULT` or `BEGIN`.

Rules in an authentication list are ordered as follows: first go all the profiles with the `BEGIN` label, followed by the profiles whose labels match the `User-Name` literally, followed finally by the rules labeled with the `DEFAULT`.[1]

Within each of the three sublists, the rules preserve the order in which they appear in the '`raddb/users`' file. Once the list is constructed, it is scanned sequentially until the rule is found whose LHS matches the incoming request. If no such rule is found, the authentication fails. Otherwise, the contents of its RHS are appended to the reply list being constructed. If the RHS of the matched rule contains the attribute `Fall-Through` with the value `Yes`, the matching continues. When the list is exhausted, the authentication result is sent back to the NAS along with the A/V pairs collected in the reply list.

User profiles are defined in the '`raddb/users`' file (see Section 4.9 [users file], page 52).

[1] For compatibility with other radius implementations, GNU Radius treats profile labels in the form `DEFAULT%d`, where *%d* represents a decimal number, in the same way it treats `DEFAULT` labels. The same applies to `BEGIN` labels.

3 How to Start the Daemon.

When started `radiusd` uses the configuration values from the following sources (in order of increasing precedence):

- Compiled-in defaults
- '`raddb/config`' file.
- Command line arguments

Whenever a command line options has its equivalent in config file the use of this equivalent should be preferred (see Section 4.1 [config file], page 22).

The following command line options are accepted:

'`-A`'
'`--log-auth-detail`'
: Enable detailed authentication logging. When this option is specified each authentication request is logged to the file '`radacct/NASNAME/detail.auth`', where NASNAME is replaced by the short name of the NAS from '`raddb/naslist`' Chapter 1 [Naming Conventions], page 5.

  Config file equivalent: `auth { detail yes; };`.

'`-a DIR`'
'`--acct-directory DIR`'
: Specify accounting directory.

  Config file equivalent: `option { acct-dir DIR; };`.

'`-b`'
'`--dbm`'
: Enable DBM support.

  Config file equivalent: `usedbm yes;`.

'`-d DIR`'
'`--config-directory DIR`'
'`--directory D`'
: Specify alternate configuration directory. Default is '`/usr/local/etc/raddb`'.

'`-f`'
'`--foreground`'
: Stay in foreground. We recommend to use it for debugging purposes only.

'`-i IP`'
'`--ip-address`'
: Specifies the IP address `radiusd` will listen on. If this option is not specified, the program will listen on all IP addresses, assigned to the machine it runs on.

  Config file equivalent: `option { source-ip IP; };`.

Note that `listen` statement in '`raddb/config`' provides a better control over IP addresses to listen on (see Section 4.1.3 [auth], page 28, and see Section 4.1.4 [acct], page 30).

'`-L`'
'`--license`'
Display GNU General Public License and exit.

'`-l DIR`'
'`--logging-directory DIR`'
Specify alternate logging directory.

Config file equivalent: `option { log-dir DIR; };`.

'`-mb`'
'`--mode b`' "Builddbm" mode. Builds a DBM version of a plaintext users database. Section 11.8 [Builddbm], page 130.

'`-mc`'
'`--mode c`' Check configuration files and exit. All errors are reported via usual log channels.

'`-mt`'
'`--mode t`' Test mode. In this mode `radiusd` starts an interactive interpreter which allows to test various aspects of its configuration.

'`-N`'
'`--auth-only`'
Process only authentication requests.

'`-n`'
'`--do-not-resolve`'
Do not resolve IP addresses for diagnostic output. This can reduce the amount of network traffic and speed up the server.

Config file equivalent: `option { resolve no };`.

'`-p PORTNO`'
'`--port PORTNO`'
Listen the UDP port PORTNO. The accounting port is computed as *PORTNO* + 1.

'`-P DIR`'
'`--pid-file-dir DIR`'
Specifies the alternate path for the pidfile.

'`-S`'
'`--log-stripped-names`'
Log usernames stripped off any prefixes/suffixes.

Config file equivalent: `auth { strip-names yes };`.

'`-s`'
'`--single-process`'
: Run in single process mode. This is for debugging purposes only. We strongly recommend *against* using this option. Use it only when absolutely necessary.

'`-v`'
'`--version`'
: Display program version and compilation options.

'`-x DEBUG_LEVEL`'
'`--debug DEBUG_LEVEL`'
: Set debugging level. *DEBUG_LEVEL* is a comma-separated list of assignments in the forms

```
MODULE
MODULE = LEVEL
```

where *MODULE* is the module name or any non-ambiguous assignment thereof, LEVEL is the debugging level in the range 0-100. Section 9.2 [Debugging], page 89

Config file equivalent:

```
logging {
        category debug {
                level DEBUG_LEVEL;
        };
};
```

'`-y`'
'`--log-auth`'
: Log authentications. With this option enabled, Radius will log any authentication attempt into its log file Chapter 8 [Logging], page 85.

Config file equivalent: `logging { category auth { detail yes; }; };` .

'`-z`'
'`--log-auth-pass`'
: Log passwords along with authentication information. *Do not use* this option. It is *very* insecure, since all users' passwords will be echoed in the logfile. This option is provided only for debugging purposes.

Config file equivalent:

```
logging {
        category auth {
                print-pass yes;
        };
};
```

See Section 4.1 [config file], page 22.

4 Radius Configuration Files

At startup, GNU Radius obtains the information vital for its functioning from a number of configuration files. These are normally found in /usr/local/etc/raddb directory, which is defined at configuration time, although their location can be specified at runtime. In the discussion below we will refer to this directory by '`raddb`'. See Chapter 1 [Naming Conventions], page 5.

Each configuration file is responsible for a certain part of the GNU Radius functionality. The following table lists all configuration files along with a brief description of their purposes.

'`config`' Determines the runtime defaults for `radiusd`, such as the IP address and ports to listen on, the sizes of the request queues, configuration of the SNMP subsystem, fine-tuning of the extension languages, etc.

'`clients`' Lists the shared secret belonging to each NAS. It is crucial for the normal request processing that each NAS have an entry in this file. The requests from NASes that are not listed in '`clients`' will be ignored, as well as those from the NASes that have a wrong value for the shared secret configured in this file.

'`naslist`' Defines the types for the known NASes. Its information is used mainly when performing multiple login checking (see Section 6.9 [Multiple Login Checking], page 74).

'`nastypes`'
Declares the known NAS types. The symbolic type names, declared in this file can be used in '`naslist`'.

'`dictionary`'
Defines the symbolic names for radius attributes and attribute values. Only the names declared in this file may be used in the files '`users`', '`hints`' and '`huntgroups`'.

'`huntgroups`'
Contains special rules that process the incoming requests basing on the NAS IP and port number they come from. These can also be used as a kind of *access control list*.

'`hints`' Defines the matching rules that modify the incoming request depending on the user name and its credentials.

'`users`' Contains the individual users' profiles.

'`realms`' Defines the Radius realms and the servers that are responsible for them.

'`access.deny`'
A list of usernames that should not be allowed access via Radius.

'`sqlserver`'
: Contains the configuration for the SQL system. This includes the type of SQL interface used, the IP and port number of the server and the definition of the SQL requests used by `radiusd`.

'`rewrite`'
: Contains the source code of functions in Rewrite extension language.

'`menus`'
: A subdirectory containing the authentication menus.

The rest of this chapter describes each of these files in detail.

4.1 Run-Time Configuration Options — '`raddb/config`'

At startup `radiusd` obtains its configuration values from three places. The basic configuration is kept in the executable module itself. These values are overridden by those obtained from '`raddb/config`' file. Finally, the options obtained from the command line override the first two sets of options.

When re-reading of the configuration is initiated either by `SIGHUP` signal or by SNMP channel any changes in the config file take precedence over command line arguments, since '`raddb/config`' is the only way to change configuration of the running program.

This chapter discusses the '`raddb/config`' file in detail.

The '`raddb/config`' consists of statements and comments. Statements end with a semicolon. Many statements contain a block of sub-statements which also terminate with a semicolon.

Comments can be written in shell, C, or C++ constructs, i.e. any of the following represent a valid comment:

```
# A shell comment
/* A C-style
 * multi-line comment
 */
// A C++-style comment
```

These are the basic statements:

4.1.1 `option` block

Syntax:

```
option {
        source-ip number ;
        max-requests number ;
        radiusd-user string ;
        exec-program-user string ;
        username-chars string ;
        log-dir string ;
        acct-dir string ;
        resolve bool ;
```

```
            max-processes number ;
            process-idle-timeout number ;
            master-read-timeout number ;
            master-write-timeout number ;
    } ;
```

Usage

The `option` block defines the global options to be used by `radiusd`.

Boolean statements

`resolve` Determines whether radius should resolve the IP addresses for diagnostic output. Specifying `resolve no` speeds up the server and reduces the network traffic.

Numeric statements

`source-ip`
: Sets the source IP address. When this statement is not present, the IP address of the first available network interface on the machine will be used as source.

`max-requests`
: Sets the maximum number of the requests in queue.

`max-processes`
: Sets the maximum number of child processes. The default value is 16. If you plan to raise this value, make sure you have enough file descriptors available, as each child occupies four descriptors for its input/output channels.

`process-idle-timeout`
: Sets the maximum idle time for child processes. A child terminates if it does not receive any requests from the main process within this number of seconds. By default, this parameter is 3600 seconds (one hour).

`master-read-timeout`
`master-write-timeout`
: These two values set the timeout values for the interprocess input/output operations in the main server process. More specifically, `master-read-timeout` sets the maximum number of seconds the main process will wait for the answer from the subprocess, and `master-write-timeout` sets the maximum number of seconds the main process will wait for the subprocess's comunication channel to become ready for input. By default, no timeouts are imposed.

String statements

`radiusd-user`
: Instructs `radiusd` to drop root privileges and to switch to the real user and group IDs of the given user after becoming daemon. Notice the following implications of this statement:

  1. All configuration files must be readable for this user.
  2. Authentication type `System` (see Section 6.5 [System Auth], page 72) requires root privileges, so it cannot be used with `radiusd-user`. Any '`raddb/users`' profiles using this authentication type will be discarded.
  3. Authentication type `PAM` (see Section 6.7 [PAM Auth], page 73) may require root provileges. It is reported to always require root privileges on some systems (notably on Solaris).
  4. `exec-program-user` statement (see below) is ignored when used with `radiusd-user`.

`exec-program-user`
: Sets the privileges for the programs executed as a result of `Exec-Program` and `Exec-Program-Wait`. The real user and group ids will be retrieved from the '`/etc/passwd`' entry for the given user.

`username-chars`
: Determines characters that are valid within a username. The alphanumeric characters are always allowed in a username, it is not necessary to specify them in this statement. By default the following characters are allowed in a username: '`.-_!@#$%^&\/"`'. The `username-chars` statement overrides this default, thus setting:

  ```
  username-chars ":"
  ```

  will restrict the set of allowed characters to the alphanumeric characters and colon. If you wish to expand the default character set, you will have to explicitly specify it in the `username-chars` argument, as shown in the example below:

  ```
  username-chars ".-_!@#$%^&\\/\":"
  ```

  (Notice the use of escape character '\').

`log-dir` Specifies the logging directory.

`acct-dir` Specifies the accounting directory.

4.1.2 `logging` block

Syntax:

```
logging {
```

```
            prefix-hook string ;
            suffix-hook string ;
            category category_spec {
                    channel channel_name ;
                    print-auth bool ;
                    print-pass bool ;
                    print-failed-pass bool ;
                    level debug_level ;
            } ;
            channel channel_name {
                    file string ;
                    syslog facility . priority [tag] ;
                    print-pid bool ;
                    print-category bool ;
                    print-cons bool ;
                    print-level bool ;
                    print-priority bool ;
                    print-tid bool;
                    print-milliseconds bool;
                    prefix-hook string ;
                    suffix-hook string ;
            };
    } ;
```

Usage

The `logging` statement describes the course followed by `radiusd`'s logging information.

The parts of this statement are discussed below.

4.1.2.1 Logging hooks

Most diagnostic messages displayed by `radiusd` describe some events that occured while processig a certain incoming request. By default they contain only a short summary of the event. *Logging hooks* are means of controlling actual amount of information displayed in such messages. They allow you to add to the message being displayed any relevant information from the incoming request that caused the message to appear.

A *hook* is a special Rewrite function that takes three arguments and returns a string. There are two kinds of logging hooks: *prefix* and *suffix*. Return value from the prefix hook function will be displayed before the actual log message, that of the suffix hook function will be displayed after the message.

Furthermore, there may be *global* and *channel-specific hooks*. Global hooks apply to all categories, unless overridden by category-specific hooks. Global prefix hook is enabled by `prefix-hook` statement appearing in the `logging` block. Global suffix hook is enabled by `suffix-hook` statement. Both statements take as their argument the name of corresponding Rewrite function.

For detailed information about writing logging hooks, See Section 10.2.7 [Logging Hook Functions], page 105.

4.1.2.2 `category` statement

Each line of logging information generated by `radiusd` has an associated *category*. The `logging` statement allows each category of output to be controlled independently of the others. The logging category is defined by *category name* and a *severity*. *category name* determines what part of radiusd daemon is allowed to send its logging information to this channel. It can be any of `main`, `auth`, `acct`, `proxy`, `snmp`. *priority* determines the minimum priority of the messages displayed by this channel. The priorities in ascending order are: `debug`, `info`, `notice`, `warn`, `err`, `crit`, `alert`, `emerg`.

The full category specification, denoted by the `category_spec` in the above section, can take any of the following three forms:

category_name
: Print the messages of given category.

priority
: Print messages of all categories, abridged by given priority. If the priority is prefixed with '`=`', only messages with given priority will be displayed. If it is prefixed with '`!`', the messages with priority other than the specified will be displayed. Otherwise, the messages with priorities equal to or greater than the specified will be displayed.

category_name . priority
: Print the messages of given category, abridged by given priority. The priority may be prefixed with either '`=`' or '`!`' as described above. The dot ('`.`') separates the priority from the category name, it may be surrounded by any amount of whitespace.

Additional category options valid for `auth` category are:

`print-auth`
: Log individual authentications.

`print-pass`
: Include passwords for successful authentications. It is *very* insecure, since all users' passwords will be echoed in the logfile. This option is provided only for debugging purposes.

`print-failed-pass`
: Include passwords for failed authentications.

4.1.2.3 `channel` statement

Channels represent methods for recording logging information. Each channel has a unique name, and any categories which specify that name in a `channel` statement will use that channel.

`radiusd` can write logging information to files or send it to syslog. The `file` statement sends the channel's output to the named file (see Chapter 1 [Naming Conventions], page 5). The `syslog` statement sends the channel's output to syslog with the specified facility and severity. Its optional last argument allows to alter default syslog tag.

Channel options modify the data flowing through the channel:

`print-pid`
: Add the process ID of the process generating the logging information.

`print-cons`
: Also send the logging information to the system console.

`print-category`
: Add the category name to the logging information.

`print-priority`
`print-level`
: Add the priority name to the logging information.

`print-milliseconds`
: Print timestamp with milliseconds.

`prefix-hook`
: Declares the name of Rewrite function used as logging prefix hook for that channel (see Section 4.1.2.1 [hooks], page 25). This overrides any global prefix hook.

`suffix-hook`
: Declares the name of Rewrite function used as logging suffix hook for that channel (see Section 4.1.2.1 [hooks], page 25). This overrides any global suffix hook.

4.1.2.4 Example of the `logging` statement

```
logging {
        channel default {
                file "radius.log";
                print-category yes;
                print-priority yes;
        };
        channel info {
                file "radius.info";
                print-pid yes;
                print-cons yes;
                print-priority yes;
        };
        channel notice {
                syslog auth.notice;
        };
```

```
        category auth {
                print-auth yes;
                print-failed-pass yes;
        };
        category notice {
                channel notice;
        };
        category info {
                channel info;
        };
        category debug {
                channel info;
                level radiusd=1,files;
        };

        category *.!debug {
                channel default;
        };
};
```

4.1.3 `auth` statement

Syntax:

```
auth {
        listen ( addr-list | no );
        forward addr-list;
        port number ;
        max-requests number ;
        time-to-live number ;
        request-cleanup-delay number ;
        detail bool ;
        strip-names bool ;
        checkrad-assume-logged bool ;
        password-expire-warning number ;
        compare-atribute-flag character ;
        trace-rules bool ;
        reject-malformed-names bool ;
} ;
```

Usage:

The `auth` statement configures the parameters of the authentication service.

`listen` statement

This statement determines on which addresses radiusd will listen for incoming authentication requests. Its argument is a comma-separated list of items in the form *ip*:*port-number*. *ip* can be either an IP address in familiar "dotted-quad" notation or a hostname. :*port-number* part may be omitted, in which case the default authentication port is assumed.

If the `listen` statement is omitted, radiusd will accept incoming requests from any interface on the machine.

The special value `no` disables listening for authentication requests.

The following example configures radius to listen for the incoming requests on the default authentication port on the address 10.10.10.1 and on port 1645 on address 10.10.11.2.

```
listen 10.10.10.1, 10.10.11.2:1645;
```

`forward` statement

This statement enables *forwarding* of the requests to the given set of servers. Forwarding is an experimental feature of GNU Radius, it differs from proxying in that the requests are sent to the remote server (or servers) *and* processed locally. The remote server is not expected to reply.

This mode is intended primarily for debugging purposes. It could also be useful in some very complex and unusual configurations.

Numeric statements

`port` Sets the number of which UDP port to listen on for the authentication requests.

`max-requests`
: Sets the maximum number of authentication requests in the queue. Any surplus requests will be discarded.

`time-to-live`
: Sets the request time-to-live in seconds. The time-to-live is the time to wait for the completion of the request. If the request job isn't completed within this interval of time it is cleared, the corresponding child process killed and the request removed from the queue.

`request-cleanup-delay`
: Sets the request cleanup delay in seconds, i.e. determines how long will the completed authentication request reside in the queue.

`password-expire-warning`
: Sets the time interval for password expiration warning. If user's password expires within given number of seconds, radiusd will send a warning along with authentication-acknowledge response. Default is 0.

Boolean statements

`detail` When set to true, `radiusd` will produce the detailed log of each received packet in the file '`radacct/`*`nasname`*`/detail.auth`'. The format of such log files is identical to the format of detailed

accounting files (see Section 7.2 [Detailed Request Accounting], page 81).

`strip-names`
: Determines whether `radiusd` should strip any prefixes/suffixes off the username before logging.

`checkrad-assume-logged`
: See Section 4.1.11 [mlc], page 39, for the description of this setting. It is accepted in `auth` for compatibility with previous versions of GNU Radius.

`trace-rules`
: Enables tracing of the configuration rules that were matched during processing of each received authentication request. See Section 9.1 [Rule Tracing], page 87, for detailed information about this mode.

`reject-malformed-names`
: Enables sending access-reject replies for the access-accept requests that contain an invalid value in `User-Name` attribute. By default such requests are discarded without answering. See the description of `username-chars` (see Section 4.1.1 [Option statement], page 22).

Character statement

`compare-attribute-flag`
: The argument to this statement is a character from '`1`' through '`9`'. This statement modifies the request comparison method for authentication requests. See Section 5.1 [Extended Comparison], page 67, for a detailed description of its usage.

4.1.4 `acct` statement

Syntax:

```
acct {
        listen ( addr-list | no );
        forward addr-list ;
        port number ;
        detail bool;
        system bool;
        max-requests number ;
        time-to-live number ;
        request-cleanup-delay number ;
        compare-atribute-flag character ;
        trace-rules bool ;
} ;
```

Usage:

The `acct` statement configures the parameters of the accounting service.

`listen` statement

This statement determines on which addresses radiusd will listen for incoming accounting requests. Its argument is a comma-separated list of items in the form *ip*:*port-number*. *ip* can be either an IP address in familiar "dotted-quad" notation or a hostname. :*port-number* part may be omitted, in which case the default accounting port is assumed.

If the `listen` statement is omitted, radiusd will accept incoming requests from any interface on the machine.

The special value `no` disables listening for accounting requests.

The following example configures radius to listen for the incoming requests on the default accounting port on the address 10.10.10.1 and on port 1646 on address 10.10.11.2.

```
listen 10.10.10.1, 10.10.11.2:1646;
```

`forward` statement

This statement enables *forwarding* of the requests to the given set of servers. Forwarding is an experimental feature of GNU Radius, it differs from proxying in that the requests are sent to the remote server (or servers) *and* processed locally. The remote server is not expected to reply.

This mode is intended primarily for debugging purposes. It could also be useful in some very complex and unusual configurations.

Numeric statements

`port` Sets the number of which port to listen for the authentication requests.

`max-requests`
: Sets the maximum number of accounting requests in the queue. Any surplus requests will be discarded.

`time-to-live`
: Sets the request time-to-live in seconds. The time-to-live is the time to wait for the completion of the request. If the request job isn't completed within this interval of time it is cleared, the corresponding child process killed and the request removed from the queue.

`request-cleanup-delay`
: Sets the request cleanup delay in seconds, i.e. determines how long will the completed account request reside in the queue.

Boolean statements

`detail` When set to `no`, disables detailed accounting (see Section 7.2 [Detailed Request Accounting], page 81).

`system` When set to `no`, disables system accounting (see Section 7.1 [System Accounting], page 81). Notice, that this will disable simultaneous use checking as well, unless you supply an alternative MLC method (currently SQL, See Section 6.9 [Multiple Login Checking], page 74, for the detailed discussion of this).

`trace-rules`
Enables tracing of the configuration rules that were matched during processing of each received accounting request. See Section 9.1 [Rule Tracing], page 87, for detailed information about this mode.

Character statement

`compare-attribute-flag`
The argument to this statement is a character from '`1`' through '`9`'. This statement modifies the request comparison method for authentication requests. See Section 5.1 [Extended Comparison], page 67, for a detailed description of its usage.

4.1.5 `usedbm` statement

Syntax:

```
usedbm ( yes | no ) ;
```

Usage

The `usedbm` statement determines whether the DBM support should be enabled.

`no` Do not use DBM support at all.

`yes` Use only the DBM database and ignore '`raddb/users`'.

4.1.6 `snmp` statement

Syntax:

```
snmp {
        port portno ;
        listen ( addr-list | no );
        max-requests number ;
        time-to-live number ;
        request-cleanup-delay number ;
        ident string ;
```

```
            community name ( rw | ro ) ;
            network name network [ network ... ] ;
            acl {
                    allow network_name community_name ;
                    deny network_name ;
            } ;
            storage {
                    file filename ;
                    perms number ;
                    max-nas-count number ;
                    max-port-count number ;
            } ;
    };
```

Usage

The `snmp` statement configures the SNMP service.

`listen` statement

The `listen` statement determines on which addresses radiusd will listen for incoming SNMP requests. The argument is a comma-separated list of items in the form *ip*:*port-number*. The *ip* can be either an IP address in familiar "dotted-quad" notation or a hostname. The :*port-number* part may be omitted, in which case the default SNMP port (161) is used.

If the `listen` statement is omitted, radiusd will accept incoming requests from any interface on the machine.

The special value `no` disables listening for SNMP requests.

The following example configures radius to listen for the incoming SNMP requests on the default SNMP port on the address 10.10.10.1 and on port 4500 on address 10.10.11.2.

```
listen 10.10.10.1, 10.10.11.2:4500;
```

Numeric statements

`port` Sets the number of which port to listen for the SNMP requests.

`max-requests`
: Sets the maximum number of SNMP requests in the queue. Any surplus requests will be discarded.

`time-to-live`
: Sets the request time-to-live in seconds. The time-to-live is the time to wait for the completion of the request. If the request job isn't completed within this interval of time it is cleared, the corresponding child process killed and the request removed from the queue.

`request-cleanup-delay`
: Sets the request cleanup delay in seconds, i.e. determines how long will the completed SNMP request reside in the queue.

String statements

`ident` Sets the SNMP server identification string.

Community and network definitions

`community` *`name`* `( rw | ro )`
: Defines the community *name* as read-write (`rw`) or read-only (`ro`).

`network` *`name network`* `[` *`network`* `... ]`
: Groups several networks or hosts under one logical network name.

Access-Control List definitions

`allow` *`network_name community_name`*
: allow hosts from the group *network_name* access to community *community_name*.

`deny` *`NETWORK_NAME`*
: Deny access to SNMP service from any host in the group *network_name*.

Storage control

GNU Radius stores the SNMP monitoring data in an area of shared memory mapped to an external file. This allows all subprocesses to share this information and to accumulate the statistics across invocations of the daemon.

The `storage` statement controls the usage of the storage for the SNMP data.

`file` Sets the file name for the SNMP storage file. Unless the filename begins with a '`/`' it is taken as relative to the current logging directory.

`perms` Sets the access permissions for the storage file. *Notice*, that this statement does not interpret its argument as octal by default, so be sure to prefix it with '`0`' to use an octal value.

`max-nas-count`
: Sets maximum number of NASes the storage file is able to handle. Default is 512. Raise this number if you see the following message in your log file:

```
reached SNMP storage limit for the number of
monitored NASes: increase max-nas-count
```

`max-port-count`
: Sets maximum number of ports the storage file is able to handle. Default is 1024. Raise this number if you see the following message in your log file:

```
reached SNMP storage limit for the number of
monitored ports: increase max-port-count
```

4.1.7 `rewrite` statement.

(This message will disappear, once this node revised.)

Syntax:

```
rewrite {
        stack-size number ;
        load-path string ;
        load string ;
};
```

Numeric statements

`stack-size`
: Configures runtime stack size for Rewrite. The *number* is the size of stack in words. The default value is 4096.

String statements

`load-path`
: Add specified pathname to the list of directories searched for rewrite files.

`load`
: Loads the specified source file on startup. Unless *string* is an absolute pathname, it will be searched in directories set up by `load-path` statement.

Loading

The default load path is `'RADDB':'DATADIR'/rewrite`.

4.1.8 `guile` statement

(This message will disappear, once this node revised.)

The `guile` statement allows to configure server interface with Guile.

Syntax

```
guile {
        debug bool ;
        load-path string ;
```

```
            load string ;
            load-module string [ string ... ] ;
            eval expression [ expression ... ] ;
            gc-interval number ;
            outfile string ;
    };
```

Usage

Boolean statements

`debug` When set to yes, enables debugging evaluator and backtraces on Guile scripts.

Numeric statements

`gc-interval`
: Configures the forced garbage collections. By default the invocation of the garbage collector is run by the internal Guile mechanism. However, you may force Radius to trigger the garbage collection at fixed time intervals. The `gc-interval` statement sets such interval in seconds.

  For more information about Guile memory management system in general and garbage collections in particular, see Section "Memory Management and Garbage Collection" in *The Guile Reference Manual*.

String statements

`eval` Evaluates its argument as `Scheme` expression.

`load-path`
: Adds specified pathname to `%load-path` variable.

`load` Loads the specified source file on startup.

`load-module`
: Loads the specified Scheme module on startup. This statement takes an arbitrary number of arguments. The first argument specifies the name of the module to load, the rest of arguments is passed to the *module initialization* funtion. Module initialization function is a function named '*`module`*`-init`', where *module* is the module name. Arguments are converted using usual `Guile` rules, except that the ones starting with a dash ('`-`') are converted to keyword arguments.

`outfile` Redirects the standard output and standard error streams of the `Guile` functions to the given file. Unless the filename starts with '`/`', it is taken relative to the current logging directory.

See Section 10.3 [Guile], page 115, for a detailed description of Guile extensions interface.

4.1.9 `message` statement

The `message` statement allows to set up the messages that are returned to the user with authentication-response packets.

Syntax

```
message {
        account-closed string ;
        password-expired string ;
        password-expire-warning string ;
        access-denied string ;
        realm-quota string ;
        multiple-login string ;
        second-login string ;
        timespan-violation string ;
};
```

All variables in `message` block take a string argument. In *string* you can use the usual C backslash notation to represent non-printable characters. The use of %C{} and %R{} sequences is also allowed (see Section 4.14 [Macro Substitution], page 64).

String statements

`account-closed`
: This message will be returned to the user whose account is administratively closed.

`password-expired`
: This message will be returned to the user whose password has expired.

`password-expire-warning`
: This is a warning message that will be returned along with an authentication-acknowledge packet for the user whose password will expire in less than *n* seconds. The value of *n* is set by `password-expire-warning` variable in `auth` block. See Section 4.1.3 [auth], page 28. In this string, you can use the %R{Password-Expire-Days} substitution, to represent the actual number of *days* left to the expiration date. The default is

```
Password Will Expire in %R{Password-Expire-Days} Days\r\n
```

`access-denied`
: This message is returned to the user who supplies an incorrect password or a not-existent user-name as his authentication credentials.

`realm-quota`
: This message is returned when the user is trying to log in using a realm, and number of users that are currently logged in from this realm reaches maximum value. For a description of realms, see Section 2.4.2.2 [Realms], page 14.

`multiple-login`
: This message is returned to the user, who has logged in more than allowed number of times. For description of how to set the maximum number of concurrent logins, see Section 13.3.25 [Simultaneous-Use], page 192.

`second-login`
: This is a special case of `multiple-login`, which is used when the user's login limit is 1.

`timespan-violation`
: This message is returned to the user who is trying to login outside of allowed time interval. For description of how to limit user's login time, see Section 13.3.14 [Login-Time], page 187.

4.1.10 `filters` statement

The `filters` statement configures user-defined external filters. See Section 10.1 [Filters], page 95, for the detailed discussion of external filters.

Syntax

```
filters {
        filter ident {
                exec-path path ;
                error-log filename ;
                common bool [max-wait];
                auth {
                        input-format fmt ;
                        wait-reply bool ;
                };
                acct {
                        input-format fmt ;
                        wait-reply bool ;
                };
        };
        ...
};
```

Each `filter` directive defines a new filter. The *ident* argument declares the name of the filter. This string must be used in `Exec-Program-Wait` or `Acct-Ext-Program` attributes to trigger invocation of this filter (see Section 13.3.7 [Exec-Program-Wait], page 181).

Usage

exec-path *path*
: Absolute path to the filter program.

error-log *filename*
: Redirect error output from the filter program to *filename*. If the *filename* does not start with a slash, it is taken relative to the current logging directory (see Section 4.1.1 [option], page 22).

auth
acct
: These compound statements define authentication and accounting parts of this filter. Any one of them may be missing. The two statements allowed within `auth` and `acct` blocks are:

  input-format *fmt*
  : Format of the input line for this filter. Usually this string uses %C`{}` notations (see Section 4.14 [Macro Substitution], page 64).

    You can also use the return value from a `rewrite` function as input line to the filter. To do so, declare:

    ```
    input-format "=my_func()";
    ```

    where *my_func* is the name of the rewrite function to invoke. The function must return string value.

  wait-reply *bool*
  : If the filter prints a single line of output for each input line, set this to `yes`. Otherwise, if the filter produces no output, use `wait-reply no`.

4.1.11 `mlc` statement

Syntax

```
mlc {
        method (system|sql);
        checkrad-assume-logged bool;
};
```

Usage

`Mlc` statement configures multiple login checking subsystem (see Section 6.9 [Multiple Login Checking], page 74).

method
: Sets the method of retrieving information about the currently open sessions. Currently two methods are implemented. Setting `method` to `system` will use system accounting database (see Section 7.1 [System Accounting], page 81). This is the default method. Setting it to `sql` will use SQL database.

checkrad-assume-logged
: `radiusd` consults the value of this variable when the NAS does not responds to checkrad queries (see Section 6.9 [Multiple Login Checking], page 74). If this variable is set to `yes`, the daemon will proceed as if the NAS returned "yes", i.e. it will assume the user is logged in. Otherwise `radiusd` assumes the user *is not* logged in.

4.2 Dictionary of Attributes — 'raddb/dictionary'

The dictionary file '`raddb/dictionary`' defines the symbolic names for radius attributes and their values (see Section 2.1 [Attributes], page 7). The file consists of a series of statements, each statement occupies one line.

In the detailed discussion below we use the following meta-syntactic characters:

number
: Denotes a decimal, octal or hexagesimal number. Usual C conventions are honored, i.e. if *number* starts with '`0x`' or '`0X`' it is read as a hex number, if it starts with '`0`' it is read as an octal number, otherwise it is read as a decimal one.

type
: Denotes an attribute type. These are valid attribute types:

  `string`
  : A string type.

  `integer`
  : An integer type.

  `ipaddr`
  : IP address in a dotted-quad form.

  `date`
  : A date in the format: `"MON DD CCYY"`, where MON is the usual three-character abbreviation, DD is day of month (1-31), CCYY is the year, including the century.

4.2.1 Comments

Comments are introduced by a pound sign ('`#`'). Everything starting from the first occurrence of '`#`' up to the end of line is ignored.

4.2.2 $INCLUDE Statement

Syntax

```
$INCLUDE 'filename'
```

Usage

The `$INCLUDE` statement causes the contents of the file '`filename`' to be read in and processed. The file is looked up in the Radius database directory, unless its name starts with a slash.

4.2.3 VENDOR Statement

Syntax

```
VENDOR  vendor-name vendor-id
```

Usage

A `VENDOR` statement defines the symbolic name *vendor-name* for vendor identifier *vendor-id*. This name can subsequently be used in `ATTRIBUTE` statements to define Vendor-Specific attribute translations. See Section 13.1.26 [Vendor-Specific], page 175.

Example

```
VENDOR  Livingston  307
```

4.2.4 ATTRIBUTE statement

Syntax

```
ATTRIBUTE  name  number  type [vendor] [flags]
```

Usage

The `ATTRIBUTE` statement defines the internal representation of an attribute: its symbolic name, data type and syntactical usage. Its parts have the following meaning:

name The attribute name.

number The attribute ID (number).

type The attribute type.

vendor Vendor name for vendor-specific attributes. For usual attributes this field is empty or contains a dash ('`-`'). The latter usage is for compatibility with previos version of GNU Radius

flags Flags, defining attribute properties (see Section 2.1 [Attributes], page 7).

The *attribute property flags* consist of a sequence of letters, whose meaning is determined by the following rules:[1]

1. The attribute usage is described by three pairs of symbols, enclosed in square brackets. Each pair describes how the attribute can be used in each of three configuration files. The first pair corresponds to '`raddb/users`', the second one corresponds to '`raddb/hints`', and the third one corresponds to '`raddb/huntgroups`'. Within each pair,

[1] The *flags* are optional for compatibility with previous versions of GNU Radius. If they are omitted, the default is '`[LRLRLR]+`'

the letter 'L' in first position means that the attribute is allowed in LHS of a rule. The letter 'R' in second position means that the attribute is allowed in RHS of a rule. The absence of any of these letters is indicated by dash ('-'). Thus, the following usage specification:

```
[L--RLR]
```

means that the attribute may be used in LHS of a rule in 'raddb/users', in RHS of a rule in 'raddb/hints', and in both sides of a rule in 'raddb/huntgroups'.

2. The attribute additivity is described by one of the following letters:

   = Additivity = Replace

   + Additivity = Append

   N Additivity = None

3. The presence of letter 'P' in property flags raises the propagation bit.
4. Letter 'l' (lower-case ell) enables logging the given attribute in detail file (see Section 7.2 [Detailed Request Accounting], page 81). This is meaningful only for internal attributes, i.e. the ones whose decimal value is greater than 255 (see Section 13.3 [Radius Internal Attributes], page 178). By default such attributes do not appear in detailed logs. The flag 'l' reverts this behavior.
5. Letter 'E' marks attributes encrypted as described in RFC 2138. Currently these are User-Password and CHAP-Password.
6. Letter 'T' marks attribute encrypted according to RFC 2868.
7. The characters from '1' to '9' denote nine user-defined flags (see Section 5.1 [Extended Comparison], page 67).

Example

```
ATTRIBUTE  Service-Type  6 integer - [LR-RLR]=P
```

This statement declares that the attribute number 6 will be referred to by the symbolic name 'Service-Type'. The attribute is of integer data type and it may be used in any part of matching rules, except in LHS of a 'raddb/hints' rule. The additivity of Service-Type is set to 'Replace'. The attribute will be propagated through the proxy chain.

4.2.5 Blocks of Vendor-Specific Attributes

Syntax

```
BEGIN VENDOR vendor-name [vendor-id]
...
END
```

Usage

The BEGIN keyword marks start of the block of definitions of vendor-specific attributes. The block is terminated by END keyword, optionally followed by

an arbitrary number of words, which are regarded as a comment. The block may contain any valid dictionary declarations, except other blocks: nesting of declaration blocks is not allowed.

If *vendor-id* is absent, the value of vendor ID is looked up in the internal table of vendors; therefore, it must be defined before `BEGIN` statement (see Section 4.2.3 [VENDOR], page 41).

`BEGIN---END` block alters the handling of `ATTRIBUTE` statements within it. If `ATTRIBUTE` statement does not contain an explicit vendor-id specification, the value of *vendor-id* is used instead.

For compatibility with FreeRadius an alternative syntax is also supported:

```
BEGIN-VENDOR vendor-name
...
END-VENDOR vendor-name
```

Such compatibility blocks must appear only ater the declaration of *vendor-name* (see Section 4.2.3 [VENDOR], page 41).

Example

The following is the usual way of definig vendor-specific attributes:

```
VENDOR          Livingston      307

ATTRIBUTE       LE-Terminate-Detail     2       string  Livingston
ATTRIBUTE       LE-Advice-of-Charge     3       string  Livingston
```

The following two examples show the alternative ways:

```
VENDOR Livingston 307
BEGIN VENDOR Livingston
ATTRIBUTE       LE-Terminate-Detail     2       string
ATTRIBUTE       LE-Advice-of-Charge     3       string
END
```

```
BEGIN VENDOR Livingston 307
ATTRIBUTE       LE-Terminate-Detail     2       string
ATTRIBUTE       LE-Advice-of-Charge     3       string
END
```

These three examples are completely equivalent to each other.

4.2.6 ALIAS statement

Syntax

```
ALIAS name  alt-name
```

Usage

The `ALIAS` statement defines an altenative name *alt-name* for attribute *name*. The latter should already be defined, otherwise an error occurs.

Example

```
ALIAS User-Password Password
```

4.2.7 PROPERTY statement

Syntax

```
PROPERTY  name  flags
PROPERTY  name  +flags [-flags ...]
```

Usage

The `PROPERTY` statement redefines property flags for attribute *name*. The attribute must be defined, otherwise an error occurs. The `PROPERTY` statement has two forms. In first form, it takes a single argument, representing new property flags for the attribute. In its second form it takes any number of arguments, each of them preceeded by '`+`' sign, inidicating addition of properties, or by '`-`' sign, indicating removal of these.

See Section 4.2.4 [ATTRIBUTE], page 41, for the discussion of attribute property flags.

Example

The following example defines that the attribute `User-Password` may be used only on left-hand side of a '`raddb/users`' entry, and that it is transmitted in encrypted form.

```
PROPERTY  User-Password [L-----]E
```

Next example illustrates adding and removing attribute properties:

```
PROPERTY  My-Attrib     +P -=
```

it adds propagation bit ('`P`') and removes '`replace`' additivity from `My-Attrib` attribute.

4.2.8 VALUE Statement

Syntax

```
VALUE   Attribute-Translation       Value-Translation       number
```

Usage

The `VALUE` statement assigns a translation string to a given value of an integer attribute. `Attribute-Translation` specifies the attribute and the `Value-Translation` specifies the name assigned to the value *number* of this attribute.

Example

The following assigns the translation string '`Login-User`' to the value 1 of the attribute '`Service-Type`'.

```
VALUE  Service-Type  Login-User  1
```

4.3 Clients List — 'raddb/clients'

The 'raddb/clients' lists NASes which are allowed to make authentication requests. As usual, the '#' character introduces a comment. Each record in the file consists of two fields, separated by whitespace. The fields are:

NAS name
: Specifies a hostname or IP address of the NAS.

Key
: Lists the encryption key shared between the server and this NAS.

If the set of NASes share the same encryption key, there are two ways to list it in 'raddb/clients'. First, if these NASes lie in a single network, you can specify this network address in NAS name field, e.g.:

```
10.10.10.0/27   seCRet
```

Notice also that specifying full netmask after the '/' character is also allowed, so that the above example could also be written as follows:

```
10.10.10.0/255.255.255.224   seCRet
```

Otherwise, the keyword DEFAULT may be used as NAS name. This notation will match any IP address, so it should be used with caution.

4.3.1 Example of 'clients' file

```
# This is a list of clients which are allowed to make authentication
# requests.
# Each record consists of two fields:
#       i.  Valid hostname.
#       ii. The shared encryption key for this hostname.
#
#Client Name            Key
#----------------       --------------------
myhost.dom.ain          guessme
merlin                  emrys
11.10.10.10             secRet
```

4.4 NAS List — 'raddb/naslist'

The 'raddb/naslist' file contains a list of NASes known to the Radius server. Each record in the file consist of the following four fields, the first two being mandatory, the last two being optional:

NAS name
: Specifies either a hostname or IP address for a single NAS or a CIDR net block address for a set of NASes. The word 'DEFAULT' may be used in this field to match any NAS.[2]

Short Name
: This field defines a short name under which this NAS will be listed in logfiles. The short name is also used as a name of the subdirectory where the detailed logs are stored.

[2] Logins from DEFAULT NASes are not reflected in SNMP variables.

Type
: Specifies the type of this NAS. Using this value `radiusd` determines the way to query NAS about the presence of a given user on it (see Section 6.9 [Multiple Login Checking], page 74). The two special types: '`true`' and '`false`', can be used to disable NAS querying. When the type field contains '`true`', `radiusd` assumes the user is logged in to the NAS, when it contains '`false`', `radiusd` assumes the user *is not* logged in. Otherwise, the type is used as a link to '`nastypes`' entry (see Section 4.5 [nastypes file], page 47).

  If this field is not present '`true`' is assumed.

Arguments
: Additional arguments describing the NAS. Multiple arguments must be separated by commas. No intervening whitespace is allowed in this field.

There are two groups of nas arguments: *nas-specific* arguments and *nas-querying* arguments. *Nas-specific* arguments are used to modify a behavior of `radiusd` when sending or receiving the information to or from a particular NAS.

Nas-querying arguments control the way `radiusd` queries a NAS for confirmation of a user's session (see Section 6.9 [Multiple Login Checking], page 74). These arguments override the ones specified in '`nastypes`' and can thus be used to override the default values.

The *nas-specific* arguments currently implemented are:

broken_pass
: This is a boolean argument that controls the encryption of user passwords, longer than 16 octets. By default, `radiusd` uses method specified by RFC 2865. However some NASes, most notably MAX Ascend series, implement a broken method of encoding long passwords. This flag instructs `radiusd` to use broken method of password encryption for the given NAS.

compare-auth-flag=*flag*
: Instructs radius to use attributes marked with a given user-defined flag when comparing authentication requests. It overrides `compare-attribute-flag` (see Section 4.1.3 [auth], page 28) for this particular NAS. See Section 5.1 [Extended Comparison], page 67, for a detailed description of its usage.

compare-acct-flag=*flag*
: Instructs radius to use attributes marked with a given user-defined flag when comparing accounting requests. It overrides `compare-attribute-flag` (see Section 4.1.4 [acct], page 30) for this particular NAS. See Section 5.1 [Extended Comparison], page 67, for a detailed description of its usage.

See Section 2.4.1 [Checking Duplicates], page 12, for general description of request comparison methods.

For the list of nas-querying arguments, See Section 4.5 [Full list of allowed arguments], page 47.

4.4.1 Example of 'naslist' file

```
# raddb/naslist: contains a list of Network Access Servers
#
# Each record consists of following fields:
#
#       i.      A valid hostname or IP address for the client.
#       ii.     The short name to use in the logfiles for this NAS.
#       iii.    Type of device. Valid values are 'true', 'false' and
#               those defined in raddb/nastypes file.

# NAS Name              Short Name      Type
#----------------       ----------      ----
myhost.dom.ain          myhost          unix
merlin                  merlin          max
11.10.10.10             arthur          livingston
```

4.5 NAS Types — 'raddb/nastypes'

The 'raddb/nastypes' file describes the ways to query NASes about active user sessions.

4.5.1 Syntax of 'raddb/nastypes'

(This message will disappear, once this node revised.)

Syntax

Each record consists of three fields separated by any amount of whitespace. The fields are:

Type — Type of the NAS which is described in this record.

Method — Method to use to query a NAS of given type.

Arguments

Arguments to pass to this method. Each argument is a pair *arg*=*value*, where *arg* is its name and *value* is a value assigned to it. The list of predefined argument names follows. Note, that no intervening whitespace is allowed in this field.

Methods

Version 1.6 of GNU Radius supports following querying methods: finger, snmp, external and guile. .

Arguments

In the discussion below *n* means numeric and *s* string value.

The following arguments are predefined:

Common for all methods

function=*s*
: Specifies the check function to use with this method (see Section 10.2.5 [Login Verification Functions], page 103). This argument must be present. For description of how this function is applied, see Section 6.9 [Multiple Login Checking], page 74.

port=*n*
: Use port number *n* instead of the default for the given method.

Method snmp

password=*s*
: Use community *s* instead of the default. This argument must be present.

retries=*n*
: Retry *n* times before giving up.

timeout=*n*
: Timeout *n* seconds on each retry.

Method finger

timeout=*n*
: Give up if the NAS does not respond within *n* seconds.

notcp
tcp=0
: Disable the use of T/TCP for hosts with a broken TCP implementation.

arg=*subst*
: Send *subst* to finger, instead of username. *subst* must be one of *macro variables*, described below.

Macro variables

The following macro-variables are recognized and substituted when encountered in the *value* pair of an argument:

‘`%u`’
: Expands to username.

‘`%s`’
: Expands to session id.

‘`%d`’
: Expands to session id converted to decimal representation.

‘`%p`’
: Expands to port number.

‘`%P`’
: Expands to port number + 1.

4.5.2 Example of nastypes file.

Note, that in the following example the long lines are broken into several lines for readability.

```
# Type      Method          Args
# ----      ------          ----
unix        finger       function=check_unix
max-f       finger       function=check_max_finger
max         snmp         oid=.1.3.6.1.4.1.529.12.3.1.4.%d,
                         function=check_snmp_u
as5300-f    finger       function=check_as5300_finger
as5300      snmp         oid=.1.3.6.1.4.1.9.9.150.1.1.3.1.2.%d,
                         function=check_snmp_u
livingston snmp          oid=.1.3.6.1.4.1.307.3.2.1.1.1.5.%P,
                         function=check_snmp_s
```

4.5.3 Standard NAS types

The '`nastypes`' shipped with version 1.6 of GNU Radius defines following NAS types:

unix — UNIX boxes running Finger
: This type suits for UNIX boxes running finger service able to return information about dial-up users active on them. To enable finger checking of a unix host add following to your '`naslist`' file:

```
#Hostname       Shortname   Type
#--------       ---------   ----
nas.name        T           unix
```

max-f — MAX Ascend with Finger
: Use this type if you have MAX Ascend terminal server that answers finger queries. The '`naslist`' entry for such NAS will look like:

```
#Hostname       Shortname   Type  Flags
#--------       ---------   ----  -----
nas.name        T           max-f broken_pass
```

Note the use of `broken_pass` flag. It is needed for most MAX Ascend servers (see Section 4.4 [naslist file], page 45).

max — MAX Ascend, answering SNMP
: Use this type if you have MAX Ascend terminal server that answers SNMP queries. The '`naslist`' entry for such NAS will look like:

```
#Hostname       Shortname   Type  Flags
#--------       ---------   ----  -----
nas.name        T           max-f broken_pass,community=comm
```

Replace *comm* with your actual SNMP community name.

as5300-f — Cisco AS5300 running finger
as5300 — Cisco AS5300 answering SNMP
livingston — Livingston Portmaster
Type `livingston` queries portmaster using SNMP.

4.6 Request Processing Hints — 'raddb/hints'

The '`raddb/hints`' file is used to modify the contents of the incoming request depending on the username. For a detailed description of this, See Section 2.4.3 [Hints], page 14.

The file contains data in *Matching Rule* format (see Section 2.3 [Matching Rule], page 11).

Notice, that versions of GNU Radius up to 1.0 allowed to use only a subset of attributes in the check list of a '`hints`' entry, namely:

- `Suffix`
- `Prefix`
- `Group`
- `User-ID`

This requirement has been removed in version 1.0.

4.6.1 Example of 'hints' file

```
## If the username starts with 'U', append the UUCP hint
DEFAULT         Prefix = "U", Strip-User-Name = No
                Hint = "UUCP"
## If the username ends with '.slip', append the SLIP service data
## and remove the suffix from the user name.
DEFAULT         Suffix = ".slip",
                   Strip-User-Name = Yes
                Hint = "SLIP",
                   Service-Type = Framed-User,
                   Framed-Protocol = SLIP
```

4.7 Huntgroups — 'raddb/huntgroups'

The '`raddb/huntgroups`' contains the definitions of the huntgroups. For a detailed description of huntgroup concept, See Section 2.4.4 [Huntgroups], page 15.

The file contains data in *Matching Rule* format (see Section 2.3 [Matching Rule], page 11).

4.7.1 Example of 'huntgroups' file.

```
## This defines the packet rewriting function for the server 11.10.10.11
DEFAULT NAS-IP-Address = 11.10.10.11, Rewrite-Function = "max_fixup"
        NULL
```

4.8 List of Proxy Realms — 'raddb/realms'

The 'raddb/realms' file lists remote Radius servers that are allowed to communicate with the local Radius server (see Section 2.4.2 [Proxying], page 13).

Each record consists of up to three fields, separated by whitespace. Two of them are mandatory. The fields are:

Realm name

Specifies the name of the realm being defined, i.e. part of the login name after the '@' symbol. There are three special forms of this field.

The name 'NOREALM' defines the empty realm, i.e. lines marked with this name will match user names without any realm suffix.

The name 'DEFAULT' defines the default realm (see Section 2.4.2.2 [Realms], page 14). The lines with this realm name will match any user name, not matched by any other line in 'raddb/realms'.

Remote server list

A comma-separated list of remote servers to which the requests for this realm should be forwarded. Each item in the list is:

```
servername[:auth-port[:acct-port]]
```

Optional *auth-port* and *acct-port* are the authentication and accounting port numbers. If *acct-port* is omitted, it is computed as *auth-port* + 1. If *auth-port* is omitted, the default authentication port number is used.

The servers from this list are tried in turn until any of them replies or the list is exhausted, whichever occurs first. The timeout value and number of retries for each server are set via `timeout` and `retry` flags (see below).

There may be cases where you would wish a particular realm to be served by the server itself. It is tempting to write

```
# Wrong!
realm.name      localhost
```

however, this will not work. The special form of the server list is provided for this case. It is the word 'LOCAL'. The correct configuration line for the above case will thus be:

```
# Use this to declare a locally handled realm
realm.nam       LOCAL
```

Flags (optional)

The flags meaningful in 'raddb/realms' are

ignorecase Boolean value. When set, enables case-insensitive comparison of realm names. For example, if a realm were defined as

```
myrealm.net     remote.server.net:1812  ignorecase
```

then user name 'user@MyREAlm.NeT' will match this definition.

strip Boolean value. Controls whether the realm name should be stripped off the username before forwarding the request to the remote server. Setting `strip` enables stripping, setting `nostrip` disables it. Default is to always strip user names.

quota=*num*
Set maximum number of concurrent logins allowed from this realm to the given value (*num*).

timeout Number of seconds to wait for reply from the remote server before retransmitting the request.

retries Number of attempts to connect a server. If the server does not respond after the last attempt, the next server from the list is tried.

auth Proxy only authentication requests.

acct Proxy only accounting requests.

4.8.1 Example of 'realms' file

Example 1.

```
# Realm                 Remote server[:port]            flags
#-----------------      ---------------------           --------
that.net                radius.that.net                 nostrip
dom.ain                 server.dom.ain:3000             strip,quota=20
remote.net              srv1.remote.net,srv2.remote.net
```

Example 2.

```
# Realm                 Remote server[:port]            flags
#----------------       ---------------------           --------
NOREALM                 radius.server.net
that.net                radius.that.net                 nostrip
dom.ain                 server.dom.ain:3000             strip,quota=20
```

4.9 User Profiles — 'raddb/users'

File 'raddb/users' contains the list of *User Profiles*. See Section 2.4.5 [User Profiles], page 16, for a description of its purpose.

4.9.1 Example of 'users' file

```
## The following entry is matched when the user appends ``.ppp'' to his
## username when logging in.
## The suffix is removed from the user name, then the password is
## looked up in the SQL database.
## Users may log in at any time. They get PPP service.
DEFAULT Suffix = ".ppp",
                Auth-Type = SQL,
                Login-Time = "Al",
                Simultaneous-Use = 1,
                Strip-User-Name = Yes
        Service-Type = Framed-User,
                Framed-Protocol = PPP

## This is for SLIP users.
## This entry is matched when the auth request matches ``SLIP'' hint
DEFAULT Hint = "SLIP",
                Auth-Type = Mysql
        Service-Type = Framed-User
                Framed-Protocol = SLIP

## The following authenticates users using system passwd files.
## The users are allowed to log in from 7:55 to 23:05 on any weekday,
## except the weekend, and from 07:55 to 12:00 on Sunday.
## Only one login is allowed per user.
## The program telauth is used to further check the authentication
## information and provide the reply pairs
## Note the use of backslashes to split a long line.
DEFAULT Auth-Type = System,
                Login-Time = "Wk0755-2305,Su0755-1200",
                Simultaneous-Use = 1
        Exec-Program-Wait = "/usr/local/sbin/telauth \
                            %C{User-Name} \
                            %C{Calling-Station-Id} \
                            %C{NAS-IP-Address} \
                            %C{NAS-Port-Id}"

## This particular user is authenticated via PAM. He is presented a
## choice from 'raddb/menus/menu1' file.
gray    Auth-Type = Pam
        Menu = menu1
```

4.10 List of Blocked Users — 'raddb/access.deny'

The '`raddb/access.deny`' file contains a list of user names which are not allowed to log in via Radius. Each user name is listed on a separate line. As usual, the '`#`' character introduces an end-of-line comment.

4.11 SQL Configuration — 'raddb/sqlserver'

The '`raddb/sqlserver`' file configures the connection to SQL server.

The file uses simple line-oriented '*keyword* `---` *value*' format. Comments are introduced by '`#`' character.

The '`sqlserver`' statements can logically be subdivided into following groups: *SQL Client Parameters*, configuring the connection between SQL client and the server, *Authentication Server Parameters*, *Authorization Parameters*, and *Accounting server parameters*.

4.11.1 SQL Client Parameters

These parameters configure various aspects of connection between SQL client and the server.

`interface` *iface-type*
: Specifies the SQL interface to use. Currently supported values for *iface-type* are `mysql` and `postgres`. Depending on this, the default communication port number is set: it is 3306 for `interface mysql` and 5432 for `interface postgres`. Use of this statement is only meaningful when the package was configured with both '`--with-mysql`' and '`--with-postgres`' option.

`server` *string*
: Specifies the hostname or IP address of the SQL server.

`port` *number*
: Sets the SQL communication port number. It can be omitted if your server uses the default port.

`login` *string*
: Sets the SQL user login name.

`password` *password*
: Sets the SQL user password.

`keepopen` *bool*
: Specify whether `radiusd` should try to keep the connection open. When set to no (the default), `radiusd` will open new connection before the transaction and close it right after finishing it. We recommend setting `keepopen` to `yes` for heavily loaded servers, since opening the new connection can take a substantial amount of time and slow down the operation considerably.

`idle_timeout` *number*
: Set idle timeout in seconds for an open SQL connection. The connection is closed if it remains inactive longer that this amount of time.

4.11.2 Authentication Server Parameters

(This message will disappear, once this node revised.)

These parameters configure the SQL authentication. The general syntax is:

`doauth` *bool*
: When set to `yes`, enables authentication via SQL. All `auth_` keywords are ignored if `doauth` is set to `no`.

`auth_db` *string*
: Specifies the name of the database containing authentication information.

`auth_query` *string*
: Specifies the SQL query to be used to obtain user's password from the database. The query should return exactly one string value — the password.

`group_query` *string*
: Specifies the query that retrieves the list of user groups the user belongs to. This query is used when `Group` or `Group-Name` attribute appears in the LHS of a user's or hint's profile.

`auth_success_query` *string*
: This query is executed when an authentication succeeds. See Section 6.10 [Auth Probing], page 76, for the detailed discussion of its purpose.

`auth_failure_query` *string*
: This query is executed upon an authentication failure. See Section 6.10 [Auth Probing], page 76, for the detailed discussion of its purpose.

Example of Authentication Server Parameters

Let's suppose the authentication information is kept in the tables `passwd` and `groups`.

The `passwd` table contains user passwords. A user is allowed to have different passwords for different services. The table structure is:

```
CREATE TABLE passwd (
  user_name           varchar(32) binary default '' not null,
  service             char(16) default 'Framed-PPP' not null,
  password            char(64)
);
```

Additionally, the table `groups` contains information about user groups a particular user belongs to. Its structure is:

```
CREATE TABLE groups (
  user_name           char(32) binary default '' not null,
  user_group          char(32)
);
```

The queries used to retrieve the information from these tables will then look like:

```
auth_query  SELECT password
            FROM passwd
            WHERE user_name = '%C{User-Name}'
            AND service = '%C{Auth-Data}'

group_query SELECT user_group
            FROM groups
            WHERE user_name = '%C{User-Name}'
```

It is supposed, that the information about the particular service a user is wishing to obtain, will be kept in `Auth-Data` attribute in LHS of a user's profile.

4.11.3 Authorization Parameters

These parameters define queries used to retrieve the authorization information from the SQL database. All the queries refer to the authentication database.

`check_attr_query` *string*
: This query must return a list of triplets:

  attr-name, *attr-value*, *opcode*

  The query is executed before comparing the request with the profile entry. The values returned by the query are added to LHS of the entry. *opcode* here means one of valid operation codes: '`=`', '`!=`', '`<`', '`>`', '`<=`', '`>=`'.

`reply_attr_query` *string*
: This query must return pairs:

  attr-name, *attr-value*

  The query is executed after a successful match, the values it returns are added to the RHS list of the matched entry, and are therefore returned to the NAS in the reply packet.

Example of Authorization Parameters

Suppose your attribute information is stored in a SQL table of the following structure:

```
CREATE TABLE attrib (
  user_name varchar(32) default '' not null,
  attr      char(32) default '' not null,
  value     char(128),
  op enum("=", "!=", "<", ">", "<=", ">=") default null
);
```

Each row of the table contains the attribute-value pair for a given user. If `op` field is `NULL`, the row describes RHS (reply) pair. Otherwise, it describes a LHS (check) pair. The authorization queries for this table will look as follows:

```
check_attr_query  SELECT attr,value,op \
                  FROM attrib \
                  WHERE user_name='%u' \
                  AND op IS NOT NULL

reply_attr_query  SELECT attr,value \
                  FROM attrib \
                  WHERE user_name='%u' \
                  AND op IS NULL
```

Now, let's suppose the '`raddb/users`' contains only one entry:

```
DEFAULT Auth-Type = SQL
        Service-Type = Framed-User
```

And the `attrib` table contains following rows:

user_name	attr	value	op
jsmith	NAS-IP-Address	10.10.10.1	=
jsmith	NAS-Port-Id	20	<=
jsmith	Framed-Protocol	PPP	NULL
jsmith	Framed-IP-Address	10.10.10.11	NULL

Then, when the user `jsmith` is trying to authenticate, the following happens:

1. Radius finds the matching entry (`DEFAULT`) in the '`raddb/users`'.
2. It queries the database using the `check_attr_query`. The triplets it returns are then added to the LHS of the profile entry. Thus, the LHS will contain:

   ```
   Auth-Type = SQL,
   NAS-IP-Address = 10.10.10.1,
   NAS-Port-Id <= 20
   ```

3. Radius compares the incoming request with the LHS pairs thus obtained. If the comparison fails, it rejects the authentication. Note that the `Auth-Type` attributes itself triggers execution of `auth_query`, described in the previous section.
4. After a successful authentication, Radius queries the database, using `reply_attr_query`, and adds its return to the list of RHS pairs. The RHS pairs will then be:

   ```
   Service-Type = Framed-User,
   Framed-Protocol = PPP,
   Framed-IP-Address = 10.10.10.11
   ```

   This list is returned to the NAS along with the authentication accept packet.

Thus, this configuration allows the user `jsmith` to use only NAS 10.10.10.1, ports from 1 to 20 inclusive. If the user meets these conditions, he is allowed to use PPP service, and is assigned IP address `10.10.10.11`.

4.11.4 Accounting Parameters

To perform the SQL accounting `radiusd` needs to know the database where it is to store the accounting information. This information is supplied by the following statements:

`doacct` *`bool`*
: When set to `yes` enables SQL accounting. All `acct_` keywords are ignored if `doacct` is set to `no`.

`acct_db` *`string`*
: Specifies the name of the database where the accounting information is to be stored.

Further, `radiusd` needs to know which information it is to store into the database and when. Each of five accounting request types (see Section 2.2.2 [Accounting Requests], page 9) has a SQL query associated with it. Thus, when radius receives an accounting request, it determines the query to use by the value of `Acct-Status-Type` attribute.

Following statements define the accounting queries:

`acct_start_query` *`string`*
: Specifies the SQL query to be used when *Session Start Packet* is received. Typically, this would be some `INSERT` statement (see Section 4.11.4.1 [Queries], page 59).

`acct_stop_query` *`string`*
: Specifies the SQL query to be used when *Session Stop Packet* is received. Typically, this would be some `UPDATE` statement.

`acct_stop_query` *`string`*
: Specifies the SQL query to be executed upon arrival of a *Keepalive Packet*. Typically, this would be some `UPDATE` statement.

`acct_nasup_query` *`string`*
: Specifies the SQL query to be used upon arrival of an *Accounting Off Packet*.

`acct_nasdown_query` *`string`*
: Specifies the SQL query to be used when a NAS sends *Accounting On Packet*.

None of these queries should return any values.

Three queries are designed for use by multiple login checking mechanism (see Section 6.9 [Multiple Login Checking], page 74):

`mlc_user_query` *`string`*
: A query retrieving a list of sessions currently opened by the given user.

`mlc_realm_query` *`string`*
: A query to retrieve a list of sessions currently open for the given realm.

`mlc_stop_query` *`string`*
: A query to mark given record as *hung*.

4.11.4.1 Writing SQL Accounting Query Templates

Let's suppose you have an accounting table of the following structure:

```
CREATE TABLE calls (
  status              int(3),
  user_name           char(32),
  event_date_time     datetime DEFAULT '0000-00-00 00:00:00' NOT NULL,
  nas_ip_address      char(17),
  nas_port_id         int(6),
  acct_session_id     char(16) DEFAULT '' NOT NULL,
  acct_session_time   int(11),
  acct_input_octets   int(11),
  acct_output_octets  int(11),
  connect_term_reason int(4),
  framed_ip_address   char(17),
  called_station_id   char(32),
  calling_station_id  char(32)
);
```

On receiving the *Session Start Packet* we would insert a record into this table with `status` set to 1. At this point the columns `acct_session_time`, `acct_input_octets`, `acct_output_octets` as well as `connect_term_reason` are unknown, so we will set them to 0:

```
# Query to be used on session start
acct_start_query     INSERT INTO calls \
                     VALUES(%C{Acct-Status-Type},\
                            '%u',\
                            '%G',\
                            '%C{NAS-IP-Address}',\
                            %C{NAS-Port-Id},\
                            '%C{Acct-Session-Id}',\
                            0,\
                            0,\
                            0,\
                            0,\
                            '%C{Framed-IP-Address}',\
                            '%C{Called-Station-Id}',\
                            '%C{Calling-Station-Id}')
```

Then, when the *Session Stop Packet* request arrives we will look up the record having `status` = 1, `user_name` matching the value of `User-Name` attribute, and `acct_session_id` matching that of `Acct-Session-Id` attribute. Once the record is found, we will update it, setting

```
status = 2
acct_session_time = value of Acct-Session-Time attribute
acct_input_octets = value of Acct-Input-Octets attribute
acct_output_octets = value of Acct-Output-Octets attribute
connect_term_reason = value of Acct-Terminate-Cause attribute
```

Thus, every record with `status` = 1 will represent the active session and every record with `status` = 2 will represent the finished and correctly closed record. The constructed `acct_stop_query` is then:

```
# Query to be used on session end
acct_stop_query      UPDATE calls \
                     SET status=%C{Acct-Status-Type},\
                         acct_session_time=%C{Acct-Session-Time},\
                         acct_input_octets=%C{Acct-Input-Octets},\
                         acct_output_octets=%C{Acct-Output-Octets},\
                         connect_term_reason=%C{Acct-Terminate-Cause} \
                     WHERE user_name='%C{User-Name}' \
                     AND status = 1 \
                     AND acct_session_id='%C{Acct-Session-Id}'
```

Upon receiving a *Keepalive Packet* we will update the information stored with `acct_start_query`:

```
acct_alive_query  UPDATE calls \
                  SET acct_session_time=%C{Acct-Session-Time},\
                      acct_input_octets=%C{Acct-Input-Octets},\
                      acct_output_octets=%C{Acct-Output-Octets},\
                      framed_ip_address=%C{Framed-IP-Address} \
                  WHERE user_name='%C{User-Name}' \
                  AND status = 1 \
                  AND acct_session_id='%C{Acct-Session-Id}'
```

Further, there may be times when it is necessary to bring some NAS down. To correctly close the currently active sessions on this NAS we will define a `acct_nasdown_query` so that it would set `status` column to 2 and update `acct_session_time` in all records having `status` = 1 and `nas_ip_address` equal to IP address of the NAS. Thus, all sessions on a given NAS will be closed correctly when it brought down. The `acct_session_time` can be computed as difference between the current time and the time stored in `event_date_time` column:

```
# Query to be used when a NAS goes down, i.e. when it sends
# Accounting-Off packet
acct_nasdown_query UPDATE calls \
                   SET status=2,\
                       acct_session_time=unix_timestamp(now())-\
                               unix_timestamp(event_date_time) \
                   WHERE status=1 \
                   AND nas_ip_address='%C{NAS-IP-Address}'
```

We have not covered only one case: when a NAS crashes, e.g. due to a power failure. In this case it does not have a time to send `Accounting-Off` request and all its records remain open. But when the power supply is restored, the NAS will send an *Accounting On packet*, so we define a `acct_`

`nasup_query` to set `status` column to 3 and update `acct_session_time` in all open records belonging to this NAS. Thus we will know that each record having `status` = 3 represents a crashed session. The query constructed will be:

```
# Query to be used when a NAS goes up, i.e. when it sends
# Accounting-On packet
acct_nasup_query   UPDATE calls \
                   SET status=3,\
                       acct_session_time=unix_timestamp(now())-\
                               unix_timestamp(event_date_time) \
                   WHERE status=1 \
                   AND nas_ip_address='%C{NAS-IP-Address}'
```

If you plan to use SQL database for multiple login checking (see Section 6.9 [Multiple Login Checking], page 74), you will have to supply at least two additional queries for retrieving the information about currently active sessions for a given user and realm (see Section 6.9.1 [Retrieving Session Data], page 74). Each of these queries must return a list consisting of 5-element tuples:

user-name, *nas-ip-address*, *nas-port-id*, *acct-session-id*

For example, in our setup these queries will be:

```
mlc_user_query SELECT user_name,nas_ip_address,\
                      nas_port_id,acct_session_id \
               FROM calls \
               WHERE user_name='%C{User-Name}' \
               AND status = 1

mlc_realm_query SELECT user_name,nas_ip_address,\
                       nas_port_id,acct_session_id \
                FROM calls \
                WHERE realm_name='%C{Realm-Name}'
```

While performing multiple login checking `radiusd` will eventually need to close *hung* records, i.e. such records that are marked as open in the database (`status=1`, in our setup), but are actually not active (See Section 6.9.2 [Verifying Active Sessions], page 75, for the description of why it may be necessary). It will by default use `acct_stop_query` for that, but it has a drawback that hung records will be marked as if they were closed correctly. This may not be suitable for accounting purposes. The special query `mlc_stop_query` is provided to override `acct_stop_query`. If we mark hung records with `status=4`, then the `mlc_stop_query` will look as follows:

```
mlc_stop_query UPDATE calls \
               SET status=4,\
                acct_session_time=unix_timestamp(now())-\
                                  unix_timestamp(event_date_time) \
               WHERE user_name='%C{User-Name}' \
                 AND status = 1 \
                 AND acct_session_id='%C{Acct-Session-Id}'
```

4.12 Rewrite functions — 'raddb/rewrite'

The file 'raddb/rewrite' contains definitions of Rewrite extension functions. For information regarding Rewrite extension language See Section 10.2 [Rewrite], page 98.

4.13 Login Menus — 'raddb/menus'

The menus is a way to allow user the choice between various services he could be provided. The menu functionality is enabled when Radius is compiled with '--enable-livingston-menus' option.

A user is presented a menu after it is authenticated if the RHS of his profile record consists of a single A/V pair in the form:

```
Menu = <menu-name>
```

The menu files are stored in directory 'raddb/menus'.

4.13.1 A menu file syntax.

A menu file is a text file containing a menu declaration and any number of choice descriptions. The menus can be nested to an arbitrary depth.

A comment is introduced by a '#' character. All characters from this one up to the end of line are discarded.

The menu declaration is contained between the words 'menu' and 'end'. Each of these must be the only word on a line and must start in column 1.

Choice descriptions follow the menu declaration. Each description starts with a line containing choice identifier. A choice identifier is an arbitrary word identifying this choice, or a word 'DEFAULT'. It is followed by comma-separated list of A/V pairs which will be returned to the server when a user selects this choice.

4.13.2 An example of menu files

Single-Level Menu

Suppose the following file is stored under 'raddb/menus/menu1':

```
menu
        *** Welcome EEE user! ***
Please select an option:

        1. Start CSLIP session
        2. Start PPP session
        3. Quit

        Option:
end
# CSLIP choice
# Framed-IP-Address of 255.255.255.254 indicates that the NAS should
# select an address for the user from its own IP pool.
```

```
1
        Service-Type = Framed-User,
        Framed-Protocol = SLIP,
        Framed-IP-Address = 255.255.255.254,
        Termination-Menu = "menu1"
# PPP choice
2
        Service-Type = Framed-User,
        Framed-Protocol = PPP,
        Framed-IP-Address = 255.255.255.254,
        Termination-Menu = "menu1"
# A special menu EXIT means abort the session
3
        Menu = "EXIT"
# Return to this menu if no valid choice have been entered
DEFAULT
        Menu = "menu1"
```

Now, suppose the '`raddb/users`' contains the following profile entry:

```
DEFAULT Auth-Type = System
        Menu = "menu1"
```

and user '`jsmith`' has a valid system account and tries to log in from some NAS. Upon authenticating the user, the Radius server sees that his reply pairs contain the `Menu` attribute. Radius then sends Access-Challenge packet to the NAS with the text of the menu in it. The '`jsmith`' then sees on his terminal:

```
        *** Welcome EEE user! ***
Please select an option:

        1. Start CSLIP session
        2. Start PPP session
        3. Quit

        Option:
```

He then enters '`2`'. The NAS sends the Access-Request packet to the server, which sees that user wishes to use option 2 and replies to the NAS with an Access-Accept packet containing the following attributes:

```
        Service-Type = Framed-User,
        Framed-Protocol = PPP,
        Framed-IP-Address = 255.255.255.254,
        Termination-Menu = "menu1"
```

The `Termination-Menu` in this list makes sure the same process will continue when '`jsmith`' logs out, i.e. he will be presented the same menu again until he enters choice '`3`' which will disconnect him.

Nested menus

In this example, the '`other`' choice refers to the menu above.

```
menu
        *** Welcome here! ***
```

```
    Please enter an option:
            ppp     ---     Start PPP session
            telnet  ---     Begin guest login session
            other   ---     Select other option

            Enter your choice:
    end
    ppp
            Service-Type = Framed-User,
            Framed-Protocol = PPP
    telnet
            Service-Type = Login-User,
            Login-IP-Host = 10.11.11.7,
            Login-Service = Telnet,
            Login-TCP-Port = 23
    other
            Menu = "menu1"
    DEFAULT
            menu = "menu2"
```

4.14 Macro Substitution

Some statements in the configuration files need to use the actual values of the attributes supplied with the request. These are:

- `Exec-Program` and `Exec-Program-Wait` assignments in '`users`' database
- SQL query templates in '`sqlserver`'

In these statements the following macros are replaced by the value of corresponding attributes:

`%Cnum` (num is a decimal number). This variable is replaced by the value of attribute number 'num'. The attribute is looked up in the incoming request pairlist.

`%C{attr-name}`
This is replaced by the value of attribute named 'attr-name'. The attribute is looked up in the incoming request pairlist.

`%Rnum` (num is a decimal number). This variable is replaced by the value of attribute number 'num'. The attribute is looked up in the reply pairlist.

`%R{attr-name}`
This is replaced by the value of attribute named 'attr-name'. The attribute is looked up in the reply pairlist.

`%D` This is replaced by current date/time (localtime).

`%G` This is replaced by current date/time in GMT.

The exact substitution procedure varies depending on the type of the attribute referenced by macro. If the attribute is of string or date type,

`radiusd` first checks if the resulting substitution should be quoted. It does so by looking at the character immediately preceeding '`%`'. If it is a single or double quote, then `radiusd` assumes the macro must be quoted and replaces it by an appropriately modified attribute value. The purpose of the modification is to ensure that no characters within the expanded string would conflict with the quoting characters. In particular, `radiusd` searches the attribute value for any of the characters '`\`', '`'`', '`"`' and prepends a '`\`' to any occurrence of these. For example, suppose that attribute `NAS-Identifier` has the value '`A "new" host`'. Then:

```
nasid=%C{NAS-Identifier} ↦ nasid=A "new" host
nasid="%C{NAS-Identifier}" ↦ nasid="A \"new\" host"
nasid=%\C{NAS-Identifier} ↦ nasid=A \"new\" host
```

The last example illustrates the use of backslash character to force string quoting in the absense of explicit quotation marks.

If an integer attribute reference is quoted, `radiusd` looks up the string translation of its value in the dictionary (see Section 4.2.8 [VALUE], page 44) and uses this string as a replacement. If no translation is found, the numeric value is used. The following example assumes that the value of `Acct-Terminate-Cause` attribute is 10:

```
reason=%C{Acct-Terminate-Cause} ↦ reason=10
reason='%C{Acct-Terminate-Cause}' ↦ reason='NAS-Request'
reason=%\C{Acct-Terminate-Cause} ↦ reason=NAS-Request
```

Again, a backslash after percent sign can be used to force dictionary lookup.

The "'`{}`' form" allows to specify default value for the substitution. The default value will be used when no such attribute is encountered in the pairlist. The syntax for specifying the default value resembles that of shell environment variables.

The substitution `%C{`*`attr-name`*`:-`*`defval`*`}` is expanded to the value of *attr-name* attribute, if it is present in the request pairlist, and to *defval* otherwise. For example:

```
%C{Acct-Session-Time:-0}
```

will return the value of Acct-Session-Time attribute or 0 if it doesn't exist in the request pairlist.

The substitution `%C{`*`attr-name`*`:=`*`defval`*`}` is expanded to the value of *attr-name* attribute. If this attribute is not present in the request pairlist, it will be created and assigned the value *defval*. E.g.:

```
%C{Acct-Session-Time:=0}
```

The substitution `%C{`*`attr-name`*`:?`*`message`*`}` is expanded to the value of *attr-name* attribute, if it is present. Otherwise the diagnostic message "*attr-name*: *message*" is issued to the log error channel, and string "*message*" is returned.

The substitution `%C{`*`attr-name`*`:+`*`retval`*`}` is expanded to empty string if the attribute *attr-name* is present in the referenced pairlist. Otherwise it is expanded to *retval*.

You can also use the following shortcuts:

`%p`	Port number
`%n`	NAS IP address
`%f`	Framed IP address
`%u`	User name
`%c`	Callback-Number
`%i`	Calling-Station-Id
`%t`	MTU
`%a`	Protocol (SLIP/PPP)
`%s`	Speed (Connect-Info attribute)

5 Request Comparison Methods

The basic notions about comparison of the incoming requests and why it is necessary were given in Section 2.4.1 [Checking Duplicates], page 12. This chapter concentrates on extended methods of request comparison and on the configuration issues.

5.1 Extended Comparison

The default comparison method may fail to recognize duplicate requests. if the originating NAS has modified the request authenticator or request identifier before retransmitting the request. If you happen to use such NASes, you will have to enable *extended request comparison* to compensate for their deficiencies.

The extended request comparison consists in comparing the *contents* of both requests. However, blindly comparing each A/V pair from both requests won't work, since many attributes do change their values between successive retransmits. Therefore, `radiusd` uses only *comparable attribute*, i.e. a user-defined subset of such attributes that can safely be used in comparison. Thus, extended request comparison works as follows:

1. The comparable attributes are extracted from each request. They form two sorted *attribute lists*.
2. If lengths of both lists differ, the requests are considered different.
3. Otherwise, the value of each A/V pair from the first list is compared against that of the corresponding A/V pair from the second list. If at least one A/V pair differs, then the requests are considered different. *Notice*, that values of `Password` and `CHAP-Password` are decoded prior to comparison.

To use the extended comparison, follow the procedure below:

1. Select user-defined attribute properties.

   The syntax of dictionary file allows for nine user-defined properties, denoted by characters '`1`' through '`9`'. You should select one of them to mark comparable attributes for authentication and another one to mark those for accounting. It is strongly suggested that you use `PROPERTY` statement in your main dictionary file (see Section 4.2.7 [PROPERTY], page 44), instead of modifying `ATTRIBUTE` statements in the underlying dictionary files.

   See Section 4.2.4 [ATTRIBUTE], page 41, for detailed description of attribute property flags.

2. To enable the extended comparison for requests coming from any NAS, declare extended comparison flags in '`raddb/config`'.

   To enable the extended comparison for authentication requests, add to your `auth` block the statement

```
compare-attribute-flag flag;
```

The *flag* is the same symbol you used in the dictionary to mark comparable attributes for authentication.

To enable the extended comparison for accounting requests, insert `compare-attribute-flag` statement into the `acct` block.

3. To enable the extended comparison for requests coming from selected NASes, declare extended comparison flags in '`raddb/naslist`'.

   Add the following statement to the declaration of those NASes, that require using the extended comparison (in flags column):

   ```
   compare-auth-flag=flag,compare-acct-flag=flag
   ```

   See Section 4.4 [naslist file], page 45, for a description of naslist file syntax.

5.1.1 An example of extended comparison configuration

In this example configuration, the user-defined flag '1' marks authentication comparable attributes, and the flag '2' marks the accounting comparable attributes.

'raddb/dictionary'

```
PROPERTY       User-Name              +12
PROPERTY       Password               +1
PROPERTY       NAS-Port-Id            +12
PROPERTY       State                  +1
PROPERTY       Called-Station-Id      +12
PROPERTY       Calling-Station-Id     +12
PROPERTY       Acct-Status-Type       +2
PROPERTY       Acct-Session-Id        +2
PROPERTY       Acct-Session-Time      +2
```

'raddb/config'

```
auth {
        max-requests 127;
        request-cleanup-delay 2;
        compare-attribute-flag 1;
};
acct {
        max-requests 127;
        request-cleanup-delay 2;
        compare-attribute-flag 2;
};
```

5.1.2 List of attributes that can be declared comparable.

The following attributes can be declared as comparable:

- `User-Name`
- `Password`
- `CHAP-Password`
- `NAS-Port-Id`
- `State`
- `Called-Station-Id`
- `Calling-Station-Id`
- `NAS-Identifier`
- `Acct-Status-Type`
- `Acct-Session-Id`
- `Acct-Session-Time`
- `User-UID`
- `User-GID`

Notice that this list is by no means an exhaustive one. Depending on a particular NAS other attributes may be safe to be used in comparisons, or, vice-versa, some attributes from this list may not be used. You should carefully analyze packets coming from your NAS before deciding which attributes to mark as comparable.

5.2 Fine-Tuning the Request Queue

As described in Section 2.4.1 [Checking Duplicates], page 12, each request is added to the request queue when `radiusd` starts processing it and is removed from there a certain amount of time after its processing was finished. The configuration parameter `request-cleanup-delay` defines how long each already processed request is kept in the queue. Its value must be synchronized with the NAS settings.

Each NAS allows to configure two parameters:

`Ntimeout` The amount of time in seconds during which the NAS is waiting for a response from radius server.

`Nretries` The number of times the NAS tries to re-send the request if it received no response from the radius server.

Of course, these parameters are named differently for different makes of NASes. Refer to your NAS documentation to find out where these values are configured.

In general, these parameters must satisfy the following relation:

```
request-cleanup-delay = Nretries * Ntimeout + const
```

where *const* is an empirical constant that depends on the average time of processing a single request. Usually its value lies between 0 and 10 seconds.

For example, if the configuration of your NAS sets

```
    Nretries = 3
    Ntimeout = 10
```

then your raddb/config should contain:

```
auth {
        request-cleanup-delay 40;
};
acct {
        request-cleanup-delay 40;
};
```

Notice the duplication of `request-cleanup-delay`: `radiusd` uses distinct values for authentication and accounting requests, however most existing NASes do not make such distinction.

6 Authentication

An *Authentication Type* specifies which credentials the user is required to supply in order to be authenticated and where the user's authentication data are stored. It is defined by the value of `Auth-Type` attribute in LHS of a '`users`' entry.

6.1 Accept Authentication Type

Accept is the simplest authentication type. Users with this authentication type will be authenticated successfully without checking any credentials. Actually this means that only username is required for authentication.

This authentication type is used for each '`users`' entry, whose LHS contains

```
Auth-Type = Accept
```

This authentication type can be used for guest accounts, e.g. the following profile in '`users`':

```
guest   Auth-Type = Accept,
                Simultaneous-Use = 10
        Service-Type = Framed-User,
                Framed-Protocol = PPP
```

allows up to 10 simultaneous guest PPP accounts. To log in using such guest account it is sufficient to use username '`guest`' and any password.

6.2 Reject Authentication Type

The *Reject* authentication type causes the request to be rejected unconditionally. It can be used to disable a user account (For another method of disabling user accounts, see Section 4.10 [access.deny file], page 53).

This authentication type is used for each '`users`' entry, whose LHS contains

```
Auth-Type = Reject
```

6.3 Local Password Authentication Type

The *Local Password* authentication type allows to keep plaintext user passwords. Although the use of this authentication type is strongly discouraged for security reasons, this is the only authentication type that can be used with CHAP authentication.

There are two ways of using this authentication type

Specifying Passwords in users File.

To keep the plaintext passwords in '`users`' file, the profile entry must follow this pattern:

```
user-name  Auth-Type = Local,
                     User-Password = plaintext
```

The *plaintext* is the user's plaintext password. Obviously, *user-name* may not be `DEFAULT` nor `BEGIN`.

Specifying Passwords in SQL Database.

```
user-name   Auth-Type = Local,
                      Password-Location = SQL
```

When the user is authenticated using such profile, its password is retrieved from the authentication database using `auth_query`. The configuration of SQL authentication is described in detail in Section 4.11.2 [Authentication Server Parameters], page 54.

6.4 Encrypted Password Authentication Type

The *Encrypted Password* type allows to keep user's passwords encrypted via DES or MD5 algorithm. There are two ways of using this authentication type.

Specifying Passwords in users File.

```
user-name  Auth-Type = Crypt-Local,
                     User-Password = crypt-pass
```

The `Crypt-Password` is a shortcut for the above notation:

```
user-name  Crypt-Password = crypt-pass
```

Specifying Passwords in SQL Database.

```
user-name   Auth-Type = Crypt-Local,
                      Password-Location = SQL
```

Using this profile, the user's password is retrieved from the authentication database using `auth_query`. The configuration of SQL authentication is described in detail on Section 4.11.2 [Authentication Server Parameters], page 54.

The shortcut for this notation is `Auth-Type = SQL`.

In any case, the passwords used with this authentication type must be either DES or MD5 hashed.

6.5 System Authentication Type

The *System* authentication type requires that the user have a valid system account on the machine where the radius server is running. The use of this type is triggered by setting

```
Auth-Type = System
```

in the LHS of a '`users`' entry.

6.6 SQL Authentication Type

Setting `Auth-Type = SQL` or `Auth-Type = Mysql` in the LHS of a '`users`' entry is a synonym for

```
Auth-Type = Crypt-Local, Password-Location = SQL
```

and is provided as a shortcut and for backward compatibility with previous versions of GNU Radius.

For description of SQL authentication, see Section 6.4 [Encrypted Password Auth], page 72. The configuration of SQL subsystem is described in Section 4.11 [sqlserver file], page 53.

6.7 PAM Authentication Type

PAM authentication type indicates that a user should be authenticated using PAM (Pluggable Authentication Module) framework. The simplest way of usage is:

```
Auth-Type = PAM
```

Any user whose '`users`' profile contains the above, will be authenticated via PAM, using service name '`radius`'. If you wish to use another service name, set it using `Auth-Data` attribute, e.g.:

```
Auth-Type = PAM,
    Auth-Data = pam-service
```

6.8 Defining Custom Authentication Types

The are three ways to define custom authentication types:

1. Write a PAM module.
2. Use a Guile procedure.
3. Use an external program

You can write a PAM module implementing the new authentication type. Then, specifying `Auth-Type = PAM` allows to apply it (see Section 6.7 [PAM Auth], page 73).

Alternatively, you may write a Scheme procedure implementing the new authentication type. To apply it, use `Scheme-Procedure` attribute in RHS. The `Auth-Type = Accept` can be used in LHS if the whole authentication burden is to be passed to the Scheme procedure. For example, if one wrote a procedure `my-auth`, to apply it to all users, one will place the following profile in his '`users`' file:

```
DEFAULT  Auth-Type = Accept
         Scheme-Procedure = "my-auth"
```

For a discussion of how to write Scheme authentication procedures, See Section 10.3.2 [Authentication with Scheme], page 116.

The third way to implement your own authentication method is using an external program. This is less effective than the methods described above,

but may be necessary sometimes. To invoke the program, use the following statement in the RHS of '`users`' entry:

```
Exec-Program-Wait = "progname args"
```

The *progname* must be the full path to the program, *args* — any arguments it needs. The usual substitutions may be used in *args* to pass any request attributes to the program (see Section 4.14 [Macro Substitution], page 64).

For a detailed description of `Exec-Program-Wait` attribute and an example of its use, see Section 13.3.7 [Exec-Program-Wait], page 181.

6.9 Multiple Login Checking

The number of sessions a user can have open simultaneously can be restricted by setting `Simultaneous-Use` attribute in the user's profile LHS (see Section 13.3.25 [Simultaneous-Use], page 192). By default the number of simultaneous sessions is unlimited.

When a user with limited number of simultaneous logins authenticates himself, Radius counts the number of the sessions that are already opened by this user. If this number is equal to the value of `Simultaneous-Use` attribute the authentication request is rejected.

This process is run in several stages. First, Radius retrieves the information about currently opened sessions from one of its accounting databases. Then, it verifies whether all these sessions are still active. This pass is necessary since an open entry might be a result of missing `Stop` request. Finally, the server counts the sessions and compares their count with the value of `Simultaneous-Use` attribute.

The following subsections address each stage in detail.

6.9.1 Retrieving Session Data

Radius retrieves the list of sessions currently opened by the user either from the system database (see Section 7.1 [System Accounting], page 81), or from the SQL database (see Section 7.3 [SQL Accounting], page 83). The system administrator determines which method to use.

By default, system accounting database is used. Its advantages are simplicity and ease of handling. It has, however, a serious deficiency: the information is kept in the local files. If you run several radius servers, each of them has no easy way of knowing about the sessions initiated by other servers.

This problem is easy to solve if you run *SQL accounting* (see Section 7.3 [SQL Accounting], page 83). In this case, each radius server stores the data in your SQL database and can easily retrieve them from there.

To enable use of SQL database for multiple login checking, do the following:

In your '`raddb/config`' file set:

```
mlc {
    method sql;
};
```

In your '`raddb/sqlserver`' file, specify the queries for retrieving the information about open sessions and, optionally, a query to close an existing open record.

There are two queries for retrieving the information: `mlc_user_query` returns the list of sessions opened by the user, `mlc_realm_query` returns the list of sessions opened for the given realm. Each of them should return a list of 4-element tuples[1]:

user-name, nas-ip-address, nas-port-id, acct-session-id

Here is an example of `mlc_user_query` and `mlc_realm_query`:

```
mlc_user_query SELECT user_name,nas_ip_address,\
                      nas_port_id,acct_session_id \
               FROM calls \
               WHERE user_name='%C{User-Name}' \
               AND status = 1

mlc_realm_query SELECT user_name,nas_ip_address,\
                       nas_port_id,acct_session_id \
                FROM calls \
                WHERE realm_name='%C{Realm-Name}'
```

Apart from these two queries you may also wish to provide a query for closing a hung record. By default, `radiusd` will use `acct_stop_query`. If you wish to override it, supply a query named `mlc_stop_query`, for example:

```
mlc_stop_query UPDATE calls \
               SET status=4,\
                acct_session_time=unix_timestamp(now())-\
                                  unix_timestamp(event_date_time) \
               WHERE user_name='%C{User-Name}' \
                 AND status = 1 \
                 AND acct_session_id='%C{Acct-Session-Id}'
```

See Section 4.11.4.1 [Queries], page 59, for detailed information on how to write these queries.

6.9.2 Verifying Active Sessions

Whatever database `radiusd` uses, an open entry in it does not necessary mean that the corresponding session is still being active. So, after retrieving the information about user sessions, Radius verifies on corresponding NASes whether these are actually active.

For each entry in the session list, if its NAS acknowledges the session, the session count is incremented. Otherwise, such entry is marked as closed in the database and is not counted.

There may also be cases when the NAS is unreachable due to some reasons. In such cases the Radius behavior is determined by the value of

[1] This interface is likely to change in future versions

`checkrad-assume-logged` in '`config`' file Section 4.1.3 [auth statement (raddb/config)], page 28. If the value is `yes`, Radius assumes the session is still active and increases the session count, otherwise it proceeds as if the NAS returned negative reply.

To query a NAS, Radius first looks up its type and additional parameters in '`naslist`' file (see Section 4.4 [naslist file], page 45). There are two predefined NAS types that cause Radius to act immediately without querying tne NAS: the special type '`true`' forces Radius to act as if the NAS returned 1, the type '`false`' forces it to act as if the NAS returned 0. If the type is neither of this predefined types, Radius uses it as a look up key into the '`nastypes`' file (see Section 4.5 [nastypes file], page 47) and tries to retrieve an entry which has matching type. If such entry does not exist, Radius issues the error message and acts accordingly to the value of configuration variable `checkrad-assume-logged`. Otherwise, Radius determines the query method to use from the second field of this entry, and constructs method arguments by appending arguments from the '`naslist`' entry to those of `nastypes` entry. Note, that the former take precedence over the latter, and can thus be used to override default values specified in '`nastypes`'.

Having determined the query method and its argument, Radius queries NAS and analyzes its output by invoking a user-supplied Rewrite function. The function to use is specified by the `function=` argument to the method. It is called each time a line of output is received from the NAS (for finger queries) or a variable is received (for SNMP queries). The process continues until the function returns 1 or the last line of output is read or a timeout occurs whichever comes first.

If the user-function returns 1 it is taken to mean the user's session is now active at the NAS, otherwise, if it replies 0 or if the end of output is reached, it is taken to mean the user's session is not active.

The syntax conventions for user-supplied functions are described in detail in Section 10.2.5 [Login Verification Functions], page 103.

6.10 Controlling Authentication Probes

Authentication probe is an attempt of a user to use other user's account, by guessing his password. The obvious indication of an authentication probe is appearence of several consecutive authentication failures for the same user. Of course, if the intruder is given sufficient number of such probes he will sooner or later succeed in finding the actual password. The conventional method to prevent this from occurring is to keep *failure counters* for each user and to lock the account when its failure counter reaches a predefined limit. Notice that a legitimate user may fail (sometimes even several times in sequence) in entering his password, so two important points should always be observed. First, failure counters record the number of consecutive authentication failures and they are reset after each successive authentication.

Secondly, the maximum number of allowed consecutive failures should be set sufficiently high.

The version 1.6 offers two ways for controlling authentication probes: using external programs and using special SQL queries.

To control authentication probes using external programs, use the combination of `Exec-Program-Wait` and `Auth-Failure-Trigger`. The program specified by `Auth-Failure-Trigger` is executed each time an authentication attempt failed. When both attributes are used together, the program invoked by `Auth-Failure-Trigger` can update the failure counter, and the one invoked by `Exec-Program-Wait` can compare the counter value with the predefined limit and reject authentication when both values become equal. Such approach is most useful in conjunction with `BEGIN` profile.

Let's suppose the program '`/sbin/check_failure`' accepts a user name and returns 1 if the failure counter for this user has reached maximum allowed value. Otherwise it returns 0 and clears the counter. Another program, '`/sbin/count_failure`' increases failure counter value for the given user name. Assuming our basic authentication type is '`PAM`', the '`raddb/users`' file will look as follows:

```
BEGIN   NULL
        Exec-Program-Wait = "/sbin/check_failure  %C{User-Name}",
        Auth-Failure-Trigger = "/sbin/count_failure %C{User-Name}",
                Fall-Through = Yes

DEFAULT Auth-Type = PAM
        Service-Type = Framed-User,
                Framed-Protocol = PPP

[... Other profiles ...]
```

The `BEGIN` profile will be executed before any other profile. It will add to the RHS `Exec-Program-Wait` and `Auth-Failure-Trigger` attributes and then `radiusd` will proceed to finding a matching profile (due to `Fall-Through` attribute). When such profile is found, the user will be authenticated according to the method set up by the profile's `Auth-Type` attribute. If authentication fails, '`/sbin/count_failure`' will be called and the user name passed to it as the argument. Otherwise, '`/sbin/check_failure`' will be invoked.

To complete the example, here are working versions of both programs. Failure counters for each user name are kept in separate file in '`/var/log/radius/fails`' directory. Both programs are written in `bash`.

The /sbin/count_failure program

```
#! /bin/bash

test $# -eq 1 || exit 1

MAXFAIL=8
REGDIR=/var/log/radius/fails

if [ -r "$REGDIR/$1" ]; then
  read COUNT < "$REGDIR/$1"
  COUNT=$((COUNT+1))
else
  COUNT=1
fi
echo $COUNT > "$REGDIR/$1"
# End of /sbin/count_failure
```

The /sbin/check_failure program

```
#! /bin/bash

test $# -eq 1 || exit 1

MAXFAIL=8
REGDIR=/var/log/radius/fails

if [ -r "$REGDIR/$1" ]; then
  read COUNT < "$REGDIR/$1"
  if [ $COUNT -ge $MAXFAIL ]; then
    echo "Reply-Message=\"Too many login failures. Your account is locked\""
    exit 1
  else
    rm "$REGDIR/$1"
  fi
fi
exit 0

# End of check_failure
```

Another way of controlling authentication probes is by using SQL database to store failure counters. Two queries are provided for this purpose in '`raddb/sqlserver`' file: `auth_success_query` is executed upon each successful authentication, and `auth_failure_query` is executed upon each authentication failure. Both queries are not expected to return any values. One obvious purpose of `auth_failure_query` would be to update failure counters and that of `auth_success_query` would be to clear them. The `auth_query` or `group_query` should then be modified to take into account the number of authentication failures.

The default SQL configuration GNU Radius is shipped with provides a working example of using these queries. Let's consider this example.

First, we create a special table for keeping authentication failure counters for each user:

```
CREATE TABLE authfail (
  # User name this entry refers to
  user_name           varchar(32) binary default '' not null,
  # Number of successive authentication failures for this user
  count               int,
  # Timestamp when this entry was last updated
  time                datetime DEFAULT '1970-01-01 00:00:00' NOT NULL,
  # Create a unique index on user_name
  UNIQUE uname (user_name)
);
```

The query `auth_fail_query` will increment the value of `count` column for the user in question:

```
auth_failure_query UPDATE authfail \
                   SET count=count+1,time=now() \
                   WHERE user_name='%C{User-Name}'
```

The query `auth_success_query` will clear `count`:

```
auth_success_query UPDATE authfail \
                   SET count=0,time=now() \
                   WHERE user_name='%C{User-Name}'
```

Now, the question is: how to use this counter in authentication? The answer is quite simple. First, let's create a special group for all the users whose authentication failure counter has reached its maximum value. Let this group be called '`*LOCKED_ACCOUNT*`'. We'll add the following entry to '`raddb/users`':

```
DEFAULT Group = "*LOCKED_ACCOUNT*",
                Auth-Type = Reject
        Reply-Message = "Your account is currently locked.\n\
Please, contact your system administrator\n"
```

which will reject all such users with an appropriate reply message.

The only thing left now is to rewrite `group_query` so that it returns '`*LOCKED_ACCOUNT*`' when `authfail.count` reaches its maximum value. Let's say this maximum value is 8. Then the following query will do the job:

```
group_query       SELECT user_group FROM groups \
                  WHERE user_name='%u' \
                  UNION \
                  SELECT CASE WHEN (SELECT count > 8 FROM authfail \
                                                    WHERE user_name='%u')
                         THEN '*LOCKED_ACCOUNT*' END
```

The default configuration comes with these queries commented out. It is up to you to uncomment them if you wish to use SQL-based control over authentication failures.

Notice the following important points when using this approach:

1. Your SQL server must support `UNION`. Earlier versions of `MySQL` lacked this support, so if you run `MySQL` make sure you run a reasonably new version (at least 4.0.18).
2. Both `auth_failure_query` and `auth_success_query` assume the database already contains an entry for each user. So, when adding a new user to the database, make sure to insert an appropriate record into `authfails` table, e.g.

```
INSERT INTO authfail VALUES('new-user',0,now());
```

7 Accounting

By default GNU Radius supports three types of accounting. Any additional accounting methods can be defined using extension mechanisms.

The accounting methods are applied to a request in a following sequence:

1. System accounting
2. Detailed request accounting
3. sql accounting
4. Custom accounting

Any method can be enabled or disabled. Thus, you can even disable them all, thereby disabling accounting altogether.

Notice, that the multiple login checking scheme relies on accounting being enabled. By default it uses system accounting, but can also be configured to use sql accounting. So, if you disable system accounting and still wish to use reliable multiple login checking, make sure you configure `radiusd` to use sql for this purpose. See Section 6.9 [Multiple Login Checking], page 74, for the detailed information about the subject.

If any accounting type in this sequence fails, the accounting is deemed to fail and all subsequent methods are not invoked.

7.1 System Accounting

Radius keeps files '`radutmp`' and '`radwtmp`' in its logging directory and stores the accounting data there. The utilities `radwho` and `radlast` can be used to list information about users' sessions.

This accounting method is enabled by default. To disable it, use `system no` statement in '`raddb/config`'. See Section 4.1.4 [acct], page 30, for more information. Please notice that disabling this authentication method will disable multiple login checking as well. Refer to Section 6.9 [Multiple Login Checking], page 74, for the detailed discussion of this.

7.2 Detailed Request Accounting

Radius stores the detailed information about accounting packets it receives in files '`radacct/`*`nasname`*`/detail`' (see Chapter 1 [Naming Conventions], page 5), where *nasname* is replaced with the short name of the NAS from the '`raddb/naslist`' file (see Section 4.4 [naslist file], page 45).

By default, this accounting type is always enabled, provided that '`radacct`' directory exists and is writable (see Chapter 1 [Naming Conventions], page 5). To turn the detailed accounting off, use the `detail` statement in the '`config`' file. For more information about it, see Section 4.1.4 [acct], page 30.

The accounting detail files consist of a record for each accounting request. A record includes the timestamp and detailed dump of attributes from the packet, e.g.:

```
Fri Dec 15 18:00:24 2000
        Acct-Session-Id = "2193976896017"
        User-Name = "e2"
        Acct-Status-Type = Start
        Acct-Authentic = RADIUS
        Service-Type = Framed-User
        Framed-Protocol = PPP
        Framed-IP-Address = 11.10.10.125
        Calling-Station-Id = "+15678023561"
        NAS-IP-Address = 11.10.10.11
        NAS-Port-Id = 8
        Acct-Delay-Time = 0
        Timestamp = 976896024
        Request-Authenticator = Unverified

Fri Dec 15 18:32:09 2000
        Acct-Session-Id = "2193976896017"
        User-Name = "e2"
        Acct-Status-Type = Stop
        Acct-Authentic = RADIUS
        Acct-Output-Octets = 5382
        Acct-Input-Octets = 7761
        Service-Type = Framed-User
        Framed-Protocol = PPP
        Framed-IP-Address = 11.10.10.125
        Acct-Session-Time = 1905
        NAS-IP-Address = 11.10.10.11
        NAS-Port-Id = 8
        Acct-Delay-Time = 0
        Timestamp = 976897929
        Request-Authenticator = Unverified
```

Notice that `radiusd` always adds two pseudo-attributes to detailed listings. Attribute `Timestamp` shows the UNIX timestamp when `radiusd` has received the request. Attribute `Request-Authenticator` shows the result of checking the request authenticator. Its possible values are:

Verified — The authenticator check was successful.

Unverified — The authenticator check failed. This could mean that either the request was forged or that the remote NAS and `radiusd` do not agree on the value of the shared secret.

None — The authenticator check is not applicable for this request type.

Notice also that the so-called *internal attributes* by default are not logged in the detail file. Internal attributes are those whose decimal value is greater than 255. Such attributes are used internally by radius and cannot be transferred via RADIUS protocol. Examples of such attributes are `Fall-Through`, `Hint` and `Huntgroup-Name`. See Section 13.3 [Radius Internal Attributes], page 178, for detailed listing of all internal attributes. The special attribute flag `l` (lower-case ell) may be used to force logging of such attributes (see Section 4.2.4 [ATTRIBUTE], page 41).

7.3 sql Accounting

The sql accounting method is enabled when Radius is configured with '`--enable-sql`' option and the '`sqlserver`' file in its configuration directory is properly set up (see Section 4.11 [sqlserver file], page 53).

This version of GNU Radius (1.6) supports MySQL and PostgreSQL servers. It also supports odbc, which can be used to build interfaces to another database management systems.

With this accounting method enabled, `radiusd` will store the information about accounting requests in the configured sql database. The accounting method is fully configurable: the Radius administrator defines both the types of requests to be accounted and the information to be stored into the database (see Section 4.11 [sqlserver file], page 53).

7.4 Defining Custom Accounting Types

If the built-in accounting methods do not meet your requirements, you can implement your own. There are two ways of doing so:

1. Using a Guile procedure.
2. Using an external program

To use a Guile procedure for accounting, the name of the procedure must be specified as a value to the `Scheme-Acct-Procedure` attribute in the RHS list of a '`hints`' entry, e.g.:

```
DEFAULT NULL Scheme-Acct-Procedure = "my-acct"
```

For a detailed description of Scheme accounting procedures, see Section 10.3.3 [Accounting with Scheme], page 117.

Another way of implementing your own accounting method is using an external program. This is less effective than the methods described above, but may be necessary sometimes. To invoke the program, use the following statement in the LHS of the '`hints`' entry:

```
Acct-Ext-Program = "progname args"
```

The *progname* must be the full path to the program, and *args* any arguments it needs. The usual substitutions may be used in *args* to pass any request attributes to the program (see Section 4.14 [Macro Substitution], page 64).

For a detailed description of `Acct-Ext-Program`, see Section 13.3.1 [Acct-Ext-Program], page 178.

8 Logging

GNU Radius reports every event worth mentioning. The events are segregated by their severity level. Radius discerns the following levels (in order of increasing severity):

`Debug` The debug messages (Section 9.2 [Debugging], page 89).

`Auth` Under this level every authentication attempt is logged. This is enabled by setting

```
level auth;
```

in the `category auth` statement of the '`config`' file.

`Proxy` Messages regarding proxy requests (see Section 2.4.2 [Proxying], page 13).

`Info` Informational messages.

`Notice` Normal, but significant conditions.

`Warning` Warning conditions. These mean some deviations from normal work.

`Error` Error conditions. Usually these require special attention.

`CRIT` Critical conditions due to which Radius is no longer able to continue working. These require urgent actions from the site administrator.

By default, all messages in all levels are output to the file '`radlog/radius.log`'. In addition, messages in level `CRIT` are also duplicated to the system console. These defaults can be overridden using `logging` statement in the '`raddb/config`' file. (See Section 4.1.2 [logging statement], page 24, for the description of logging statement syntax; see Chapter 1 [Naming Conventions], page 5 for information about the locations of different Radius configuration files.)

9 Problem Tracking

9.1 Rule Tracing

If you have more than one entry in your '`users`' file it is not always obvious which of the entries were used for authentication. The authentication data flow becomes even harder to understand if there are some complex rules in the '`hints`' and '`huntgroups`' files.

The rule tracing mode is intended to help you find out the exact order of the rules that each request matched during processing. The mode is toggled by `trace-rules` statement in `auth` or `acct` block of your '`config`' file. When rule tracing mode is on for a given type of requests, `radiusd` will display the data flow diagram for each processed request of this type. The diagram is output on `info` logging category, it represents the list of rules in reverse chronological order. Each rule is represented by its location in the form *filename*:*line*. To make the output more compact, if several rules appear in the same configuration file, their locations are listed as a comma-separated list of numbers after the file name. Furthermore, if the configuration files have the same path prefix, then only the first file name appears with the full prefix.

Here is an example of trace rule diagram:

```
Oct 31 11:37:17 [28322]: Auth.info: (Access-Request foo 170 bar):
rule trace: /etc/raddb/users:157,22,3; huntgroups:72; hints:34
```

This diagram means, that the authentication request from server '`foo`' for user '`bar`' with ID 170 matched the following rules

File name	Line number
'`/etc/raddb/hints`'	34
'`/etc/raddb/huntgroups`'	72
'`/etc/raddb/users`'	3
'`/etc/raddb/users`'	22
'`/etc/raddb/users`'	157

As a practical example, let's suppose you have the following setup. There are three classes of users:

1. Users from group "root" are authenticated using system password database and get rlogin access to the server 192.168.10.1
2. Users from group "staff" are also authenticated using system password database, but they are granted only telnet access to the server 192.168.10.2
3. Finally, the rest of users is authenticated against SQL database and get usual PPP access.

In addition, users from the first two classes are accounted using custom Scheme procedure `staff-acct`.

The configuration files for this setup are showed below:

Contents of '`hints`':

```
DEFAULT  Group = "root"
         Scheme-Acct-Procedure = "staff-acct",
                   Hint = "admin"

DEFAULT  Group = "staff"
         Scheme-Acct-Procedure = "staff-acct",
                   Hint = "staff"
```

Contents of file '`users`':

```
DEFAULT Auth-Type = SQL,
              Simultaneous-Use = 1
        Service-Type = Framed-User,
              Framed-Protocol = PPP

DEFAULT Hint = "admin",
             Auth-Type = System
        Service-Type = Login-User,
             Login-IP-Host = 192.168.0.1,
             Login-Service = Rlogin

DEFAULT Hint = "staff",
              Auth-Type = System,
              Simultaneous-Use = 1
         Service-Type = Login-User,
              Login-IP-Host = 192.168.0.2,
              Login-Service = Telnet
```

Now, let's suppose that user '`svp`' is in the group '`staff`' and is trying to log in. However, he fails to do so and in `radiusd` logs you see:

```
Nov 06 21:25:24: Auth.notice: (Access-Request local 61 svp):
  Login incorrect [svp]
```

Why? To answer this question, you add to `auth` block of your '`config`' the statement

```
trace-rules yes;
```

and ask user '`svp`' to retry his attempt. Now you see in your logs:

```
Nov 06 21:31:24: Auth.notice: (Access-Request local 13 svp):
  Login incorrect [svp]
Nov 06 21:31:24: Auth.info: (Access-Request local 13 svp):
  rule trace: /etc/raddb/users:1, hints: 5
```

This means that the request for '`svp`' has first matched rule on the line 1 of file '`hints`', then the rule on line 1 of file '`users`'. Now you see the error:

the entries in 'users' appear in wrong order! After fixing it your 'users' looks like:

```
DEFAULT Hint = "admin",
             Auth-Type = System
        Service-Type = Login-User,
             Login-IP-Host = 192.168.0.1,
             Login-Service = Rlogin

DEFAULT  Hint = "staff",
              Auth-Type = System,
              Simultaneous-Use = 1
         Service-Type = Login-User,
              Login-IP-Host = 192.168.0.2,
              Login-Service = Telnet

DEFAULT Auth-Type = SQL,
              Simultaneous-Use = 1
        Service-Type = Framed-User,
              Framed-Protocol = PPP
```

Now, you ask 'svp' to log in again, and see:

```
Nov 06 21:35:14: Auth.notice: (Access-Request local 42 svp):
  Login OK [svp]
Nov 06 21:35:14: Auth.info: (Access-Request local 42 svp):
  rule trace: /etc/raddb/users:7, hints: 5
```

Let's also suppose that user 'plog' is not listed in groups "root" and "staff", so he is supposed to authenticate using SQL. When he logs in, you see in your logs:

```
Nov 06 21:39:05: Auth.notice: (Access-Request local 122 plog):
  Login OK [svp]
Nov 06 21:39:05: Auth.info: (Access-Request local 122 plog):
  rule trace: /etc/raddb/users:14
```

9.2 Debugging

GNU Radius provides extensive debugging features. These are enabled either by the '`--debug`' ('`-x`') command line option to `radiusd` (see Chapter 3 [Invocation], page 17), or by the `level` statement in the debug category (see Section 4.1.2 [logging statement], page 24). Both cases require as an argument a valid debug specification.

A debug specification sets the module for which the debugging should be enabled and the debugging level. The higher the level is, the more detailed information is provided. The module name and level are separated by an equal sign. If the level is omitted, the highest possible level (100) is assumed. The module name may be abbreviated to the first N characters, in which

case the first matching module is selected. Several such specifications can be specified, in which case they should be separated by commas. For example, the following is a valid debug specification:

```
proxy.c=10,files.c,config.y=1
```

It sets debug level 10 for module `proxy.c`, 100 for `files.c`, and 1 for `config.y`.

The modules and debugging levels are subject to change from release to release.

9.3 Test Mode

Test mode is used to test various aspects of radius configuration, without starting the daemon. To enter test mode, run

```
radiusd -mt
```

You will see usual `radiusd` diagnostics and the following two lines:

```
** TEST SHELL **
(radiusd) _
```

The string '`** TEST SHELL **`' indicates that `radiusd` has entered test mode, the string '`(radiusd)`' is the shell prompt, indicating that `radiusd` is waiting for your commands.

The syntax of test shell command resembles that of Bourne shell: each command consists of a list of *words* separated by any amount of whitespace. Each word is either a sequence of *allowed word characters* (i.e. alphabetical characters, decimal digits, dashes and underscores), or any sequence of characters enclosed in a pair of double quotes. The very first word is a *command verb*, the rest of words are arguments to this command verb. A command verb may be used in its full form, in its abbreviated form (i.e. you may type only several first characters of the verb, the only condition being that they do not coincide with another command verb), or in it's short form.

The first command you should know is `help` (or, in its short form, `h`). This command takes no arguments and displays the short summary of all the available commands. Here is an example of its output:

```
(radiusd) help
h       help                            Print this help screen
q       query-nas NAS LOGIN SID PORT [IP]
                                        Query the given NAS
g       guile                           Enter Guile
rs      rewrite-stack [NUMBER]          Print or set the Rewrite
                                        stack size
r       run-rewrite FUNCTION(args..)    Run given Rewrite function
s       source FILE                     Source the given Rewrite file
t       timespan TIMESPAN [DOW [HH [MM]]]
                                        Check the timespan interval
d       debug LEVEL                     Set debugging level
rd      request-define [PAIR [,PAIR]]   Define a request
rp      request-print                   Print the request
quit    quit                            Quit the shell
```

Each line of the output consists of three fields. The first field shows the short command form. The second one lists its full form and its arguments, optional arguments being enclosed in square brackets. The third field contains short textual description of the command.

`query-nas` *`nas login sid port`* [*`ip`*] [Test Shell Command]
`q` [Test Shell Abbreviation]

Queries the given NAS about the session described by its arguments. This command is useful in testing simultaneous login verification (see Section 6.9 [Multiple Login Checking], page 74. Its arguments are

nas Specifies the NAS to query. It cn be its short name as defined in '`raddb/naslist`', or its fully qualified domain name, or its IP address.

login Name of the user whose session should be verified.

sid Session ID.

port Port number on the NAS.

ip Framed IP address, assigned to the user.

The command displays the following result codes:

0 The session is not active.

1 The session is active

-1 Some error occurred.

`guile` [Test Shell Command]
`g` [Test Shell Abbreviation]

Enter Guile shell. The command is only available if the package has been compiled with Guile support. For more information, See Section 10.3 [Guile], page 115.

`rewrite-stack` [*number*] [Test Shell Command]
`rs` [Test Shell Abbreviation]
Prints or sets the Rewrite stack size.

`run-rewrite` *function*(*args* ...) [Test Shell Command]
`r` [Test Shell Abbreviation]
Runs given Rewrite *function* and displays its return value. Function arguments are specified in the usual way, i.e. as a comma-separated list of Rewrite tokens.

If the function being tested operates on request contents (see Section 10.2.4 [Rewriting Incoming Requests], page 99), you may supply the request using `request-define` command (see below).

`source` *file* [Test Shell Command]
`s` [Test Shell Abbreviation]
Reads and compiles ("source") the given Rewrite *file*. The command prints '`0`' if the file was compiled successfully. Otherwise, it prints '`1`' and any relevant diagnostics.

`timespan` *timespan* [*day-of-week* [*hour* [*minutes*]]] [Test Shell Command]
`t` [Test Shell Abbreviation]
Checks whether the given time falls within the timespan interval. Its first argument, *timespan*, contains the valid `radiusd` timespan specification (see Section 13.3.14 [Login-Time], page 187). Rest of arguments define the time. If any of these is omitted, the corresponding value from current local time is used.

day-of-week
Ordinal day of week number, counted from 0. I.e.: Sunday – 0, Monday – 1, etc.

hour Hours counted from 0 to 24.

minutes Minutes.

The following set of samples illustrates this command:

```
(radiusd) timespan Wk0900-1800
ctime: Tue Dec  2 16:08:47 2003
inside Wk0900-1800: 6720 seconds left

(radiusd) timespan Wk0900-1800 0
ctime: Sun Nov 30 16:09:03 2003
OUTSIDE Wk0900-1800: 60660 seconds to wait

(radiusd) timespan Wk0900-1800 0 12 30
ctime: Sun Nov 30 12:30:13 2003
OUTSIDE Wk0900-1800: 73800 seconds to wait

(radiusd) timespan Wk0900-1800 1 05 00
ctime: Mon Dec  1 05:00:33 2003
OUTSIDE Wk0900-1800: 14400 seconds to wait

(radiusd) timespan Wk0900-1800 1 09 10
ctime: Wed Jan  7 22:09:41 2004
OUTSIDE Wk0900-1800: 39060 seconds to wait

(radiusd) timespan Wk0900-1800 1 09 10
ctime: Mon Dec  1 09:10:44 2003
inside Wk0900-1800: 31800 seconds left

(radiusd)
```

`debug` *`level`* [Test Shell Command]
`d` [Test Shell Abbreviation]

Set debugging level. *Level* is any valid debug level specification (see Section 9.2 [Debugging], page 89).

`request-define` [*`pair`* [,*`pair`*]] [Test Shell Command]
`rd` [Test Shell Abbreviation]

Define a request for testing Rewrite functions. The optional arguments are a comma-separated list of A/V pairs. If they are omitted, the command enters interactive mode, allowing you to enter the desired A/V pairs, as in the following example:

```
(radiusd) request-define
Enter the pair list. End with end of file
[radiusd] User-Name = smith, User-Password = guessme
[radiusd] NAS-IP-Address = 10.10.10.1
[radiusd] NAS-Port-Id = 34
[radiusd]
(radiusd)
```

Notice that any number of A/V pairs may be specified in a line. To finish entering the request, either type an EOF character or enter an empty line.

`request-print` [Test Shell Command]
`rp` [Test Shell Abbreviation]

Prints the request, defined by `request-define`.

```
(radiusd) request-print
    User-Name = (STRING) smith
    User-Password = (STRING) guessme
    NAS-IP-Address = (IPADDR) 10.10.10.1
    NAS-Port-Id = (INTEGER) 34
(radiusd)
```

`quit` [Test Shell Command]

Immediately quits the shell.

10 Extensions

The use of extension language allows extending the functionality of GNU Radius without having to modify its source code. The two extension languages supported are Rewrite and Scheme. Use of Rewrite is always enabled. Use of Scheme requires Guile version 1.4 or higher.

10.1 Filters

The simplest way to extend the functionality of Radius is to use filters. A filter is an external program that communicates with Radius via its standard input and output channels.

10.1.1 Getting Acquainted with Filters

Suppose we wish to implement an authentication method based on the user name and the user's calling station ID. We have a database of user names with valid IDs, and the new method should authenticate a user only if the combination {*user_name*, *id*} is found in this database.

We write a filter program that reads its standard input line by line. Each input line must consist of exactly two words: the user name and the calling station ID. For each input line, the program prints `0` if the {*user_name*, *id*} is found in the database and `1` otherwise. Let's suppose for the sake of example that the database is a plaintext file and the filter is written in a shell programming language. Then it will look like

```
#! /bin/sh

DB=/var/db/userlist

while read NAME CLID
do
    if grep "$1:$2" $DB; then
        echo "0"
    else
        echo "1"
    fi
done
```

10.1.2 Declaring the Filter

Here is how this filter is declared in the '`raddb/config`' file:

```
filters {
    filter check_clid {
        exec-path "/usr/libexec/myfilter";
        error-log "myfilter.log";
        auth {
            input-format "%C{User-Name}
            %C{Calling-Station-Id}";
            wait-reply yes;
```

```
        };
    };
};
```

Let's analyze this declaration line by line:

1. `filters {`

   This keyword opens the filters declaration block. The block may contain several declarations.

2. `filter check_clid {`

   This line starts the declaration of this particular filter and names it '`check_clid`'.

3. `exec-path "/usr/libexec/myfilter";`

   This line tells `radiusd` where to find the executable image of this filter.

4. `error-log "myfilter.log";`

   The diagnostic output from this filter must be redirected to the file '`myfilter.log`' in the current logging directory

5. `auth {`

   This filter will process authentication requests.

6. `input-format "%C{User-Name} %C{Calling-Station-Id}";`

   Define the input line format for this filter. The %C`{}` expressions will be replaced by the values of the corresponding attributes from the incoming request (see Section 4.14 [Macro Substitution], page 64).

7. `wait-reply yes;`

   `radiusd` will wait for the reply from this filter to decide whether to authenticate the user.

10.1.3 Invoking the Filter from a User Profile

To invoke this filter from the user profile, specify its name prefixed with '`|`' in the value of `Exec-Program-Wait` attribute, like this:

```
DEFAULT Auth-Type = System,
                Simultaneous-Use = 1
        Exec-Program-Wait = "|check_clid"
```

10.1.4 Adding Reply Attributes

Apart from simply deciding whether to authenticate a user, the filter can also modify the reply pairs.

```
#! /bin/sh

DB=/var/db/userlist

while read NAME CLID
do
    if grep "$1:$2" $DB; then
        echo "0 Service-Type = Login, Session-Timeout = 1200"
```

```
    else
        echo "1 Reply-Message =
              \"You are not authorized to log in\""
    fi
done
```

10.1.5 Accounting Filters

Let's suppose we further modify our filter to also handle accounting requests. To discern between the authentication and accounting requests we'll prefix each authentication request with the word '`auth`' and each accounting request with the word '`acct`'. Furthermore, the input line for accounting requests will contain a timestamp.

Now, our filter program will look as follows:

```
#! /bin/sh

AUTH_DB=/var/db/userlist
ACCT_DB=/var/db/acct.db

while read CODE NAME CLID DATE
do
    case CODE
    auth)
        if grep "$1:$2" $DB; then
            echo "0 Service-Type = Login, \
                  Session-Timeout = 1200"
        else
            echo "1 Reply-Message = \
                  \"You are not authorized to log in\""
        fi
    acct)
        echo "$CODE $NAME $CLID $DATE" >> $ACCT_DB
done
```

Its declaration in the '`raddb/config`' will also change:

```
filter check_clid {
    exec-path "/usr/libexec/myfilter";
    error-log "myfilter.log";
    auth {
        input-format "auth %C{User-Name}
                      %C{Calling-Station-Id}";
        wait-reply yes;
    };
    acct {
        input-format "acct %C{User-Name}
                      %C{Calling-Station-Id} %D";
        wait-reply no;
    };
};
```

(The `input-format` lines are split for readability. Each of them is actually one line).

Notice `wait-reply no` in the `acct` statement. It tells `radiusd` that it shouldn't wait for the response on accounting requests from the filter.

10.1.6 Invoking the Accounting Filter

To invoke the accounting filter, specify its name prefixed with a vertical bar character as a value of `Acct-Ext-Program` in our '`raddb/hints`' file. For example:

```
DEFAULT NULL
        Acct-Ext-Program = "|check_clid:
```

10.2 Rewrite

Rewrite is the GNU Radius extension language. Its name reflects the fact that it was originally designed to rewrite the broken request packets so they could be processed as usual (see Section 10.2.4 [Rewriting Incoming Requests], page 99). Beside this basic use, however, Rewrite functions are used to control various aspects of GNU Radius functionality, such as verifying the activity of user sessions, controlling the amount of information displayed in log messages, etc (see Section 10.2.3 [Interaction with Radius], page 99).

10.2.1 Syntax Overview

The syntax of Rewrite resembles that of C. Rewrite has two basic data types: integer and string. It does not have global variables; all variables are automatic. The only exceptions are the A/V pairs from the incoming request, which are accessible to Rewrite functions via the special notation `%[attr]`.

10.2.2 Quick Start

As an example, let's consider the following Rewrite function:

```
string
foo(integer i)
{
    string rc;
    if (i % 2)
        rc = "odd";
    else
        rc = "even";
    return "the number is " + rc;
}
```

The function takes an integer argument and returns the string '`the number is odd`' or '`the number is even`', depending on the value of `i`. This illustrates the fact that in Rewrite the addition operator is defined on the string type. The result of such operation is the concatenation of operands.

Another example is a function that adds a prefix to the `User-Name` attribute:

```
integer
px_add()
{
        %[User-Name] = "pfx-" + %[User-Name];
        return 0;
}
```

This function manipulates the contents of the incoming request; its return value has no special meaning.

10.2.3 Interaction with Radius

A Rewrite function can be invoked in several ways, depending on its purpose. There are three major kinds of Rewrite functions:

- Functions used to rewrite the incoming requests.
- Functions designed for simultaneous login verification.
- Functions used to generate Radius attribute values.
- Logging hooks.

10.2.4 Rewriting Incoming Requests

The need for rewriting the incoming requests arises from the fact that some NASes are very particular about the information they send with the requests. There are cases when the information they send is hardly usable or even completely unusable. For example, a Cisco AS5300 terminal server used as a voice-over IP router packs a lot of information into its `Acct-Session-Id` attribute. Though the information stored there is otherwise relevant, it makes proper accounting impossible, since the `Acct-Session-Id` attributes in the start and stop packets of the same session become different, and thus Radius cannot determine the session start to which the given session stop request corresponds (see Section 13.2.7 [Acct-Session-Id], page 177).

In order to cope with such NASes, GNU Radius is able to invoke a Rewrite function upon arrival of the packet and before processing it further. This function can transform the packet so that it obtains the form prescribed by RFCs and its further processing becomes possible.

For example, in the case of the AS5300 router, a corresponding Rewrite function parses the `Acct-Session-Id` attribute; breaks it down into fields; stores them into proper attributes, creating them if necessary; and finally replaces `Acct-Session-Id` with its real value, which is the same for the start and stop records corresponding to a single session. Thus all the information that came with the packet is preserved, but the packet itself is made usable for proper accounting.

A special attribute, `Rewrite-Function`, is used to trigger invocation of a Rewrite function. Its value is a name of the function to be invoked.

When used in a '`naslist`' profile, the attribute causes the function to be invoked when the incoming request matches the huntgroup (see Section 2.4.4 [Huntgroups], page 15). For example, to have a function `fixup` invoked for

each packet from the NAS `10.10.10.11`, the following huntgroup rule may be used:

```
DEFAULT  NAS-IP-Address = 11.10.10.11
         Rewrite-Function = "fixup"
```

The `Rewrite-Function` attribute may also be used in a '`hints`' rule. In this case, it will invoke the function if the request matches the rule (see Section 2.4.3 [Hints], page 14). For example, this '`hints`' rule will cause the function to be invoked for each request containing the user name starting with '`P`':

```
DEFAULT  Prefix = "P"
         Rewrite-Function = "fixup"
```

Note that in both cases the attribute can be used either in LHS or in RHS pairs of a rule.

The packet rewrite function must be declared as having no arguments and returning an integer value:

```
integer fixup()
{
}
```

The actual return value from such a function is ignored, the integer return type is just a matter of convention.

The following subsection present some examples of packet rewrite functions.

10.2.4.1 Examples of Various Rewrite Functions

The examples found in this chapter are working functions that can be used with various existing NAS types. They are taken from the '`rewrite`' file contained in distribution of GNU Radius.

1. Port rewriting for MAX Ascend terminal servers

Some MAX Ascend terminal servers pack additional information into the `NAS-Port-Id` attribute. The port number is constructed as *XYYZZ*, where $X = 1$ for digital, $X = 2$ for analog, YY is the line number (1 for first PRI/T1/E1, 2 for second, and so on), and ZZ is the channel number (on the PRI or channelized T1/E1).

The following rewrite functions are intended to compute the integer port number in the range (1 .. *portcnt*), where *portcnt* represents the real number of physical ports available on the NAS. Such a port number can be used, for example, to create a dynamic pool of IP addresses (see Section 13.1.8 [Framed-IP-Address], page 168).

```
/*
 * decode MAX port number
 * input: P        --  The value of NAS-Port-Id attribute
 *        portcnt  --  number of physical ports on the NAS
 */
integer
max_decode_port(integer P, integer portcnt)
{
    if (P > 9999) {
        integer s, l, c;

        s = P / 10000;
        l = (P - (10000 * s))/100;
        c = P - ((10000 * s) + (100 * l));
        return (c-1) + (l-1) * portcnt;
    }
    return P;
}

/*
 * Interface function for MAX terminal server with 23 ports.
 * Note that it saves the received NAS-Port-Id attribute in
 * the Orig-NAS-Port-Id attribute. The latter must be
 * defined somewhere in the dictionary
 */
integer
max_fixup()
{
    %[Orig-NAS-Port-Id] = %[NAS-Port-Id];
                                # Preserve original data
    %[NAS-Port-Id] = max_decode_port(%[NAS-Port-Id], 23);
    return 0;
}
```

2. Session ID parsing for Cisco AS 5300 series

Cisco VOIP IOS encodes a lot of other information into its `Acct-Session-Id`. The pieces of information are separated by '`/`' characters. The part of `Acct-Session-Id` up to the first '`/`' character is the actual session ID.

On the other hand, its accounting packets lack `NAS-Port-Id`, though they may contain the vendor-specific pair with code 2 (vendor PEC 9), which is a string in the form '`ISDN 9:D:999`' ('`9`' represents any decimal digit). The number after the last '`:`' character can be used as a port number.

The following code parses `Acct-Session-Id` attribute and stores the information it contains in various other attributes, generates a normal `Acct-Session-Id`, and attempts to generate a `NAS-Port-Id` attribute.

```
/*
 * The port rewriting function for Cisco AS5300 used for
 * VoIP. This function is used to generate NAS-Port-Id pair
 * on the basis of vendor-specific pair 2. If the latter is
 * in the form "ISDN 9:D:999" (where each 9 represents a
 * decimal digit), then the function returns the number
 * after the last colon. This is used as a port number.
 */
integer
cisco_pid(string A)
{
    if (A =~
        ".*\([0-9][0-9]*\):
         [A-Z0-9][A-Z0-9]*:\([0-9][0-9]*\)") {
        return (integer)\2;
    }
    return -1;
}

/*
 * This function parses the packed session id.
 * The actual sid is the number before the first slash
 * character.  Other possibly relevant fields are also
 * parsed out and saved in the Voip-* A/V pairs. The latter
 * should be defined somewhere in the dictionary.
 * Note that the regular expression in this example
 * spans several lines for readability. It should be on one
 * line in real file.
 */
string
cisco_sid(string S)
{
   if (S =~ "\(.[^/]*\)/[^/]*/[^/]*/\([^/]*\)/\([^/]*\)/
             \([^/]*\)/\([^/]*\)/\([^/]*\)/\([^/]*\)
             /\([^/]*\).*") {
        %[Voip-Connection-ID] = \2;
        %[Voip-Call-Leg-Type] = \3;
        %[Voip-Connection-Type] = \4;
        %[Voip-Connect-Time] = \5;
        %[Voip-Disconnect-Time] = \6;
        %[Voip-Disconnect-Cause] = \7;
        %[Voip-Remote-IP] = \8;
        return \1;
   }
   return S;
}
```

```
/*
 * Normalize cisco AS5300 packets
 */
integer
cisco_fixup()
{
    integer pid;

    if ((pid = cisco_pid(%[Cisco-PRI-Circuit])) != -1) {
        if (*%[NAS-Port-Id])
            %[Orig-NAS-Port-Id] = %[NAS-Port-Id];
        %[NAS-Port-Id] = pid;
    }
    if (*%[Acct-Session-Id]) {
        %[Orig-Acct-Session-Id] = %[Acct-Session-Id];
        %[Acct-Session-Id] = cisco_sid(%[Acct-Session-Id]);
    }
    return 0;
}
```

3. User-name rewriting for NT machines

Users coming from Windows NT machines often authenticate themselves as '`NT_DOMAIN\`*`username`*'. The following function selects the user-name part and stores it in the `User-Name` attribute:

```
integer
login_nt(string uname)
{
    integer i;

    if ((i = index(uname, '\\')) != -1)
        return substr(uname, i+1, -1);
    return uname;
}

integer
nt_rewrite()
{
    %[Orig-User-Name] = %[User-Name];
    %[User-Name] = login_nt(%[User-Name]);
    return 0;
}
```

10.2.5 Login Verification Functions

A login verification function is invoked to process the output from the NAS. This process is described in Section 6.9 [Multiple Login Checking], page 74. The function to be invoked for given NAS is defined by a `function` flag in the '`raddb/nastypes`' or '`raddb/naslist`' file (see Section 4.5 [nastypes file], page 47). It must be defined as follows:

`integer check` (*string* ***str***, *string* ***name***, *integer* ***pid***, *string* ***sid***) [Function Template]

Its arguments are:

str Input string. If the query method is `finger`, this is the string of output received from the NAS with trailing newline stripped off. If the query method is `snmp`, it is the received variable value converted to its string representation.

name User name.

pid Port ID of the session.

sid Session ID.

The function should return non-0 if its arguments match the user's session, and 0 otherwise.

10.2.5.1 Examples of Login Verification Functions

As an example, let's consider the function for analyzing a line of output from a standard UNIX `finger` service. In each line of `finger` output the first field contains the user name; the third field, the The function must return 1 if the three fields match the input user name and port and session IDs:

```
integer
check_unix(string str, string name, integer pid, string sid)
{
    return field(str, 1) == name
           && field(str, 3) == pid
           && field(str, 7) == sid;
}
```

The next example is a function to analyze a line of output from an SNMP query returning a user name. This function must return 1 if the entire input line matches the user name:

```
integer
check_username(string str, string name, integer pid, string sid)
{
    return str == name;
}
```

10.2.6 Attribute Creation Functions

These are the functions used to create Radius reply attributes. An attribute creation function can take any number of arguments. The type of its return is determined by the type of Radius attribute the value will be assigned to. To invoke the function, write its name in the A/V pair of the RHS in the '`raddb/users`' file, e.g.:

```
DEFAULT Auth-Type = SQL
        Service-Type = Framed-User,
            Framed-IP-Address = "=get_ip_addr(10.10.10.1)"
```

The function `get_ip_addr` will be invoked after successful authentication and it will be passed the IP `10.10.10.1` as its argument. An example of a useful function that can be invoked this way is

```
integer
get_ip_address(integer base)
{
    return base + %[NAS-Port-Id] - %[NAS-Port-Id]/16;
}
```

10.2.7 Logging Hook Functions

A logging hook functions should be declared as follows:

`string hook` (*integer* ***reqtype***, *string* ***nasid***, *integer* ***reqid***) [Function Template]

reqtype Type of the request. It can be converted to string using `request_code_string` function (see Section 10.2.8.7 [Built-in Functions], page 111).

nasid NAS identifier from '`raddb/naslist`', or its host name if not declared there

reqid Request identifier.

Notice that the hook function *shall not* produce any side effects, in particular it shall not modify the incoming request in any way.

Following is an example prefix hook function that formats the incoming request data:

```
string
compat_log_prefix(integer reqtype, string nas, integer id)
{
        string result;

        return "(" + request_code_string(reqtype) + " "
                   + nas + " " + (string)id + " " + %[User-Name] + ")";
}
```

Here is a sample log produced by `radiusd` before and after enabling this function:

```
Auth.notice: Login OK [jsmith]
...
Auth.notice: (AUTHREQ nas-2 251 jsmith): Login OK [jsmith]
```

10.2.8 Full Syntax Description

10.2.8.1 Rewrite Data Types

There are only two data types: `integer` and `string`, the two being coercible to each other in the sense that a string can be coerced to an integer if it

contains a valid ASCII representation of a decimal, octal, or hex number, and an integer can always be coerced to a string, the result of such coercion being the ASCII string that is the decimal representation of the number.

10.2.8.2 Rewrite Symbols

A *symbol* is a lexical token. The following symbols are recognized:

Arithmetical operators
: These are '`+`', '`-`', '`*`', '`/`' representing the basic arithmetical operations, and '`%`' meaning remainder.

Comparison operators
: These are: '`==`', '`!=`', '`<`', '`<=`', '`>`', '`>=`' with the same meaning they have in C. Special operators are provided for regular-expression matching. The binary operator '`=~`' returns true if its left-hand-side operand matches the regular expression on its right-hand side (see Section 10.2.8.6 [Regular Expressions], page 110). '`!~`' returns true if its left-hand-side operand does *not* match the regexp on its right-hand side. The right-hand-side operand of '`!~`' or '`=~`' must be a literal string, i.e., the regular expression must be known at compile time.

Unary operators
: The unary operators are '`-`' and '`+`' for unary plus and minus, '`!`' for boolean negation, and '`*`' for testing for the existence of an attribute.

Boolean operators
: These are '`&&`' and '`||`'.

Parentheses '`(`' and '`)`'
: These are used to change the precedence of operators, to introduce type casts (type coercions), to declare functions, and to pass actual arguments to functions.

Curly braces ('`{`' and '`}`')
: These are used to delimit blocks of code.

Numbers
: Numbers follow the usual C convention for integers. A number consisting of a sequence of digits is taken to be octal if it begins with '`0`' (digit zero), and decimal otherwise. If the sequence of digits is preceded by '`0x`' or '`0X`', it is taken to be a hexadecimal integer.

IP Numbers
: IP numbers are represented by a standard numbers-and-dots notation. IP numbers do not constitute a separate data type, rather they are in all respects similar to initeger numbers.

Characters

These follow the usual C convention for characters, i.e., they consist either of an ASCII character itself or of its value, enclosed in a pair of singlequotes. The character value begins with '\' (backslash) and consists either of three octal or of two hexadecimal digits. A character does not form a special data type; it is represented internally by an integer.

Quoted strings

These follow slightly modified C conventions for strings. A string is a sequence of characters surrounded by double quotes, as in '`"..."`'. In a string, the double quote character '`"`' must be preceeded by a backslash '\'. A '\' and an immediately following newline are ignored. Following escape sequences have special meaning:

\a	Audible bell character (ASCII 7)
\b	Backspace (ASCII 8)
\e	Escape character (ASCII 27)
\f	Form feed (ASCII 12)
\n	Newline (ASCII 10)
\r	Carriage return (ASCII 13)
\t	Horizontal tab (ASCII 9)
\\	Backslash
\ooo	('`o`' represents an octal digit) A character whose ASCII value is represented by the octal number '`ooo`'.
\xHH \XHH	('`H`' represents a hex digit) A character whose ASCII value is represented by the hex number '`HH`'.
\(	Two characters '`\(`'.
\)	Two characters '`\)`'.

If the character following the backslash is not one of those specified, the backslash is ignored.

Attribute values

The incoming request is passed implicitly to functions invoked via the `Rewrite-Function` attribute. It is kept as an associative array, whose entries can be accessed using the following syntax:

'%[' *attribute-name* ']'
'%[' *attribute-name* ']' '(' *n* ')'

The first form returns the value of the attribute *attribute-name*. Here *attribute-name* should be a valid Radius dictionary name (see Section 4.2 [dictionary file], page 40).

The second form returns the value of the *n*th attribute of type *attribute-name*. The index *n* is counted from zero, so

```
%[attribute-name](0)
```

is equivalent to

```
%[attribute-name]
```

Identifiers Identifiers represent functions and variables. These are described in the next sub-subsection.

Regexp group references

A sequence of characters in the form

```
'\number'
```

refers to the contents of parenthesized group number *number* obtained as a result of the last executed '`=~`' command. The regexp group reference has always string data type. For example:

```
string
basename(string arg)
{
    if (arg =~ ".*/\(.*\)\..*")
        return \1;
    else
        return arg;
}
```

This function strips from *arg* all leading components up to the last slash character, and all trailing components after the last dot character. It returns *arg* unaltered if it does not contain slashes and dots. It is roughly analogous to the system `basename` utility.

10.2.8.3 Rewrite Identifiers

A valid identifier is a string of characters meeting the following requirements:

1. It starts with either a lower- or an uppercase letter of the Latin alphabet or either of the following symbols: '`_`', '`$`'.
2. It consists of alphanumeric characters, underscores('`_`'), and dollar signs ('`$`').

10.2.8.4 Rewrite Declarations

Function declarations

A Rewrite function is declared as follows:

```
type function-name (parameter-list)
```

where *type* specifies the return type of the function, *function-name* declares the symbolic name of the function, and *parameter-list* declares the formal

parameters to the function. It is a comma-separated list of declarations in the form

```
type parm-name
```

type being the parameter type, and *parm-name* being its symbolic name. Both *function-name* and *parm-name* should be valid identifiers.

Variable declarations

There are no global variables in Rewrite. All variables are local. The local variables are declared right after the opening curly brace (‘{’) and before any executable statements. The declaration syntax is

```
type ident_list ;
```

Here *ident_list* is either a valid Rewrite identifier or a comma-separated list of such identifiers.

Note that, unlike in C, no assignments are allowed in variable declarations.

10.2.8.5 Rewrite Statements

The Rewrite statements are: expressions, assignments, conditional statements, and return statements. A statement is terminated by a semicolon.

Expressions

An *expression* is one of the following:

- A variable identifier
- A type coercion expression
- An arithmetic expression
- A boolean expression
- An assignment
- A function call
- A delete statement

Type coercion

The type coercion is like a type cast in C. Its syntax is

```
‘(’ type ‘)’ ident
```

The result of type coercion is as follows:

type	Variable type	Resulting conversion
integer	integer	No conversion. This results in the same integer value.

integer	string	If the string value of the variable is a valid ASCII representation of the integer number (either decimal, octal, or hex), it is converted to the integer; otherwise the result of the conversion is undefined.
string	integer	The ASCII representation (in decimal) of the integer number.
string	string	No conversion. This results in the same string value.

Assignment

An assignment is

```
ident = expression ;
```

The variable *ident* is assigned the value of *expression*.

Function calls

These take the form

```
ident ( arg-list )
```

where *ident* is the identifier representing the function, and *arg-list* is a comma-separated list of expressions supplying actual arguments to the function. The number of the expressions must correspond exactly to the number of formal parameters in the function definition. The function that *ident* references can be either a compiled function or a built-in function.

'delete' statement

The '`delete`' statement is used to delete an attribute or attributes from the incoming request. Its syntax is:

```
delete attribute-name;
delete attribute-name(n);
```

The first variant deletes *all* the attributes of the given type. The second variant deletes only the *n*th occurrence of the matching attribute.

10.2.8.6 Regular Expressions

Rewrite uses POSIX regular expressions (See Section "Regular Expression Library" in *Regular Expression Library*, for the detailed description of these).

You control the exact type of regular expressions by the use of the pragmatic comment `regex`. Its syntax is as follows:

```
#pragma regex option-list
```

Option-list is a whitespace-separated list of options. Each option is one of the following words prefixed with '`+`' or '`-`':

`extended` Use POSIX extended regular expression syntax when interpreting regular expressions.

`icase` Do not differentiate case. Subsequent regular expression comparisons will be case insensitive.

`newline` Match-any-character operators don't match a newline.

A non-matching list ('`[^...]`') not containing a newline does not match a newline.

Match-beginning-of-line operator ('`^`') matches the empty string immediately after a newline.

Match-end-of-line operator ('`$`') matches the empty string immediately before a newline.

Prefixing an option with '`+`' means to enable the corresponding behavior. Prefixing it with '`-`' means to disable it. Thus, the following statement:

```
#pragma regex +extended +icase
```

will enable extended POSIX regular expressions using case-insensitive comparison.

Using the following comment:

```
#pragma regex -extended
```

will switch to the basic POSIX regular expressions.

The settings of a `regex` pragmatic comment remain in force up to the end of current source file, or to the next `regex` comment, whichever occurs first.

For compatibility with previous versions, GNU Radius uses the following defaults:

```
#pragma regex -extended -icase -newline
```

i.e. all regular expressions are treated as basic POSIX, comparison is case-sensitive.

10.2.8.7 Rewrite Built-in Functions

The following built-in functions are provided:

`integer length` (*string* `s`) [Function]

Returns the length of the string *s*.

```
length("string") ⇒ 6
```

`integer index` (*string* `s`, *integer* `c`) [Function]

Returns the index of the first occurrence of the character *c* in the string *s*. Returns −1 if no such occurrence is found.

```
index("/raddb/users", 47) ⇒ 0

index("/raddb/users", 45) ⇒ −1
```

`integer rindex` (*string* `s`, *integer* `i`) [Function]
Returns the index of the last occurrence of the character *c* in the string *s*. Returns −1 if no such occurrence is found.

```
rindex("/raddb/users", 47) ⇒ 6
```

`string substr` (*string* `s`, *integer* `start`, *integer* `length`) [Function]
Returns the substring of *s* of length at most *length* starting at position *start*.

```
substr("foo-bar-baz", 3, 5) ⇒ "-bar-"
```

All character positions in strings are counted from 0.

`string field` (*string* `buffer`, *integer* `n`) [Function]
This function regards the *buffer* argument as consisting of fields separated with any amount of whitespace. It extracts and returns the *n*th field. *n* is counted from 1.

```
field("GNU's not UNIX", 1) ⇒ "GNU's"
field("GNU's not UNIX", 2) ⇒ "not"
field("GNU's not UNIX", 3) ⇒ "UNIX"
field("GNU's not UNIX", 4) ⇒ ""
```

`integer logit` (*string* `msg`) [Function]
Outputs its argument to the Radius log channel `info`. Returns 0. For debugging purposes.

`integer inet_aton` (*string* `str`) [Function]
Converts the Internet host address *str* from the standard numbers-and-dots notation into the equivalent integer in host byte order.

```
inet_aton("127.0.0.1") ⇒ 2130706433
```

`string inet_ntoa` (*integer* `ip`) [Function]
Converts the Internet host address *ip* given in host byte order to a string in standard numbers-and-dots notation.

```
inet_ntoa(2130706433) ⇒ "127.0.0.1"
```

`integer htonl` (*integer* `n`) [Function]
Converts the integer *n*, regarded as long, from host to network byte order.

`integer ntohl` (*integer* `n`) [Function]
Converts the integer *n*, regarded as long, from network to host byte order.

`integer htons` (*integer* `n`) [Function]
Converts the integer *n*, regarded as short, from host to network byte order.

`integer ntohs` (*integer* `n`) [Function]
Converts the integer *n*, regarded as short, from network to host byte order.

`string gsub` (*string* ***regex***, *string* ***repl***, *string* ***str***) [Function]
For each substring matching the regular expression *regex* in the string *str*, substitute the string *repl*, and return the resulting string.

```
gsub("s","S","strings")
    ⇒ "StringS"
gsub("[0-9][0-9]*","N","28 or 29 days")
    ⇒ "N or N days"
gsub("[()'\"]","/","\"a\" (quoted) 'string'")
    ⇒ "/a/ /quoted/ /string/"
```

`string qprn` (*string* ***str***) [Function]
Replace all non-printable characters in string S by their corresponding hex value preceeded by a percent sign. Return the resulting string. Printable are alphabetical characters, decimal digits and dash ('-'). Other characters are considered non-printable. For example:

```
qprn("a string/value") ⇒ "a%20string%2Fvalue"
```

`string quote_string` (*string* ***str***) [Function]
Replace all non-printable characters in string *str* by their three-digit octal code prefixed with a backslash, or by their C escape notation, as appropriate. *Non-printable* characters depend on the locale settings. For example, suppose that the current locale is set to ISO-8859-1 (a so called "Latin-1" character set) and ⋆ represents a tab character. Then:

```
quote_string("François contains non⋆printable chars")
  ⇒ "Fran\347ois contains non\tprintable chars"
```

`string unquote_string` (*string* ***str***) [Function]
Replace C escape notations in string *str* with corresponding characters using current locale. For example, for ISO-8859-1 locale:

```
unquote_string("Fran\347ois") ⇒ "François"
```

`string toupper` (*string* ***str***) [Function]
Returns the copy of the string *str* with all alphabetical characters converted to upper case. For example:

```
toupper("a-string") ⇒ "A-STRING"
```

`string tolower` (*string* ***str***) [Function]
Returns the copy of the string *str* with all alphabetical characters converted to lower case. For example:

```
tolower("A-STRING") ⇒ "a-string"
```

`string request_code_string` (*integer* ***code***) [Function]
Converts integer RADIUS request code to its textual representation as per RFC 3575. This function is useful in logging hooks (see Section 4.1.2.1 [hooks], page 25).

```
request_code_string(4) ⇒ "Accounting-Request"
```

Native Language Support

The native language support is provided via the functions described below. These functions are interfaces to GNU `gettext` library. For the information about general concepts and principles of Native Language Support, please refer to Section "gettext" in *GNU* `gettext` *utilities*.

The default current textual domain is '`radius`'.

`string textdomain` (*string* ***domain***) [Function]
Sets the new value for the current textual domain. This domain is used by the functions `gettext` and `ngettext`. Returns the name of the previously used domain.

`string gettext` (*string* ***msgid***) [Function]
`string _` (*string* ***msgid***) [Function]
The function returns the translation of the string *msgid* if it is available in the current domain. If it is not available, the argument itself is returned.

The second form of this function provides a traditional shortcut notation.

For a detailed description of the GNU `gettext` interface, refer to Section "Interface to gettext" in *GNU* `gettext` *utilities*.

`string dgettext` (*string* ***domain***, *string* ***msgid***) [Function]
Returns the translation of the string *msgid* if it is available in the domain *domain*. If it is not available, the argument itself is returned.

`string ngettext` (*string* ***msgid_singular***, *string* ***msgid_plural***, *integer* ***number***) [Function]
The `ngettext` function is used to translate the messages that have singular and plural forms. The *msgid_singular* parameter must contain the singular form of the string to be converted. It is also used as the key for the search in the catalog. The `msgid_plural` parameter is the plural form. The parameter *number* is used to determine the plural form. If no message catalog is found *msgid_singular* is returned if ***number*** `== 1`, otherwise *msgid_plural*.

For a detailed description of the GNU `gettext` interface for the plural translation, refer to Section "Additional functions for plural forms" in *GNU* `gettext` *utilities*.

`string dngettext` (*string* ***domain***, *string* ***msg_sing***, *string* ***msg_plur***, *integer* ***number***) [Function]
Similar to `ngettext`, but searches translation in the given *domain*.

Request Accessors

The following functions are used to read some internal fields of a RADIUS request.

`Integer request_source_ip ()` [Function]
Returns source IP address of the currently processed request. This function can be used to add `NAS-IP-Address` attribute to the requests lacking one, e.g.:

```
integer
restore_nas_ip()
{
        if (!*%[NAS-IP-Address])
                %[NAS-IP-Address] = request_source_ip();
        return 0;
}
```

`Integer request_source_port ()` [Function]
Returns the source UDP port.

`Integer request_id ()` [Function]
Returns the request identifier.

`Integer request_code ()` [Function]
Returns the request code.

10.3 Guile

The name Guile stands for *GNU's Ubiquitous Intelligent Language for Extensions*. It provides a Scheme interpreter conforming to the R4RS language specification. This section describes use of Guile as an extension language for GNU Radius. It assumes that the reader is sufficiently familiar with the Scheme language. For information about the language, refer to Section "Top" in *Revised(4) Report on the Algorithmic Language Scheme*. For more information about Guile, see Section "Overview" in *The Guile Reference Manual*.

Scheme procedures can be called for processing both authentication and accounting requests. The invocation of a Scheme procedure for an authentication request is triggered by the `Scheme-Procedure` attribute; the invocation for an accounting request is triggered by the `Scheme-Acct-Procedure` attribute. The following sections address these issues in more detail.

10.3.1 Data Representation

A/V pair lists are the main object Scheme functions operate upon. Scheme is extremely convenient for representation of such objects. A Radius A/V pair is represented by a Scheme pair; e.g.,

```
Session-Timeout = 10
```

is represented in Guile as

```
(cons "Session-Timeout" 10)
```

The `car` of the pair can contain either the attribute dictionary name or the attribute number. Thus, the above pair may also be written in Scheme as

```
(cons 27 10)
```

(because `Session-Timeout` corresponds to attribute number 27).

Lists of A/V pairs are represented by Scheme lists. For example, the Radius pair list

```
User-Name = "jsmith",
        User-Password = "guessme",
        NAS-IP-Address = 10.10.10.1,
        NAS-Port-Id = 10
```

is written in Scheme as

```
(list
  (cons "User-Name" "jsmith")
  (cons "User-Password" "guessme")
  (cons "NAS-IP-Address" "10.10.10.1")
  (cons "NAS-Port-Id" 10))
```

10.3.2 Authentication with Scheme

The Scheme procedure used for authentication must be declared as follows:

`auth-function` *request-list check-list reply-list* [Function Template]

Its arguments are:

request-list
: The list of A/V pairs from the incoming request

check-list
: The list of A/V pairs from the LHS of the profile entry that matched the request

reply-list
: The list of A/V pairs from the RHS of the profile entry that matched the request

The function return value determines whether the authentication will succeed. The function must return either a boolean value or a pair. The return of `#t` causes authentication to succeed. The return of `#f` causes it to fail.

For a function to add something to the reply A/V pairs, it should return a pair in the form

```
(cons return-code list)
```

where *return-code* is a boolean value of the same meaning as described above. *list* is a list of A/V pairs to be added to the reply list. For example, the following function will always deny the authentication, returning an appropriate message to the user:

```
(define (decline-auth request-list check-list reply-list)
  (cons #f
        (list
         (cons "Reply-Message"
               "\r\nSorry, you are not
                allowed to log in\r\n"))))
```

As a more constructive example, let's consider a function that allows the authentication only if a user name is found in its internal database:

```
(define staff-data
  (list
   (list "scheme"
         (cons
          (list (cons "NAS-IP-Address" "127.0.0.1"))
          (list (cons "Framed-MTU" "8096")))
         (cons
          '()
          (list (cons "Framed-MTU" "256"))))))

(define (auth req check reply)
  (let* ((username (assoc "User-Name" req))
         (reqlist (assoc username req))
         (reply-list '()))
    (if username
        (let ((user-data (assoc (cdr username) staff-data)))
          (rad-log L_INFO (format #f "~A" user-data))
          (if user-data
              (call-with-current-continuation
               (lambda (xx)
                 (for-each
                  (lambda (pair)
                    (cond
                     ((avl-match? req (car pair))
                      (set! reply-list (avl-merge
                                        reply-list
                                        (cdr pair)))
                      (xx #t))))
                  (cdr user-data))
                 #f)))))
    (cons
     #t
     reply-list)))
```

To trigger the invocation of the Scheme authentication function, assign its name to the `Scheme-Procedure` attribute in the RHS of a corresponding '`raddb/users`' profile. For example:

```
DEFAULT Auth-Type = SQL
        Scheme-Procedure = "auth"
```

10.3.3 Accounting with Scheme

The Scheme accounting procedure must be declared as follows:

`acct-function-name` *request-list* [Function Template]

Its argument is:

request-list
: The list of A/V pairs from the incoming request

The function must return a boolean value. The accounting succeeds only if it has returned `#t`.

Here is an example of a Scheme accounting function. The function dumps the contents of the incoming request to a file:

```
(define radius-acct-file "/var/log/acct/radius")

(define (acct req)
  (call-with-output-file radius-acct-file
    (lambda (port)
      (for-each (lambda (pair)
                  (display (car pair) port)
                  (display "=" port)
                  (display (cdr pair) port)
                  (newline port))
                req)
      (newline port)))
  #t)
```

10.3.4 Radius-Specific Functions

`avl-delete` *av-list attr* [Scheme Function]
Delete from *av-list* the pairs with attribute *attr*.

`avl-merge` *dst src* [Scheme Function]
Merge *src* into *dst*.

`avl-match?` *target list* [Scheme Function]
Return `#t` if all pairs from *list* are present in *target*.

`rad-dict-name->attr` *name* [Scheme Function]
Return a dictionary entry for the given attribute *name* or `#f` if no such name was found in the dictionary.

A dictionary entry is a list in the form

`dict-entry` *name-string attr-number type-number vendor* [Scheme List]

where the arguments are as follows:

name-string
: The attribute name

value-number
: The attribute number

type-number
: The attribute type

vendor
: The vendor PEC, if the attribute is a vendor-specific one, or `#f` otherwise.

`rad-dict-value->name` *attr value* [Scheme Function]
Returns the dictionary name of the given *value* for an integer-type attribute *attr*, which can be either an attribute number or its dictionary name.

`rad-dict-name->value` *attr value* [Scheme Function]
Convert a symbolic attribute value name into its integer representation.

`rad-dict-pec->vendor` *pec* [Scheme Function]
Convert a PEC to the vendor name.

`rad-log-open` *prio* [Scheme Function]
Open Radius logging to the severity level *prio*.

`rad-log-close` [Scheme Function]
Close a Radius logging channel opened by a previous call to `rad-log-open`.

`rad-rewrite-execute-string` *string* [Scheme Function]
Interpret *string* as an invocation of a function in Rewrite language and execute it.

Return value: return of the corresponding Rewrite call, translated to the Scheme data type.

`rad-rewrite-execute` *arglist* [Scheme Function]
Execute a Rewrite language function. `(car arglist)` is interpreted as a name of the Rewrite function to execute, and `(cdr arglist)` as a list of arguments to be passed to it.

Return value: return of the corresponding Rewrite call, translated to the Scheme data type.

`rad-openlog` *ident option facility* [Scheme Function]
Scheme interface to the system `openlog()` call.

`rad-syslog` *prio text* [Scheme Function]
Scheme interface to the system `syslog()` call.

`rad-closelog` [Scheme Function]
Scheme interface to the system `closelog()` call.

`rad-utmp-putent` *status delay list radutmp_file radwtmp_file* [Scheme Function]
Write the supplied data into the radutmp file. If *radwtmp_file* is not nil, the constructed entry is also appended to *wtmp_file*.

list is:

`utmp-entry` *user-name orig-name port-id port-type session-id caller-id framed-ip nas-ip proto* [Scheme List]

user-name The user name

orig-name The original user name from the request

port-id The value of the `NAS-Port-Id` attribute

port-type A number or character indicating the port type

session-id	The session ID
caller-id	The value of the `Calling-Station-Id` attribute from the request
framed-ip	The framed IP assigned to the user
nas-ip	The NAS IP
proto	A number or character indicating the type of the connection

11 Utility Programs

11.1 radwho

`Radwho` displays the list of users currently logged in by the Radius server.

Default output information is made compatible with that of the standard UNIX `finger(1)` utility. For each user the following information is displayed: login name, name, connection protocol, NAS port, login date, NAS name, assigned IP or corresponding network name.

When used with '`-l`' option, the long output format is used. In this format the following information is output:

'`Login`' Login name of the user

'`SessionID`'
Unique session ID assigned by the terminal server.

'`Proto`' Connection prototype.

'`Port`' Port number

'`When`' Login date and time

'`From`' Name of the NAS that accepted the connection.

'`Location`'
Framed IP or the corresponding network name.

'`Caller`' Caller station ID ad reported by the NAS.

'`Duration`'
Duration of the session.

11.1.1 radwho Command Line Options

The following command line options can be used to modify the behavior of the program:

'`-A`'
'`--all`' Display the information about logged-out users as well. The logged-out users are shown with '`Proto`' field set to HUP.

'`-c`'
'`--calling-id`'
Display the calling station ID in the second column. Equivalent to '`--format clid`'.

'`-d NAME`'
'`--directory NAME`'
Set the Radius configuration directory name.

'`-D` *fmt*'
'`--date-format` *fmt*'
: Set the date representation. Th *fmt* is usual `strftime(3)` format string. It defaults to `%a %H:%M`, i.e. the abbreviated weekday name according to the current locale, and the hour and the minutes as two-digit decimal numbers.

'`-e` *STRING*'
'`--empty` *STRING*'
: Display any empty field as *STRING*. This is useful when the output of `radwho` is fed to some analyzing program, as it helps to keep the same number of columns on each line of output.

'`-F`'
'`--finger`'
: Start in `fingerd` mode. In this mode `radwho` emulates the behavior of the `fingerd(8)` utility. Use this option if starting `radwho` from the '`/etc/inetd.conf`' line like this[1]:

```
finger stream tcp nowait nobody /usr/sbin/radwho
radwho -fL
```

  This mode is also enabled by default if `radwho` notices that its name (`argv[0]`) is '`fingerd`' or '`in.fingerd`'.

'`-H`'
'`--no-header`'
: Don't display header line.

'`-i`'
'`--session-id`'
: Display session ID instead of GECOS in the second column. Equivalent to '`--format sid`'.

'`-I`'
'`--ip-strip-domain`'
: Display hostnames without domain part.

'`-u`'
'`--local-also`'
: Display information about local users from the system '`utmp`' file. May prove useful when running `radwho` as a finger daemon.

'`-n`'
'`--no-resolve`'
: Do not resolve IP.

[1] In this example the statement has been split on two lines to fit the page width. It must occupy a *single line* in the real configuration file.

‘`-o format`’
‘`--format format`’
: Select customized output format. This can also be changed by setting the value of environment variable `RADWHO_FORMAT`. The *format* is either a symbolic name of one of the predefined formats or a format specification (see next subsection).

‘`-s`’
‘`--secure`’
: Run in secure mode. Queries without a user name are rejected.

11.1.2 `radwho` Format Strings

A format string controls the output of every record from ‘`radutmp`’. It contains two types of objects: ordinary characters, which are simply copied to the output, and format specifications, each of which causes output of a particular piece of information from the ‘`radutmp`’ record.

Each format specification starts with an opening brace and ends with a closing brace. The first word after the brace is the name of the format specification. The rest of words are *positional arguments* followed by *keyword arguments*. Both are optional. The keyword arguments begin with a colon and must follow the positional arguments.

The full list of format specifications follows.

`newline` [*count*] [Format Spec]
: Causes the newline character to be output. If the optional *count* is supplied, that many newlines will be printed

`tab` [*num*] [Format Spec]
: Advance to the next tabstop in the output stream. If optional *num* is present, then skip *num* tabstops. Each tabstop is eight characters long.

The following specifications output particular fields of a ‘`radutmp`’ record. They all take two positional arguments: *width* and *title*.

The first argument, *width* sets the maximum output length for this specification. If the number of characters actually output is less than the width, they will be padded with whitespace either to the left or to the right, depending on the presence of the `:right` keyword argument. If the number of characters is greater than *width*, they will be truncated to fit. If *width* is not given, the exact data are output as is.

The second argument, *title*, gives the title of this column for the heading line. By default no title is output.

Every field specification accepts at least two keyword arguments. The keyword `:right` may be used to request alignment to the right for the data. This keyword is ignored if *width* is not given.

The keyword `:empty` followed by a string causes `radwho` to output that string if the resulting value for this specification would otherwise be empty.

`login` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Print the user login name.

`orig-login` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Print original login name as supplied with the request.

`gecos` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
The GECOS field from the local '`/etc/passwd`' corresponding to the login name. If the user does not have a local account, his login name is output.

`nas-port` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
NAS port number

`session-id` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
The session ID.

`nas-address` *`width title`* [*:empty* *`repl`*][*:right*][*:nodomain*] [Format Spec]
The NAS name or IP.

The `:nodomain` keyword suppresses the output of the domain part of the name, i.e., the hostname is displayed only up to the first dot.

`framed-address` *`width title`* [*:empty* *`repl`*][*:right*][*:nodomain*] [Format Spec]
Framed IP assigned to the user, if any.

The `:nodomain` keyword suppresses the output of the domain part of the name, i.e. the hostname is displayed only up to the first dot.

`protocol` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Connection protocol as reported by `Framed-Protocol` attribute. If the symbolic value is found in the dictionary file, it will be displayed. Otherwise, the numeric value will be displayed as is.

`time` *`width title`* [*:empty* *`repl`*][*:right*][*:format* *`date-format`*] [Format Spec]
Date and time when the session started.

The `:format` keyword introduces the `strftime` format string to be used when converting the date for printing. The default value is `%a %H:%M`.

`duration` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Total time of the session duration.

`delay` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Delay time (see Section 13.2.2 [Acct-Delay-Time], page 175).

`port-type` *`width title`* [*:empty* *`repl`*][*:right*] [Format Spec]
Port type as reported by the value of the `NAS-Port-Type` attribute. If the symbolic value is found in the dictionary file, it will be displayed. Otherwise, the numeric value will be displayed as is.

`clid` *`width title`* [*:empty* ***`repl`***][*:right*] [Format Spec]
The calling station ID.

`realm` *`width title`* [*:empty* ***`repl`***][*:right*][*:nodomain*] [Format Spec]
If the request was forwarded to a realm server, print the symbolic name of the realm from the '`raddb/realms`' file. If no symbolic name is found, print the remote server IP or hostname. In the latter case, the `:nodomain` keyword may be used to suppress the output of the domain part of the name, i.e. to display the hostname only up to the first dot.

11.1.3 `radwho` Predefined Formats

The predefined formats are:

'`default`' Default output format. Each record occupies one line. The fields output are: login name, GECOS name, connection protocol, port number, time when the connection was initiated, NAS IP, and assigned framed IP. This corresponds to the following format specification (split in several lines for readability:

```
(login 10 Login) (gecos 17 Name) \
(protocol 5 Proto) (nas-port 5 TTY) \
(time 9 When) (nas-address 9 From) \
(framed-address 16 Location)
```

'`sid`' The same as '`default`', except that the session ID is output in the second column.

'`clid`' The same as '`default`', except that the calling station ID is output in the second column.

'`long`' Outputs all information from each '`radutmp`' record. It is equivalent to specifying the following format string:

```
(login 32 Login) (session-id 32 SID) \
(protocol 5 Proto) (nas-port 5 Port) \
(time 27 Date) (nas-address 32 NAS) \
(clid 17 CLID) (duration 7 Duration) \
(framed-address 16 Location) (realm 16 Realm)
```

'`gnu`' Each '`radutmp`' record is represented as a table. It is equivalent to specifying the following format string:

```
User: (login)(newline)\
In real life: (gecos)(newline)\
Logged in: (time)(newline)\
NAS: (nas-address)(newline)\
Port: (nas-port)(newline)\
CLID: (clid)(newline)\
Protocol: (protocol)(newline)\
Session ID: (session-id)(newline)\
Uptime: (duration)(newline)\
Assigned IP: (framed-address)(newline)\
Realm: (realm)(newline)"
```

11.2 radlast

The `radlast` utility lists sessions of specified users, NASes, NAS ports, and hosts, in reverse time order. By default, each line of output contains the login name, the NAS short name and port number from where the session was conducted, the host IP or name, the start and stop times for the session, and the duration of the session. If the session is still continuing, `radlast` will so indicate.

When the '`-l`' option is specified, `radlast` produces long output. It includes following fields:

- Login name
- NAS short name
- Port number
- Connection protocol
- Port type
- Session ID
- Caller ID
- Framed IP address
- Session Start Time
- Session Stop Time
- Duration of the Session

11.2.1 radlast Command Line Options

Use following command line options to control the behavior of the `radlast` utility:

'`-`*`number`*'
'`-c` *`number`*'
'`--count` *`number`*'
: When given this option, `radlast` will output at most this many lines of information.

'`-f`'
'`--file` *`name`*'
: Read the specified file instead of the default '`/var/log/radwtmp`'.

'`-h` *`hostname`*'
'`--host` *`hostname`*'
: Report the logins from given host. Host can be either a name or a dotted-quad Internet address.

'`-n` *`shortname`*'
'`--nas` *`shortname`*'
: Report the logins from the given NAS.

‘`-l`’
‘`--long-format`’
: Long output format. Report all the information stored in ‘`radwtmp`’ file.

‘`-p` *number*’
‘`--port` *number*’
: Report the logins on a given port. The port may be specified either fully or abbreviated, e.g. `radlast -p S03` or `radlast -p 3`.

‘`-s`’
‘`--show-seconds`’
: Report the duration of the login session in seconds instead of the default days, hours, and minutes.

‘`-t`’
: The same as ‘`-p`’. This flag is provided for compatibility with `last(1)`.

‘`-w`’
‘`--wide`’
: Widen the duration field to show seconds as well as the default days, hours and minutes.

If multiple arguments are given, the logical OR operation between them is assumed, i.e., the information selected by each argument is printed. This, however, does not apply to the ‘`-c`’ option. That option is always combined with the rest of command line by logical AND.

The pseudo-user ‘`~reboot`’ logs in on every reboot of the network access server.

If `radlast` is interrupted, it indicates to what date the search had progressed.

11.3 radzap

`radzap` searches the Radius accounting database for matching login records and closes them.

At least one of the options ‘`-n`’, ‘`-p`’, or the user name must be specified. If they are used in conjunction, they are taken as if joined by the logical AND operation.

`radzap` operates in two modes: silent and confirm. The silent mode is enabled by default. When run in this mode, `radzap` deletes every record that matches the search conditions given.

In confirm mode `radzap` will ask for a confirmation before zapping each matching record. Any line beginning with a ‘`y`’ is taken as a positive response; any other line is taken as a negative response.

The confirm mode is toggled by the command line option ‘`-c`’.

Syntax

```
radzap [options] [username]
```

Options are:

'`-c`'
'`--confirm`'
: Enable confirm mode.

'`-d` *dir*'
'`--directory` *dir*'
: Specify alternate configuration directory. Default is '`/usr/local/etc/raddb`'.

'`-f` *file*'
'`--file` *file*'
: Operate on *file* instead of the default '*RADLOG*`/radutmp`'.

'`-l` *dir*'
'`--log-directory` *dir*'
: Search the file '`radutmp`' in the given directory.

  This option is deprecated. It is currently retained for backward compatibility with previous versions.

'`-q`'
'`--quiet`' Disable confirm mode.

'`-h`'
'`--help`' Display a short help summary, and exit.

'`-n` *name*'
'`--nas` *name*'
: Specify NAS name to zap user from.

'`-p` *port*'
'`--port` *port*'
: Specify the port number of the session to be zapped. The port number can be specified either in its full form, e.g. `radzap -p S02`, or in its short form, e.g. `radzap -p 2`.

11.4 radgrep

This utility allows one to quickly look up the user in the Radius accounting database, using a regular expression match. `radgrep` scans the output of `radwho` utility and outputs only the lines that match given regular expressions.

Syntax

`radgrep` accepts two sets of options separated by '`--`' (double hyphen). The first subset is passed as the command line to the `radwho` utility. The second one is passed to `grep`.

11.5 radping

This utility is a shell program that determines the user's framed IP and runs `ping` on that address.

Syntax

```
radping username
radping -c calling-station-id
```

The second way of invoking the program allows one to use the calling station ID to indicate the user.

11.6 radauth

The `radauth` utility sends the Radius server an `Access-Request` packet and displays the result it gets. If the server responds with `Access-Accept` `radauth` can also send an `Accounting-Request` thereby initiating user's session.

The utility is a `radtest` program. See Section 12.2.12 [Sample Radtest Program], page 155, for the detailed discussion of its internals.

Invocation

```
radauth [options] [command] user-name [password]
```

Options are:

'`-v`' Print verbose descriptions of what is being done.

'`-n nas-ip`'
Set NAS IP address

'`-s sid`' Set accounting session ID

'`-P port`' Set NAS port number.

Valid commands are:

auth Send only `Access-Request`. This is the default.

acct Send `Access-Request`. If successfull, send `Accounting-Request` with `Acct-Status-Type = Start`.

start Send `Accounting-Request` with `Acct-Status-Type = Start`.

stop `Accounting-Request` with `Acct-Status-Type = Stop`.

The program determines which Radius server to use, the authentication port number, and the shared secret, following the procedure common to all client scripts (see Section 12.1 [client.conf], page 135).

11.7 radctl

`Radctl` is a control interface to the `radiusd` daemon. It allows the user running it to query `radiusd` about various aspects of its work and to issue administrative commands to it. The syntax is

```
radctl command [args]
```

where *command* is a command telling `radctl` which actions to take, and *args* are optional arguments to the command. Only one command can be specified per invocation.

The valid commands are as follows:

`start [args]`
: If `radiusd` is not running already, it is started. When present, *args* are passed as the command line to the server.

`stop`
: Stops running `radiusd`.

`restart [args]`
: Stops the server and then starts it again. When present, *args* are passed as the command line to the server.

`reload`
: Causes the running `radiusd` server to reread its configuration files.

`dumpdb`
: Tells `radiusd` to dump its user hash table into the file '*radlog*`/radius.parse`'. This can be used for debugging configuration files.

`status`
`which`
: This command shows the line of `ps(1)` describing the running copy of `radiusd` program. The exact look depends on the version of operating system you are running. Please refer to "man ps" for more detail on `ps` output.

  Here is an example of what `radctl status` prints on `GNU/Linux`:

  ```
  19692 ?        01:53:11 radiusd
  ```

  Here, first field is the PID of the process, second field ('`?`') indicates that the running program has detached from the controlling terminal, the third field gives total amount of CPU time used by the program, and, finally, the last field shows the full name under which the command was invoked.

11.8 builddbm

Usage

`builddbm` converts the plaintext Radius users database into DBM files. Some versions of the Radius daemon have used this to speed up the access to the users database. However, with GNU Radius things go the other way around. The server reads the entire plaintext database, converts it into internal form,

and stores into a hash table, which provides for fast access. Actually, using a DBM version of the users database slows down the access unless the machine that runs the Radius daemon is short of address space for the daemon to store the users database.

Syntax

When used without arguments, the `builddbm` utility attempts to convert the file '`raddb/users`' to '`raddb/users.db`' or to the pair '`raddb/users.pag`', '`raddb/users.dir`', depending on the version of the DBM library used.

If used with one argument, that argument is taken as the name of the plaintext database file to operate upon.

Use the following command line options to modify the operation of `buildbm`:

'`-d` *dir*' Specifies alternate directory for the Radius configuration files. This defaults to '`/usr/local/etc/raddb`'.

'`-h`' Outputs short usage summary and exits with 0 exit code.

11.9 `radscm`: A Guile Interface to Radius Functions

`radscm` is a Scheme interpreter based on Guile with the addition of special functions and variables for communicating with `radiusd`. This chapter concentrates on the special features provided by `radscm`. Refer to Guile documentation for information about Scheme and Guile (see Section "Overview" in *The Guile Reference Manual*).

Variables

`%raddb-path` [Variable]

A path to the Radius configuration directory.

`rad-server-list` [Function]

A list of radius servers. Each element of the list is:

```
(list id-str host-str secret-str auth-num acct-num
      cntl-num)
```

where the arguments are as follows:

id-str	Server ID
host-str	Server hostname or IP
secret-str	Shared secret key to use
auth-num	Authentication port number
acct-num	Accounting port number
cntl-num	Control channel port number

Thus, each entry can be used as an argument to `rad-client-set-server` or `rad-client-add-server`.

Functions

`rad-send-internal` *port-number code-number pair-list* [Function]

Sends the request to currently selected server. Arguments are:

port-number
: Port number to use. These values are allowed:

0	Authentication port
1	Accounting port
2	Control port

The actual port numbers are those configured for the given server.

code-number
: Request code.

pair-list
: List of attribute-value pairs. Each pair is either

```
(cons attr-name-str value)
```

or

```
(cons attr-number value)
```

Return: On success,

```
(list return-code-number pair-list)
```

On failure,

```
'()
```

`rad-send` *port-number code-number pair-list* . *verbose* [Function]

Sends a radius request. Actually it does the same work as `rad-send-internal`, but if *verbose* is specified, the verbose report about interaction with the radius server is printed.

`rad-client-list-servers` [Function]

List currently configured servers. Two columns for each server are displayed: server ID and IP.

`rad-get-server` [Function]

Returns the ID of the currently selected server.

`rad-client-set-server` *list* [Function]

Selects for use the server described by *list*. Here *list* takes the form

```
(list id-str host-str secret-str auth-num acct-num
      cntl-num)
```

where the elements are as follows:

id-str	Server ID
host-str	Server hostname or IP
secret-str	Shared secret key to use

auth-num	Authentication port number
acct-num	Accounting port number
cntl-num	Control channel port number

`rad-client-add-server` *list* [Function]
Adds the server described by *list* to the list of active servers. Here *list* takes the form

```
(list id-str host-str secret-str auth-num acct-num
      cntl-num)
```

where the elements are as follows:

id-str	Server ID
host-str	Server hostname or IP
secret-str	Shared secret key to use
auth-num	Authentication port number
acct-num	Accounting port number
cntl-num	Control channel port number

`rad-read-no-echo` *prompt-str* [Function]
Prints the given *prompt-str*, disables echoing, reads a string up to the next newline character, restores echoing, and returns the string entered. This is the interface to the C `getpass(3)` function.

`rad-client-source-ip` *ip-str* [Function]
Sets the IP to be used as source. *ip-str* can be either an IP in dotted-quad form or a hostname.

`rad-client-timeout` *number* [Function]
Sets the timeout in seconds for waiting for a server reply.

`rad-client-retry` *number* [Function]
Sets the number of retries for sending requests to a Radius server.

`rad-format-code` *dest-bool code-number* [Function]
Format a radius reply code into a human-readable form. *dest-bool* has the same meaning as in `format` (see Section "Formatted Output" in *The Guile Reference Manual*.)

`rad-format-pair` *dest-bool pair* [Function]
Format a radius attribute-value pair for output. *dest-bool* has the same meaning as in `format`. *pair* is either

```
(cons name-str value)
```

or

```
(cons attr-number value)
```

where *value* may be of any type appropriate for the given attribute.

`rad-print-pairs` *dest-bool pair-list* [Function]
Output the radius attribute-value pairs from *pair-list*. *dest-bool* has the same meaning as in format. *pair-list* is a list of pairs in the form

```
(cons name-str value)
```

or

```
(cons attr-number value)
```

where *value* may be of any type appropriate for the given attribute.

All `Reply-Message` pairs from the list are concatenated and displayed as one.

`rad-format-reply-msg` *pair-list* . *text* [Function]
Concatenate and print text from all `Reply-Message` pairs from *pair-list*. If *text* is specified, it is printed before the concatenated text.

`rad-list-servers` [Function]
For each server from `rad-server-list`, print its ID and hostname or IP.

`rad-select-server` *ID-STR* [Function]
Select the server identified by *id-str* as a current server. The server data are looked up in rad-server-list variable.

`rad-add-server` *id-str* [Function]
Add the server identified by *id-str* to the list of current servers. The server data are looked up in rad-server-list variable.

12 Client Package

Beside the Radius server and accompanying utilities, GNU Radius provides a set of utilities to be used as Radius clients.

The following sections describe in detail the parts of the Radius client package.

12.1 Client Configuration

All programs from the client package share the same configuration file: '`raddb/client.conf`'. The file uses simple line-oriented syntax. Empty lines are ignored; the command '`#`' introduces an end-of-line comment.

The source IP is introduced with the `source_ip` statement. Its syntax is:

```
source_ip ip-addr
```

where *ip-addr* must be the IP in dotted-quad notation.

The Radius server to send the requests to is introduced with `server` statement:

```
server name ip-addr secret auth-port acct-port
```

Its parts are:

name The server name. It is reserved for further use.

ip-addr The server IP.

secret The shared secret to be used when sending requests to this server.

auth-port The authentication port number.

acct-port The accounting port number.

If several `server` statement are present, they are tried in turn until one of them replies to the request.

The amount of time a client program waits for the reply from a server is configured using the `timeout` statement:

```
timeout number
```

If the program does not receive any response within *number* seconds, it assumes the server does not respond and either retries the transmission or tries the next available server. The number of retries is set with the `retry` statement:

```
retry number
```

The example '`raddb/client.conf`' follows:

```
server first 10.11.10.1 secret 1645 1646
server second 10.11.10.1 secret 1645 1646
source_ip 127.0.0.1
timeout 3
retry 5
```

12.2 radtest

`Radtest` is a radius client shell, providing a simple and convenient language for sending requests to RADIUS servers and analyzing their reply packets.

12.2.1 Invoking radtest

(This message will disappear, once this node revised.)

'`-a variable=value`'
'`--assign=variable=value`'
: Assign a *value* to *variable*. See Section 12.2.4.5 [Assignment Options], page 142, for a detailed discussion.

'`-f file`'
'`--file=file`'
: Read input from *file*. Stops further processing of the command line.

'`-i`'
'`--no-interactive`'
: Disable interactive mode.

'`-n`'
'`--dry-run`'
: Check the input file syntax and exit.

'`-q`'
'`--quick`' Do not read the configuration file.

'`-r number`'
'`--retry=number`'
: Set number of retries.

'`-s server`'
'`--server=server`'
: Set radius server parameters.

'`-t number`'
'`--timeout=number`'
: Set timeout

'`-v`'
'`--verbose`'
: Verbose mode

'`-x debugspec`'
'`--debug=debugspec`'
: Set debugging level

'`-d dir`'
'`--directory dir`'
: Specify alternate configuration directory. Default is '`/usr/local/etc/raddb`'.

‘`-L`’
‘`--license`’
: Print license and exit.

‘`-?`’
‘`--help`’ Print short usage summary

‘`--usage`’ Print even shorter usage summary.

‘`-V`’
‘`--version`’
: Print program version.

12.2.2 Literal Values

There are four basic data types in `radtest` language: `integer`, `ipaddr`, `string` and `avlist`.

12.2.2.1 Numeric Values

Integer means a signed integer value in the range -2147483648..2147483647.

Ipaddr is an unsigned integer value suitable for representing IPv4 addresses. These can be input either as decimal numbers or as IP addresss in usual “dotted-quad” notation.

As a convenience measure, RADIUS request code names can be used in integer context. The following table lists currently defined request names with their integer codes:

Access-Request	1
Access-Accept	2
Access-Reject	3
Accounting-Request	4
Accounting-Response	5
Accounting-Status	6
Password-Request	7
Password-Ack	8
Password-Reject	9
Accounting-Message	10
Access-Challenge	11
Status-Server	12
Status-Client	13
Ascend-Terminate-Session	31
Ascend-Event-Request	33
Ascend-Event-Response	34
Ascend-Allocate-IP	51
Ascend-Release-IP	52

12.2.2.2 Character Strings

String is an arbitrary string of characters. Any input token consisting of letters of Latin alphabet, decimal digits, underscores dashes and dots and

starting with a Latin alphabet letter or underscores is considered a string. To input strings containing other letters, surround them by double quotes. The following are valid strings:

```
A-string
"String, containing white space"
```

The double quote character '"' must be preceeded by a backslash '\' if it is part of a string:

```
"Always quote \" character"
```

Generally speaking, '\' is an *escape character*, that alters the meaning of the immediately following character. If it is located at the end of the line, it allows to input newline character to strings:

```
"This string contains a \
newline character."
```

Other special escape sequences are:

\a Audible bell character (ASCII 7)

\b Backspace (ASCII 8)

\e Escape character (ASCII 27)

\f Form feed (ASCII 12)

\n Newline (ASCII 10)

\r Carriage return (ASCII 13)

\t Horizontal tab (ASCII 9)

\\ Backslash

\ooo ('o' represents an octal digit) A character whose ASCII value is represented by the octal number 'ooo'.

\xHH
\XHH ('H' represents a hex digit) A character whose ASCII value is represented by the hex number 'HH'.

If the character following the backslash is not one of those specified, the backslash is ignored.

An important variant of `string` is a *numeric string*, or *STRNUM* for short. A numeric string is a string that can be converted to a number, for example `"+2"`. This concept is used for type conversion between `integer` and `string` values.

Another way to represent strings is using *here document* syntax. Its format is as follows:

```
<<[-]delimiter
  text
delimiter
```

Delimiter is any word you choose to delimit the text, *text* represent the text of the string. If *delimiter* is prepended by a dash, any leading tabulation

characters will be removed from *text*. This allows for natural indentation of '`here document`' constructs.

The '`here document`' construct is especially useful to represent strings containing embedded newlines, as shown in the example below:

```
print <<EOT
usage: foo [OPTIONS] [NAME...]
OPTIONS are:
  -h            Print this help list.
EOT
```

12.2.2.3 Lists of A/V pairs

Avlist are whitespace or comma-separated lists of RADIUS attribute-value pairs. A syntax for A/V pair is

```
name op value
```

where *name* is attribute name, *op* is a comparison operator ('`=`', '`!=`', '`<`', '`<=`', '`>`', '`>=`'), and *value* is any valid `radtest` data or expression. An A/V pair list must be enclosed in parentheses. This is an example of an A/V pair list consisting of two pairs:

```
( User-Name = "test" NAS-IP-Address = 10.10.10.1 )
```

An empty pair list is represented by a pair of parentheses: `()`.

12.2.3 Reserved Keywords

The following keywords are reserved in `radtest`:

```
acct, and, auth, begin, break, case, continue,
do, else, end, exit, expect, getopt, if,
in, input, not, or, print, return, send,
set, shift, while
```

The reserved keywords may be used as variable names, provided that the following requrements are met:

- In assignment, these names are quoted using single quotes.

  ```
  'case' = 1
  ```

- When dereferencing, the use of curly braces is obligatory:

  ```
  ${case} + 2
  ```

12.2.4 Variables

Variables are means of storing data values at one point of your program for using them in another parts of it. Variables can be assigned either in the program itself, or from the `radtest` command line.

12.2.4.1 Using Variables

The name of a variable must be a sequence of letters, digits, underscores and dashes, but it may not begin with a digit or dash. Notice, that in contrast to the majority of programming languages, use of dashes (minus signs) is allowed in user names. This is because traditionally RADIUS attribute names

contain dashes, so extending this practice to variable names makes `radtest` programs more consistent. On the other hand, this means that you should be careful when using minus sign as a subtraction operator (see [minus-ambiguity], page 141). Case is significant in variable names: `a` and `A` are different variables.

A name of a variable may coincide with one of `radtest` reserved keywords. See Section 12.2.3 [Reserved Keywords], page 139, for description on how to use such variables.

A few variables have special built-in meanings (see Section 12.2.4.6 [Built-in Variables], page 143). Such variables can be assigned and accessed just as any other ones. All built-in variables names are entirely upper-case.

Variables are never declared, they spring into existence when an assignment is made to them. The type of a variable is determined by the type of the value assigned to it.

12.2.4.2 Variable Assignments

An *assignment* stores a new value into a variable. It's syntax is quite straightforward:

```
variable = expression
```

As a result of the assignment, the *expression* is evaluated and its value is assigned to *variable*. If *variable* did not exist before the assignment, it is created. Otherwise, whatever old value it had before the assignment is forgotten.

It is important to notice that variables do *not* have permanent types. The type of a variable is the type of whatever value it currently holds. For example:

```
foo = 1
print $foo ⇒ 1
foo = "bar"
print $foo ⇒ bar
foo = ( User-Name = "antonius" NAS-IP-Address = 127.0.0.1 )
print $foo ⇒ ( User-Name = "antonius" NAS-IP-Address = 127.0.0.1 )
```

Another important point is that in `radtest`, assignment is not an expression, as it is in many other programming languages. So `C` programmers should resist temptation to use assignments in expressions. The following is *not* correct:

```
x = y = 1
```

Finally, if the variable name coincides with one of `radtest` keywords, it must be enclosed in single quotes:

```
'case' = 1
```

12.2.4.3 Dereferencing Variables

Dereferencing a variable means accessing its value. The simplest form of dereferencing is by prepending a dollar sign to the variable name:

```
foo = 1
print foo ⇒ foo
print $foo ⇒ 1
```

Notice, that in the example above, the first `print` statement understands `foo` as a literal string, whereas the second one prints the value of the variable.

Dereferencing an undefined variable produces error message:

```
print $x error  variable 'x' used before definition
```

Optionally, the variable name may be surrounded by curly braces. Both `$foo` and `${foo}` are equivalent. The use of the latter form is obligatory only when the variable name coincides with one of the reserved keywords (see Section 12.2.3 [Reserved Keywords], page 139). It also can be used to resolve ambiguity between using dash as a part of user name and as a subtraction operator:

```
long-name = 2
$long-name ⇒ 2
$long-name-1 error  variable 'long-name-1' used before definition
${long-name}-1 ⇒ 1
$long-name - 1 ⇒ 1
```

We recommend to always surround '`-`' with whitespace when it is used as arithmetic operator.

The `${}` notation also permits some operations similar to shell variable substitution.

${variable:-text}
: Use default values. If *variable* is unset, return *text*, otherwise return the value of the *variable*.

```
$x error  variable 'x' used before definition
${x:-1} ⇒ 1
x = 2
${x:-1} ⇒ 2
```

${variable:=text}
: Assign default values. If *variable* is unset, *text* is assigned to it. The expression always returns the value of the variable.

```
$x error  variable 'x' used before definition
${x:=1} ⇒ 1
$x ⇒ 1
```

${variable:?text}
: Display error if unset. If *variable* is unset, *text* is written to the standard error (if *text* is empty, the default diagnostic message is used) and further execution of the program is aborted. Otherwise, the value of *variable* is returned.

```
$x error  variable 'x' used before definition
${x:?} error  x: variable unset
${x:?foobar} error  foobar
```

${variable::text}
: Prompt for the value if unset. If *variable* is unset, `radtest` prints *text* (or a default message, if it is empty), reads the standard input up to the newline character and returns the value read. Otherwise, the value of the variable is returned. This notation provides a convenient way for asking user to supply default values.

```
${x::} ⊣ (<teletype>:1)x?
${x::Enter value of x: } ⊣ Enter value of x:
```

${variable:&text}
: Prompt for the value with echo turned off if unset. This is similar to the `${variable::text}`, with the exception that the input value will not be echoed on the screen. This notation provides a convenient way for asking user to supply default values for variables (such as passwords, shared secrets, etc.) while preventing them from being compromised.

12.2.4.4 Accessing Elements of A/V Pair Lists

Elements of an `avlist` are accessed as if it were an array, i.e.:

```
$variable [ attribute-name ]
```

If the attribute *attribute-name* is of `string` data type and *variable* may contain more than one pair with this attribute, adding an asterisk after *attribute-name* returns concatenated values of all such pairs:

```
$variable [ attribute-name * ]
```

Examples:

```
x = (NAS-Port-Id = 127.0.0.1 \
     Reply-Message = "a long"
     Reply-Message = " string"

$x[NAS-Port-Id] ⇒ 127.0.0.1
$x[Reply-Message] ⇒ "a long"
$x[Reply-Message*] ⇒ "a long string"
```

12.2.4.5 Assignment Options

You can set any `radtest` variable from the command line. There are two ways of doing so.

First, you can use *variable assignment option* '`--assign`' (or '`-a`'). Its syntax is:

```
--assign variable=text
-a variable=text
```

For example:

```
radtest -a foobar=5
```

Another way is useful when you load a `radtest` program by '`--file`' or '`-f`'. This second way consists in including a variable assignment in the form

```
variable=text
```

in the command line after the script name. For example:

```
radtest -f myprog.rad foo=5 addr=127.0.0.1
```

This method is especially useful for executable scripts that are run using `#!` shell magic. Consider a simple script:

```
#! /usr/local/bin/radtest -f
print $addr
```

The value of *addr* can be given to the script from the command line as in the example below:

```
myprog.rad addr=127.0.0.1
```

12.2.4.6 Built-in Variables

The following variables are predefined:

_ (an underscore character)
: Contains the result of last evaluated expression.

REPLY_CODE
: Contains the last reply code received from the RADIUS server (`integer`).

REPLY
: Contains the A/V pairs lastly received from the RADIUS server (`avlist`).

SOURCEIP
: Contains the source IP address of the RADIUS client (`ipaddr`). By default, it equals the IP address set via `source_ip` statement in your '`client.conf`' file (see Section 12.1 [client.conf], page 135).

INPUT
: The value of the input read by `input` statement (see Section 12.2.11 [Built-in Primitives], page 153).

OPTVAR
: The option obtained by the recent call to `getopt` (see Section 12.2.11 [Built-in Primitives], page 153).

OPTARG
: Argument to the option obtained by the recent call to `getopt`.

OPTIND
: Index of the next command line argument to be processed by `getopt`. If the last call to `getopt` returned false, `OPTIND` contains index of the first non-optional argument in the command line.

12.2.5 Positional Parameters

Normally `radtest` stops parsing its command line when it encounters either first non-optional argument (i.e. the one not starting with dash), or an argument consisting of two dashes. The rest of the command line starting from the first non-optional argument forms *positional parameters*. These parameters are said to form the *top-level environment*.

Similarly, when invoking a user-defined function (see Section 12.2.7 [Function Definitions], page 149), arguments passed to it are said to form the

current environment of the function. These arguments are positional parameters for this function.

Positional parameters are assigned numbers starting from 1. To access (*dereference*) a positional parameter, the syntax `$n` is used, where *n* is the number of the parameter. Alternative forms, such as `${n}` or `${n:-text}`, can also be used. These work exactly as described in Section 12.2.4.3 [Dereferencing Variables], page 140).

The number of positional parameters can be accessed using a special notation `$#`.

Several things need to be mentioned:

- All top-level positional parameters have `string` data type, whereas the types of positional parameters in a function current environment are determined before inoking the function.
- Special notion `$0` returns the name of the function being evaluated. When used in the top-level environment, it returns the name of `radtest` program as given by '`--file`' ('`-f`') option.
- Dereferencing non-existing parameter returns empty string. This differs from dereferencing non-existing variable, which results in error.
- `AWK` programmers should note that assignments (see Section 12.2.4.5 [Assignment Options], page 142) are not included in the top level environment (see example below).

For example, suppose you run:

```
radtest -f script.rad name foo=bar 5
```

Then, the top-level environment of program '`script.rad`' consists of the following variables:

```
$0 ⇒ script.rad
$1 ⇒ name
$2 ⇒ 5
```

12.2.6 Expressions

An *expression* evaluates to a value, which can be printed, assigned to a variable, used in a conditional statement or passed to a function. As in other languages, expressions in `radtest` include literals, variable and positional parameter dereferences, function calls and combinations of these with various operators.

12.2.6.1 Arithmetic Operations

`Radtest` provides the common arithmetic operators, which follow normal precedence rules (see Section 12.2.6.8 [Precedence], page 148), and work as you would expect them to. The only notable exception is subtraction operator (minus) which can be used as part of a variable or attribute name, and therefore expressions like `$x-3` are ambiguous. This expression can be thought of either as a dereference of the variable `x-3` (see Section 12.2.4.3

[Dereferencing Variables], page 140), or as subtraction of the value `3` from the value of the variable `x`. `Radtest` always resolves this ambiguityin the favor of variable dereference. Therefore we advise you to always surround minus sign by whitespace, if it is used as a subtraction operator. So, instead of `$x-3`, write `$x - 3`. For other methods of solving this ambiguity, See [minus-ambiguity], page 141.

This table lists the arithmetic operators in order from highest precedence to lowest:

`- x` Negation.

`+ x` Unary plus. This is equivalent to `x`.

`x * y` Multiplication.

`x / y` Division.

`x % y` Remainder.

`x + y` Addition.

`x - y` Subtraction.

Unary plus and minus have the same precedence, the multiplication, division and remainder all have the same precedence, and addition and subtraction have the same precedence.

If *x* and *y* are of different data types, their values are first coerced to a common data type, selected using a set of rules (see Section 12.2.6.6 [Conversion Between Data Types], page 147).

12.2.6.2 String Operations

There is only one string operation: concatenation. It is represented by plus sign, e.g.:

```
"string" + "ent" ⇒ "stringent"
```

12.2.6.3 Operations on A/V Lists

(This message will disappear, once this node revised.)

The following operations are defined on A/V lists:

`x + y` *Addition.* The A/V pairs from *y* are added to *x*, honoring the respective pairs additivity (see Section 2.1 [additivity], page 7). For example:

```
( User-Name = "foo" ) + ( Password = "bar" )
⇒ ( User-Name = "foo" Password = "bar" )

( User-Name = "foo" Service-Type = Login-User ) + \
 ( Service-Type = Framed-User Password = "bar" )
⇒ ( User-Name = "foo" \
          Service-Type = Framed-User \
          Password = "bar" )
```

`x - y` *Subtraction.* The result of this operation is an A/V list consisting of pairs from *x*, which are not found in *y*.

```
( User-Name = "foo" Service-Type = Login-User ) - \
( Service-Type = Framed-User )
⇒ ( User-Name = "foo" )
```

Notice, that only attribute name matters, its value is ignored.

`x % y` *Intersection.* The result of this operation is an A/V pair list consisting of pairs from *x* which are also present in *y*.

```
( User-Name = "foo" Service-Type = Login-User ) - \
( Service-Type = Framed-User )
⇒ ( Service-Type = Login-User )
```

12.2.6.4 Comparison Operations

Comparison expressions compare operands for relationships such as equality. They return boolean values, i.e. `true` or `false`. The comparison operations are nonassociative, i.e. they cannot be used together as in:

```
# Wrong!
1 < $x < 2
```

Use boolean operations (see Section 12.2.6.5 [Boolean Ops], page 147) to group comparisons together.

Comparison operations can only be used in conditional expressions.

This table lists all comparison operators in order from highest precedence to lowest (notice, however, the comment after it):

`x = y` True if *x* is equal to *y*. `C` and `AWK` programmers, please note *single* equal sign!

`x != y` True if *x* is not equal to *y*.

`x < y` True if *x* is less than *y*.

`x <= y` True if *x* is less than or equal to *y*.

`x > y` True if *x* is greater than *y*.

`x >= y` True if *x* is greater than or equal to *y*.

Operators `=` and `!=` have equal precedence. Operators `<`, `<=`, `>`, `>=` have equal precedence.

Most operators are defined for all `radtest` data types. However, only `=` and `!=` are defined for `avlist`s. Using any other comparison operator with `avlist`s produces error.

If *x* and *y* are of different data types, their values are first coerced to a common data type, selected using a set of rules (see Section 12.2.6.6 [Conversion Between Data Types], page 147).

12.2.6.5 Boolean Operations

A *boolean operation* is a combination of comparison expressions. Boolean operations can only be used in conditional expressions.

This table lists all comparison operators in order from highest precedence to lowest.

`not` *x*
`!` *x* — True if *x* is false.

x `and` *y* — True if both *x* and *y* are true. The subexpression *y* is evaluated only if *x* is true.

x `or` *y* — True if at least one of *x* or *y* is true. The subexpression *y* is evaluated only if *x* is false.

12.2.6.6 Conversion Between Data Types

(This message will disappear, once this node revised.)

The unary negation operand is always converted to `integer` type:

```
-(1 + 1)  ⇒ -2
-(127.0.0.1 + 2) ⇒ -2130706435
- ("1" + "1") ⇒ -11
- "text"  error  cannot convert string to integer
```

The unary `not` operand is converted using the following rules:

1. If the operand is `integer`, no conversion is performed.
2. If the operand is `STRNUM` (see [STRNUM], page 138) or `ipaddr`, it is converted to integer.
3. If the operand is `string` (but is not `STRNUM`), the result of `not` is `true` only if the operand is an empty string.
4. If the operand is `avl`, the result of `not` is `true` if the list is empty.

Examples:

```
not 0 ⇒ 1
not 10 ⇒ 0
not "23" ⇒ 0
not "0" ⇒ 1
not "text" ⇒ 0
not "" ⇒ 1
not 127.0.0.1 ⇒ 0
not 0.0.0.0 ⇒ 1
```

When operands of two different data types are used in a binary operation, one of the operands is converted (*cast*) to another operand's type according to the following rules:

1. If one of the operands is literal, `radtest` attemtps to convert another operand to the literal data type. If this attempt fails, it goes on to rule 2.

2. If one of operands is `STRNUM` (see [STRNUM], page 138) and another is of numeric data type (i.e. either `integer` or `ipaddr`), the latter is converted to string representation.
3. If one of the operands is `ipaddr` and another is `integer`, the latter is converted to `ipaddr`.
4. Otherwise, if one of the operands is string, the second operand is also converted to string.
5. Otherwise, the two operands are incompatible. `Radtest` prints appropriate diagnostics and aborts execution of the current statement.

12.2.6.7 Function Calls

A *function* is a name for a particular sequence of statements. It is defined using special definition syntax (see Section 12.2.7 [Function Definitions], page 149). Normally a function return some value. The way to use this value in an expression is with a *function call* expression, which consists of the function name followed by a comma-separated list of *arguments* in parentheses. The arguments are expressions which provide values for the function call environment (see Section 12.2.5 [Positional Parameters], page 143. When there is more than one argument, they are separated by commas. If there are no arguments, write just '`()`' after the function name. Here are some examples:

```
foo()             no arguments
bar(1)            one argument
bar(1, "string")  two arguments
```

12.2.6.8 Operator Precedence (How Operators Nest)

Operator precedence determines the order of executing operators, when different operators appear close by in one expression. For example, `*` has higher precedence than `+`; thus, `a + b * c` means to multiply `b` and `c`, and then add `a` to the product.

You can overrule the precedence of the operators by using parentheses. You can think of the precedence rules as saying where the parentheses are assumed to be if you do not write parentheses yourself. Thus the above example is equivalent to `a + (b * c)`.

When operators of equal precedence are used together, the leftmost operator groups first. Thus, `a - b + c` groups as `(a - b) + c`.

This table lists `radtest` operators in order from highest precedence to the lowest:

`$`	Dereference.
`(...)`	Grouping.
`+ - not !`	Unary plus, minus. Unary boolean negation.
`* / %`	Multiplication, division, modulus.

`+ -` Addition, subtraction.

`< <= = != > >=`
Relational operators.

`and` Logical '`and`'.

`or` Logical '`or`'.

12.2.7 Function Definitions

A *function* is a name for a particular sequence of statements. The syntax for the function definition is:

```
name
begin
  ...
end
```

where *name* is function name and '`...`' represent a non-empty list of valid `radtest` statements.

Notice that newline characters are obligatory after *name*, `begin` and before the final `end` keyword.

If the function accepts arguments, these can be referenced in the function body using `$n` notation (see Section 12.2.5 [Positional Parameters], page 143). To return the value from the function `return` statement is used.

For example, here is a function that computes sum of the squares of its two arguments:

```
hypo
begin
        return $1*$1 + $2*$2
end
```

12.2.8 Interacting with Radius Servers

`Radtest` provides two commands for interaction with remote RADIUS servers.

Command `send` sends request to the server specified in '`raddb/client.conf`'. Its syntax is:

```
send [flags] port-type code [expr-or-pair-list]
```

Optional *flags* can be used for fine-tuning the internals of `send`. You will seldom need to use these, unless you are developing GNU Radius. See Section 12.2.11 [Built-in Primitives], page 153, for the detailed description of these.

The first obligatory argument, *port-type*, specifies which RADIUS port to send the request to. Specifying '`auth`' will send the request to the authentication port (see Section 12.1 [client.conf], page 135); specifying '`acct`' will send it to the accounting port (see Section 12.1 [client.conf], page 135).

Argument *code* gives the request code. It is either a number or a symbolic request code name (see Section 12.2.2.1 [Numeric Values], page 137).

The last argument, *expr-or-pair-list* is either a `radtest` expression evaluating to `avlist` or a list of A/V pairs. These pairs will be included in the request.

Here are several examples:

```
# Send a Status-Server request without attributes.
send auth Status-Server

# Send an Access-Request with two attributes
send auth Access-Request User-Name = "foo" User-Password = "bar"

# Send an Accounting-Request, taking attributes from the variable
# attr
send acct Accounting-Request $attr
```

Command `send` stores the reply code into the variable `REPLY_CODE` and reply pairs into the variable `REPLY` (see Section 12.2.4.6 [Built-in Variables], page 143).

Another primitive is `expect`. `Expect` takes at most two arguments: a request code (either numeric or symbolic, (see Section 12.2.2.1 [Numeric Values], page 137)) and optional list of A/V pairs (similar to `send` *expr-or-pair-list* argument). `Expect` check if these match current `REPLY_CODE` and `REPLY` values and if so, prints the string '`PASS`'. Otherwise, it prints '`FAIL`'. This command is designed primarily for use in GNU Radius testsuite.

`Expect` is usually used right after `send`, as shown in the example below:

```
send auth Access-Request User-Name = "foo" User-Password = "bar"
expect Access-Accept Reply-Message = "Access allowed"
```

12.2.9 Conditional Statements

(This message will disappear, once this node revised.)

`Radtest` provides two kinds of conditional statements: `if` and `case`.

If statement

An `if` statement in its simplest form is:

```
if cond stmt
```

where *cond* is a conditional expression and *stmt* is a valid `radtest` statement. Optional newline may be inserted between *cond stmt*.

In this form, `if` evaluates the condition and if it yields true, executes the statement. For example:

```
if $REPLY[NAS-IP-Address] = 127.0.0.1
   print "Request from localhost"
```

More complex form of this statement allows to select between the two statements:

```
if cond stmt-1 else stmt-2
```

Here, *stmt-1* will be executed if *cond* evaluates to true, and *stmt-2* will be executed if *cond* evaluates to false.

Notice, that an optional newline is allowed between *cond* and *stmt-1* and right after `else` keyword. However, a newline before `else` constitutes an error.

If several statements should be executed in a branch of the `if` statement, use compound statement as in the example below:

```
if $REPLY_CODE != Accounting-Response
begin
  print "Accounting failed.\n"
  exit 1
end else
  print "Accounting succeeded.\n"
```

`If` statements can be nested to any depth.

Case statement

`Case` statement allows select a statement based on whether a `string` expression matches given regular expression. The syntax of `case` statement is:

```
case expr in
expr-1 ) stmt-1
expr-2 ) stmt-2
...
expr-n ) stmt-n
end
```

where *expr* is a control expression, *expr-1*, *expr-2* etc. are expressions evaluating to *extended* POSIX regular expressions (for the detailed description of these see Section "Regular Expression Library" in *Regular Expression Library*).

`Case` statement first evaluates *expr* and converts it to `string` data type. Then it evaluates each *expr-n* in turn and tests if the resulting regular expression matches *expr*. If so, the statement *stmt-n* is executed and the execution of `case` statement finishes.

The following example illustrates the concept:

```
case $COMMAND in
"auth.*")       authenticate($LIST, no)
"acct")         authenticate($LIST, yes)
".*")           begin
                  print "Unknown command."
                  exit 1
                end
end
```

`Bourne shell` programmers should notice that:

- `Case` statement ends with `end`, not `esac`.
- There is no need to put `;;` at the end of each branch,
- Boolean operations are not allowed in *expr-n*.

12.2.10 Loops

(This message will disappear, once this node revised.)

Two looping constructs are provided: `while` and `do...while`.

While loop

The syntax of a while loop is:

```
while cond
  stmt
```

Newline after *cond* is obligatory.

Do...while loop

```
do
  stmt
while cond
```

As usual `do...while` loop differs from its `while` counterpart in that its *stmt* is executed at least once.

The looping constructs can be nested to any depth.

Two special statements are provided for branching within loop constructs. These are `break` and `continue`.

`Break` statement stops the execution of the current loop statement and passes control to the statement immediately following it

```
while $x < 10
begin
  if $x < $y
     break
  ...
  x = $x + 1
end
print "OK\n"
```

In the example above, execution of `break` statement passes control to `print` statement.

`Break` may also take an argument: a literal number representing the number of nested loop statements to break from. For example, the `break` statement in the sample code below will exit from the outermost `while`:

```
while $y < 10
begin
  while $x < 10
  begin
    if $x < $y
       break 2
    ...
    x = $x + 1
  end
  ...
  y = $y + 1
end
print "OK\n"
```

`Continue` statement passes control to the condition of the current looping construct. When used with a numeric argument, the latter specifies the number of the nesting looping construct to pass control to (as with `break`, the innermost loop is considered to have number 1, so `continue` is equivalent to `continue 1`).

12.2.11 Built-in Primitives

`getopt` *`optstring`* [*`opt`* [*`arg`* [*`ind`*]]] [Radtest built-in]

`Getopt` is used to break up command line options for subsequent parsing.

The only mandatory argument, *optstring* is a list of short (one-character) options to be recognized. Each short option character in *optstring* may be followed by one colon to indicate it has a required argument, and by two colons to indicate it has an optional argument.

Each subsequent invocation of `getopt` processes next command line argument. `Getopt` returns true if the argument is an option and returns false otherwise. It stores the retrieved option (always with a leading dash) in the variable *opt* (`OPTVAR` by default). If the option has an argument, the latter is stored in the variable *arg* (`OPTARG` by default). Index of the next command line argument to be processed is preserved in the variable *ind* (`OPTIND` by default).

The usual way of processing command line options is by invoking `getopt` in a condition expression of `while` loop and analyzing its return values within the loop. For example:

```
while getopt "hf:"
case $OPTVAR in
"-h")  print "Got -h option\n"
"-f")  print "Got -f option. Argument is " $OPTARG "\n"
".*")  begin
          print "Unknown option: " $OPTVAR "\n"
          exit 1
       end
  end
end
```

`input [expr name]` [Radtest statement]
Evaluates *expr* and prints its result on standard output. Then reads a line from standard input and assigns it to the variable *name*.

If *expr* is given, *name* must also be present.

If *name* is not given, variable *INPUT* is used by default.

`set options` [Radtest statement]
Sets `radtest` command line options. *Options* should be a valid `radtest` command line (see Section 12.2.1 [Invoking radtest], page 136).

`shift [expr]` [Radtest statement]
Shift positional parameters left by one, so that `$2` becomes `$1`, `$3` becomes `$2` etc. `$#` is decremented. `$0` is not affected.

If *expr* is given, it is evaluated, converted to integer and used as shift value. Thus `shift 2` shifts all positional parameters left by 2.

`return [expr]` [Radtest statement]
Returns from the current function (see Section 12.2.7 [Function Definitions], page 149). If *expr* is present, it is evaluated and the value thus obtained becomes the function return value.

It is an error to use `return` outside of a function definition.

`break [n]` [Radtest statement]
Exit from within a loop.If *n* is specified, break from *number* levels. *n* must be >= 1. If *n* is greater than the number of enclosing loops, an error message is issued.

See Section 12.2.10 [Loops], page 152, for the detailed discussion of the subject.

`continue [n]` [Radtest statement]
Resume the next iteration of the enclosing loop. If *n* is specified, resume at the *n*th enclosing loop. *n* must be >= 1. If *n* is greater than the number of enclosing loops, an error message is issued.

See Section 12.2.10 [Loops], page 152, for the detailed discussion of the subject.

`exit [expr]` [Radtest statement]
Exit to the shell. If *expr* is specified, it is evaluated and used as exit code. Otherwise, 0 is returned to the shell.

`print expr-list` [Radtest statement]
Evaluate and print expressions. *Expr-list* is whitespace or comma-separated list of expressions. Each expression is evaluated in turn and printed to the standard output.

`send [flags] port-type code expr-or-pair-list` [Radtest statement]

Send a request to the RADIUS server and wait for the reply. Stores reply code in the variable `REPLY_CODE` and reply A/V pairs in the variable `REPLY` (see Section 12.2.8 [Interacting with Radius Servers], page 149).

flags are a whitespace-separated list of variable assignments. Following variables are understood:

`repeat=n` Unconditionally resend the request *n* times.

`id=n` Specify the request ID.

`keepauth=1`
Do not alter request authenticator when resending the request.

port-type Specifies which port to use when sending the request. Use '`auth`' to send the request to the authentication port (see Section 12.1 [client.conf], page 135), and '`acct`' to send it to the accounting port (see Section 12.1 [client.conf], page 135).

code RADIUS request code. Either numeric or symbolic (see Section 12.2.2.1 [Numeric Values], page 137).

expr-or-pair-list
Specifies the A/V pairs to include in the request. This argument is either an expression evaluating to `avlist`, or an immediate `avlist` (see Section 12.2.2.3 [Avlists], page 139). In the latter case, the parentheses around the list are optional.

`expect code [expr-or-pair-list]` [Radtest statement]

Test if `REPLY_CODE` matches *code* and, optionally, if `REPLY` matches *expr-or-pair-list*. If so, print the string '`PASS`', otherwise print '`FAIL`'.

See Section 12.2.8 [Interacting with Radius Servers], page 149, for the detailed discussion of this statement.

12.2.12 Sample Radtest Program

As an example, let's consider `radauth` program (see Section 11.6 [Radauth], page 129). Its main purpose is to send authentication request to the remote server, analyze its reply and if it is positive, send an appropriate accounting record, thereby initiating user's session. Optionally, the script should also be able to send a lone accounting record.

In the discussion below, we will show and explain subsequent parts of the script text. For the ease of explanation, each line of program text will be prepended by its ordinal line number.

Parsing command line options

The script begins as follows:

```
1 #! /usr/bin/radtest -f
2
3 while getopt "n:s:P:hv"
4 begin
5   case $OPTVAR in
6   "-n") NASIP = $OPTARG
7   "-s") SID = $OPTARG
8   "-P") PID = $OPTARG
9   "-v") set -v
```

1 It is a *pragmatic comment* informing shell that it should run `radtest` in order to interpret the program.

3 This line starts option processing loop. `Getopt` (see Section 12.2.11 [Built-in Primitives], page 153) in line 3 analyzes each subsequent command line argument and if it is an option checks whether it matches one of the option letters defined in its first argument. The option letter will be returned in `OPTVAR` variable, its argument (if any) – in `OPTARG` variable.

4 – 8 `OPTARG` value is analyzed using `case` statement. Lines 6 – 8 preserve `OPTARG` values in appropriate variables for later use. `NASIP` will be used as the value of `NAS-IP-Address` attribute, `SID` is the session id (`Acct-Session-Id` attribute), and `PID` is the port number (for `NAS-Port-Id` attribute.

9 This line sets '`-v`' option to the `radtest` interpreter (see Section 12.2.1 [Invoking radtest], page 136).

The next piece of code handles '`-h`' and erroneous options:

```
10   "-h") begin
11           print <<-EOT
12            usage: radauth [OPTIONS] [COMMAND] login [password]
13            Options are:
14            -v         Print verbose descriptions of what is being done
15            -n IP      Set NAS IP address
16            -s SID     Set session ID
17            -P PORT    Set NAS port number
18            COMMAND is one of:
19            auth       Send only Access-Request (default)
20            acct       Send Access-Request. If successfull, send
21                       accounting start request
22            start      Send accounting start request
23            stop       Send accounting stop request
24            EOT
25           exit 0
26         end
27   ".*") begin
28           print "Unknown option: " $OPTVAR "\n"
29           exit 1
30         end
31   end
32 end
```

10 – 26 Print short description and exit, if the program is given '`-h`'. Notice that '`here document`' syntax is used to print the text (See Section 12.2.2.2 [Strings], page 137, for its description). The leading whitespace in lines 12 to 24 is composed of tabulation characters (ASCII 9), not usual space characters (ASCII 32), as required by '`<<-`' construct.

27 – 30 These lines handle unrecognized options.

31 Closes case statement started on line 5

32 Closes compound statement started on line 4

Checking Command Line Consistency

```
33
34 shift ${OPTIND}-1
35
36 if $# > 3
37 begin
38         print "Wrong number of arguments."
39         print "Try radauth -h for more info"
40         exit 1
41 end
```

34 `OPTIND` keeps the ordinal number of the first non-optional argument. This line shifts off all the options processed by `getopt`, so that the first non-optional argument may be addressed by `$1` notation. Notice use of curly braces to solve *minus ambiguity* (see [minus-ambiguity], page 141).

36 – 41 At this point we may have at most three arguments: command, user name, and password. If there are more, display the diagnostic message and exit the program.

Next piece of code:

```
42
43 case $1 in
44 "auth|acct|start|stop") begin
45                            COMMAND=$1
46                            shift 1
47                          end
48 ".*")   COMMAND="auth"
49 end
50
51 LOGIN=${1:?User name is not specified. Try radauth -h for more info.}
52
53 if ${NASIP:-} = ""
54         NASIP=$SOURCEIP
55
56 LIST = ( User-Name = $LOGIN NAS-IP-Address = $NASIP )
```

43 – 48 Check if a command is given. If so, store command name in the variable `COMMAND` and shift arguments by one, so login becomes argument `$1`. Otherwise, assume '`auth`' command.

51 If the user login name is supplied, store it into `LOGIN` variable. Otherwise, print diagnostic message and exit.

53 – 54 Provide a default value for `NASIP` variable from the built-in variable `SOURCEIP` (see Section 12.2.4.6 [Built-in Variables], page 143)

56 The variable `LIST` will hold the list of A/V pairs to be sent to the server. This line initializes it with a list of two A/V pairs: `User-Name` and `NAS-IP-Address`.

Defining Accounting Function

Accounting function will be used to send accounting requests to the server. It is supposed to take a single argument: an `avlist` of A/V pairs to be sent to the server.

```
57
58 'acct'
59 begin
60   if ${SID:-} = ""
61     input "Enter session ID: " SID
62   if ${PID:-} = ""
63     input "Enter NAS port ID: " PID
64   send acct Accounting-Request $1 + \
          (Acct-Session-Id = $SID NAS-Port-Id = $PID)
```

58 – 59 These lines start the function definition. Notice quoting of the function name ('`acct`'): it is necessary because it coincides with

a reserved keyword (see Section 12.2.3 [Reserved Keywords], page 139).

60 – 61 If the value of `SID` (session ID) is not supplied, prompt the user to input it.

62 – 63 If the value of `PID` (port ID) is not supplied, prompt the user to input it.

64 Send accounting request. The list of A/V pairs to send is formed by concatenating `Acct-Session-Id` and `NAS-Port-Id` attributes to the function's first argument.

The final part of `acct` function analyzes the reply from the server:

```
65   if $REPLY_CODE != Accounting-Response
66   begin
67     print "Accounting failed.\n"
68     exit 1
69   end
70   print "Accounting OK.\n"
71   exit 0
72 end
73
```

Notice, that `acct` never returns. Instead it exits with an error code indicating success or failure.

Defining Authentication Function

The purpose of the authentication function `auth` is to send an `Access-Request` to the server and perform some actions based on its reply.

The function will take three arguments:

`$1` The list of A/V pairs to include in the request.

`$2` User password.

`$3` This argument indicates whether accounting request must be sent after successful authentication. String '`yes`' means to send the accounting request, '`no`' means not to send it.

The function is not expected to return. Instead it should exit to the shell with an appropriate error code.

```
74 'auth'
75 begin
76   send auth Access-Request $1 + (User-Password = $2)
```

74 – 75 Begin the function definition. Notice quoting of the function name ('`auth`'): it is necessary because it coincides with a reserved keyword (see Section 12.2.3 [Reserved Keywords], page 139).

76 Send the initial authentication request. The list of A/V pairs is formed by appending `User-Password` pair to the list given by the first argument to the function.

The rest of the function analyzes the reply from the server and takes appropriate actions. Notice that if the server replies with an `Access-Challenge` packet, we will have to send subsequent authentication requests, so this piece of code is enclosed within a `while` loop.

First, the function handles `Access-Accept` and `Access-Reject` replies:

```
77   while 1
78   begin
79     if $REPLY_CODE = Access-Accept
80     begin
81       print "Authentication passed. " + $REPLY[Reply-Message*] + "\n"
82       if ${3:-no} = no
83         exit 0
84       'acct'($1 + ( Acct-Status-Type = Start ))
85     end else if $REPLY_CODE = Access-Reject
86     begin
87       print "Authentication failed. " + $REPLY[Reply-Message*] + "\n"
88       break
```

77 Begin an "endless" `while` loop. It will eventually be exited either using `break`, or using `exit` (see below).

79 – 84 Hanlde `Access-Accept` replies:

81 Print the reply message. Notice the use of '`*`' to print all the instances of `Reply-Message` attribute from the reply packet (see Section 12.2.4.4 [Accessing Elements of A/V Pair Lists], page 142).

82 – 83 If the third argument is missing or is a string '`no`', exit indicating success (see Section 12.2.4.3 [Dereferencing Variables], page 140).

84 Otherwise, call `acct` function to perform accounting. The A/V pairs included in the accounting request are formed by adding `Acct-Status-Type` attribute to the list given by the first argument to the function.

85 – 88 Handle `Access-Reject` replies. Print the reply message and break from the loop.

Next piece of code deals with `Access-Challenge` replies. For simplicity we assume that such replies always carry user menus (See Section 4.13 [menus directory], page 62, for the description of these). So, upon receiving an `Access-Challenge` we should print out the menu, read the users selection and send back an `Access-Request` to the server. This part is the only one that actually continues the loop at line 77.

```
 89     end else if $REPLY_CODE = Access-Challenge
 90     begin
 91       print $REPLY[Reply-Message*]
 92       input
 93       send auth Access-Request \
 94         (User-Name = $LOGIN User-Password = $INPUT \
             State = $REPLY[State])
```

91 Print the menu contents carrieb by `Reply-Message` attributes. There may be several instances of the attribute, hence the use of '`*`' to concatenate their values together.

92 Read the input from the user. The input will be stored in `INPUT` variable. See Section 12.2.11 [Built-in Primitives], page 153, for the description of `input` statement.

93 – 94 Send an `Access-Request` packet with three attributes. `User-Password` contains the user reply, `State` contains the menu state from the server reply packet.

Final part of the function:

```
 95     end else begin
 96       print "Authentication failed. Reply code " + $REPLY_CODE + "\n"
 97       break
 98     end
 99   end
100   exit 1
101 end
102
```

95 – 98 Handle unknown reply codes.

99 Closes the while loop started on line 77.

100 Exit to the shell indicating failure. This statement will be reached only if a `break` is executed either on line 88 or on line 97.

101 Closes function definition started on lines 74 – 75

Final Part of Radauth Program

The final part selects an action based on the user command and executes it. It is equivalent to the `main` function in a `C` program:

```
103 case ${COMMAND} in
104 "auth")   'auth'($LIST, ${2:&Password: }, no)
105 "acct")   'auth'($LIST, ${2:&Password: }, yes)
106 "start")  'acct'($LIST+(Acct-Status-Type = Start))
107 "stop")   'acct'($LIST+(Acct-Status-Type = Stop))
108 ".*")       begin
109               print "Unknown command. Try radauth -h for more info"
110               exit 1
111             end
112 end
113
114 # End of radauth
```

103 Select an action based on the value of `COMMAND` variable.

104 – 105 Call `auth` function. If the second argument is given in the command line, its value is taken as user's password. Otherwise, the user is prompted for the password with the string '`Password: `'. The input is read with echo turned off to prevent the password from being compromised (the '`:&`' construct, see Section 12.2.4.3 [Dereferencing Variables], page 140).

106 – 107 Call `acct` function for '`start`' and `stop` commands.

108 – 111 Handle an unknown command verb.

112 Closes `case` statement from line 103.

12.3 radsession

`radsession` is a Guile script that sends authentication and accounting requests to the Radius server. To invoke the script, run

```
radsession options action
```

Possible actions are:

'`--auth`' Send authentication request.

'`--start`' Send accounting start request.

'`--stop`' Send accounting stop request.

Options determine the contents of the request's pairlist. They are:

'`-l STRING`'
'`--login STRING`'
Set login name.

'`-p STRING`'
'`--passwd STRING`'
Set password.

'`-n IP`'
'`--nas IP`' Set the value of `NAS-IP-Address` attribute.

‘`-s STRING`’
‘`--sid STRING`’
: Set the session ID (`Acct-Session-Id` attribute).

‘`-P NUMBER`’
‘`--port NUMBER`’
: Set the port number (`NAS-Port-Id` attribute).

‘`-h`’
‘`--help`’ Print a short usage message and exit.

‘`-v`’
‘`--verbose`’
: Verbosely list the contents of the received reply.

12.4 nas.scm

`nas.scm` is a Guile program that allows one to convert a GNU/Linux box into a NAS. It requires Guile version 1.4 or better and PPP version 2.3.7 or better.

To use it, you will basically need to do the following:

1. Create links:

   ```
   ln -s libexec/nas.scm /etc/ppp/ip-up
   ln -s libexec/nas.scm /etc/ppp/ip-down
   ```

   Here, *libexec* denotes the location of your libexec directory, where `nas.scm` is installed. If not overridden at configure time, it defaults to ‘*prefix*`/libexec`’. These links assure that `ppp` will invoke `nas.scm` when the user’s session starts and ends, thus giving it a possibility to send accounting requests.
2. Configure the file ‘`raddb/client.conf`’.
3. Edit the file ‘`raddb/nas.rc`’. The supplied ‘`nas.rc`’ template is tailored to work in most environments. The only variables you may need to change are `nas-log-facility`, specifying the syslog facility to be used for logging, and `pppd-args`, keeping the arguments to be given to `ppp`.
4. Configure your ‘`/etc/inittab`’ and `getty`.

   For example, if you use `mgetty`, then the ‘`inittab`’ entries for dial-up lines will look like:

   ```
   d0:345:respawn:/sbin/mgetty ttyS0 vt100
   d1:345:respawn:/sbin/mgetty ttyS1 vt100
   ...
   ```

   `mgetty`’s ‘`login.config`’ will then contain the following line:

   ```
   *       -       -       /usr/local/libexec/nas.scm @
   ```

   If you use `agetty`, then the ‘`inittab`’ will contain (with the long lines split for readability)

```
d0:345:respawn:/sbin/agetty -mt60 \
   -l /usr/local/libexec/nas.scm 38400,19200,9600 \
   ttyS0 vt100
d1:345:respawn:/sbin/agetty -mt60 \
   -l /usr/local/libexec/nas.scm 38400,19200,9600 \
   ttyS1 vt100
...
```

12.5 pam_radius.so

`pam_radius.so` is a PAM module for Radius authentication. The module understands the following command line options:

'`audit`' Enable audit information.

'`debug[=`*level*`]`'
: Enable debugging information. The higher *level* is, the more debugging info is output. When omitted, *level* defaults to 100.

  Debugging levels equal to or greater than 10 compromise users' passwords, so use them sparingly.

'`use_authtok`'
: Use the authentication token passed from the previous module in the stack.

'`confdir=`*path*'
: Look for configuration files in *path*. The default is '`$sysconfdir/etc/raddb`'.

'`attr:`' This keyword marks the end of command line options. The part of the command line after it is parsed as a whitespace-separated list of A/V pairs to be sent with the request.

'`service_type=`*type*'
: This option is retained for compatibility with the 0.96 series of GNU Radius. It is equivalent to

```
attr: Service-Type=type
```

The `pam_radius.so` module logs its messages under `LOG_AUTH` syslog facility.

13 Attribute List

The following sections describe the most frequently used Radius attributes. Each attribute is described as follows:

`ATTRIBUTE` *`name value type`*

Users: *`user-flags`*
Hints: *`hints-flags`*
Huntgroups: *`huntgroup-flags`*
Additivity: *additivity*
Proxy propagated: *prop*

These values have the following meaning:

name The attribute name.

value The attribute number.

type The attribute type.

user-flags Syntax flags defining in which part of a '`raddb/users`' entry this attribute may be used. The flags consist of two letters: '`L`' means the attribute can be used in the LHS, '`R`' means it can be used in the RHS.

hints-flags Syntax flags defining in which part of a '`raddb/hints`' entry this attribute may be used.

huntgroup-flags
Syntax flags defining in which part of a '`raddb/huntgroups`' entry this attribute may be used.

additivity The *additivity* of the attribute determines what happens if a rule attempts to add to the pair list an attribute that is already present in this list. Depending on its value, the actions of the server are:

Append New attribute is appended to the end of the list.

Replace New attribute replaces the old.

Drop New attribute is dropped. The old one remains in the list.

prop Is the attribute propagated back to the NAS if the server works in proxy mode?

The entry N/A for any of this fields signifies "not applicable".

13.1 Authentication Attributes

These are the attributes the NAS uses in authentication packets and expects to get back in authentication replies. These can be used in matching rules.

13.1.1 CHAP-Password

```
ATTRIBUTE CHAP-Password 3 string
```

Users:	L-
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	No

This attribute indicates the response value provided by a PPP Challenge-Handshake Authentication Protocol (CHAP) user in response to the challenge. It is only used in Access-Request packets.

The CHAP challenge value is found in the CHAP-Challenge attribute (60) if present in the packet, otherwise in the request authenticator field.

13.1.2 Callback-Id

```
ATTRIBUTE Callback-Id 20 string
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	No

This attribute indicates the name of a place to be called, to be interpreted by the NAS. It may be used in Access-Accept packets.

13.1.3 Callback-Number

```
ATTRIBUTE Callback-Number 19 string
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	No

This attribute indicates a dialing string to be used for callback. It may be used in Access-Accept packets. It may be used in an Access-Request packet as a hint to the server that a Callback service is desired, but the server is not required to honor the hint.

13.1.4 Called-Station-Id

```
ATTRIBUTE Called-Station-Id 30 string
```

Users:	L-
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute allows the NAS to send in the Access-Request packet the phone number that the user called, using Dialed Number Identification

(DNIS) or similar technology. Note that this may be different from the phone number the call comes in on. It is only used in Access-Request packets.

13.1.5 Calling-Station-Id

```
ATTRIBUTE Calling-Station-Id 31 string
```

Users:	L-
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute allows the NAS to send in the Access-Request packet the phone number that the call came from, using automatic number identification (ANI) or similar technology. It is only used in Access-Request packets.

13.1.6 Class

```
ATTRIBUTE Class 25 string
```

Users:	LR
Hints:	LR
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute is available to be sent by the server to the client in an Access-Accept and should be sent unmodified by the client to the accounting server as part of the Accounting-Request packet if accounting is supported.

13.1.7 Framed-Compression

```
ATTRIBUTE Framed-Compression 13 integer
```

Users:	LR
Hints:	-R
Huntgroups:	LR
Additivity:	Replace
Proxy propagated:	Yes

```
VALUE      Framed-Compression  None                0
VALUE      Framed-Compression  Van-Jacobson-TCP-IP  1
```

This attribute indicates a compression protocol to be used for the link. It may be used in Access-Accept packets. It may be used in an Access-Request packet as a hint to the server that the NAS would prefer to use that compression, but the server is not required to honor the hint.

More than one compression protocol attribute may be sent. It is the responsibility of the NAS to apply the proper compression protocol to appropriate link traffic.

13.1.8 Framed-IP-Address

```
ATTRIBUTE Framed-IP-Address 8 ipaddr
```

Users:	LR
Hints:	-R
Huntgroups:	LR
Additivity:	Replace
Proxy propagated:	No

This attribute indicates the address to be configured for the user. It may be used in Access-Accept packets. It may be used in an Access-Request packet as a hint by the NAS to the server that it would prefer that address, but the server is not required to honor the hint.

The value `0xFFFFFFFF` (`255.255.255.255`) indicates that the NAS should allow the user to select an address. The value `0xFFFFFFFE` (`255.255.255.254`) indicates that the NAS should select an address for the user (e.g. assigned from a pool of addresses kept by the NAS). Other valid values indicate that the NAS should use that value as the user's IP.

When used in a RHS, the value of this attribute can optionally be followed by a plus sign. This usage means that the value of `NAS-Port-Id` must be added to this IP before replying. For example,

```
Framed-IP-Address = 10.10.0.1+
```

13.1.9 Framed-IP-Netmask

```
ATTRIBUTE Framed-IP-Netmask 9 ipaddr
```

Users:	LR
Hints:	-R
Huntgroups:	LR
Additivity:	Replace
Proxy propagated:	No

This attribute indicates the IP netmask to be configured for the user when the user is a router to a network. It may be used in Access-Accept packets. It may be used in an Access-Request packet as a hint by the NAS to the server that it would prefer that netmask, but the server is not required to honor the hint.

13.1.10 Framed-MTU

```
ATTRIBUTE Framed-MTU 12 integer
```

Users:	LR
Hints:	-R
Huntgroups:	-R
Additivity:	Replace
Proxy propagated:	Yes

This attribute indicates the maximum transmission unit to be configured for the user, when it is not negotiated by some other means (such as PPP). It is only used in Access-Accept packets.

13.1.11 Framed-Protocol

```
ATTRIBUTE Framed-Protocol 7 integer
```

Users:	LR
Hints:	-R
Huntgroups:	LR
Additivity:	Replace
Proxy propagated:	Yes

```
VALUE      Framed-Protocol   PPP                1
VALUE      Framed-Protocol   SLIP               2
```

This attribute indicates the framing to be used for framed access. It may be used in both Access-Request and Access-Accept packets.

13.1.12 Framed-Route

```
ATTRIBUTE Framed-Route 22 string
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	No

This attribute provides routing information to be configured for the user on the NAS. It is used in the Access-Accept packet and can appear multiple times.

13.1.13 Framed-Routing

```
ATTRIBUTE Framed-Routing 10 integer
```

Users:	-R
Hints:	-R
Huntgroups:	-R
Additivity:	Replace
Proxy propagated:	No

```
VALUE      Framed-Routing    None               0
VALUE      Framed-Routing    Broadcast          1
VALUE      Framed-Routing    Listen             2
VALUE      Framed-Routing    Broadcast-Listen   3
```

This attribute indicates the routing method for the user when the user is a router to a network. It is only used in Access-Accept packets.

13.1.14 Idle-Timeout

```
ATTRIBUTE Idle-Timeout 28 integer
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	Yes

This attribute sets the maximum number of consecutive seconds of idle connection allowed to the user before termination of the session or prompt. The server may send this attribute to the client in an Access-Accept or Access-Challenge.

13.1.15 NAS-IP-Address

```
ATTRIBUTE NAS-IP-Address 4 ipaddr
```

Users:	L-
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute indicates the identifying IP of the NAS which is requesting authentication of the user. It is only used in Access-Request packets. Each Access-Request packet should contain either a `NAS-IP-Address` or a `NAS-Identifier` attribute (Section 13.1.16 [NAS-Identifier], page 170).

13.1.16 NAS-Identifier

```
ATTRIBUTE NAS-Identifier 32 string
```

Users:	L-
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute contains a string identifying the NAS originating the access request. It is only used in Access-Request packets. Either `NAS-IP-Address` or `NAS-Identifier` should be present in an Access-Request packet.

See Section 13.1.15 [NAS-IP-Address], page 170.

13.1.17 NAS-Port-Id

```
ATTRIBUTE NAS-Port-Id 5 integer
```

Users:	LR
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

This attribute indicates the physical port number of the NAS that is authenticating the user. It is only used in Access-Request packets. Note that here we are using "port" in its sense of a physical connection on the NAS, not in the sense of a TCP or UDP port number.

Some NASes try to encode various information in the `NAS-Port-Id` attribute value. For example, the MAX Ascend terminal server constructs `NAS-Port-Id` by concatenating the line type (one digit), the line number (two digits), and the channel number (two digits), thus producing a five-digit port

number. In order to normalize such encoded port numbers we recommend using a rewrite function (see Section 4.12 [rewrite file], page 62). A rewrite function for MAX Ascend servers is provided in the distribution.

13.1.18 NAS-Port-Type

```
ATTRIBUTE NAS-Port-Type 61 integer
```

Users: --
Hints: --
Huntgroups: --
Additivity: Append
Proxy propagated: No

```
VALUE      NAS-Port-Type     Async              0
VALUE      NAS-Port-Type     Sync               1
VALUE      NAS-Port-Type     ISDN               2
VALUE      NAS-Port-Type     ISDN-V120          3
VALUE      NAS-Port-Type     ISDN-V110          4
```

This attribute indicates the type of the physical port of the NAS that is authenticating the user. It can be used instead of or in addition to the `NAS-Port-Id` (Section 13.1.17 [NAS-Port-Id], page 170) attribute. It is only used in Access-Request packets. Either `NAS-Port` or `NAS-Port-Type` or both should be present in an Access-Request packet, if the NAS differentiates among its ports.

13.1.19 Reply-Message

```
ATTRIBUTE Reply-Message 18 string
```

Users: -R
Hints: --
Huntgroups: --
Additivity: Append
Proxy propagated: Yes

This attribute indicates text that may be displayed to the user.

When used in an Access-Accept, it is the success message.

When used in an Access-Reject, it is the failure message. It may indicate a dialog message to prompt the user before another Access-Request attempt.

When used in an Access-Challenge, it may indicate a dialog message to prompt the user for a response.

Multiple `Reply-Message` attributes may be included, and if any are displayed, they must be displayed in the same order as they appear in in the packet.

13.1.20 Service-Type

```
ATTRIBUTE Service-Type 6 integer
```

Users: LR
Hints: -R

Huntgroups: LR
Additivity: Replace
Proxy propagated: Yes

```
VALUE      Service-Type      Login-User           1
VALUE      Service-Type      Framed-User          2
VALUE      Service-Type      Callback-Login-User  3
VALUE      Service-Type      Callback-Framed-User 4
VALUE      Service-Type      Outbound-User        5
VALUE      Service-Type      Administrative-User  6
VALUE      Service-Type      NAS-Prompt-User      7
VALUE      Service-Type      Authenticate-Only    8
VALUE      Service-Type      Call-Check           10
```

This attribute indicates the type of service the user has requested, or the type of service to be provided. It may be used in both Access-Request and Access-Accept packets.

When used in an Access-Request the service type represents a hint to the Radius server that the NAS has reason to believe the user would prefer the kind of service indicated.

When used in an Access-Accept, the service type is an indication to the NAS that the user must be provided this type of service.

The meaning of various service types is as follows:

`Login-User`
: The user should be connected to a host.

`Framed-User`
: A framed protocol, such as PPP or SLIP, should be started for the user. The `Framed-IP-Address` attribute (see Section 13.1.8 [Framed-IP-Address], page 168) will supply the IP to be used.

`Callback-Login-User`
: The user should be disconnected and called back, then connected to a host.

`Callback-Framed-User`
: The user should be disconnected and called back; then a framed protocol, such as PPP or SLIP, should be started for the user.

`Outbound-User`
: The user should be granted access to outgoing devices.

`Administrative-User`
: The user should be granted access to the administrative interface to the NAS, from which privileged commands can be executed.

`NAS-Prompt`
: The user should be provided a command prompt on the NAS, from which nonprivileged commands can be executed.

`Authenticate-Only`
: Only authentication is requested, and no authorization information needs to be returned in the Access-Accept.

`Call-Check`
`Callback-NAS-Prompt`
: The user should be disconnected and called back, then provided a command prompt on the NAS, from which nonprivileged commands can be executed.

13.1.21 `Session-Timeout`

```
ATTRIBUTE Session-Timeout 27 integer
```

Users:	`-R`
Hints:	`--`
Huntgroups:	`--`
Additivity:	Replace
Proxy propagated:	Yes

This attribute sets the maximum number of seconds of service to be provided to the user before termination of the session or prompt. The server may send this attribute to the client in an Access-Accept or Access-Challenge.

13.1.22 `State`

```
ATTRIBUTE State 24 string
```

Users:	`LR`
Hints:	`LR`
Huntgroups:	`LR`
Additivity:	Append
Proxy propagated:	No

This attribute is available to be sent by the server to the client in an Access-Challenge and **must** be sent unmodified from the client to the server in the new Access-Request reply to that challenge, if any.

This attribute is available to be sent by the server to the client in an Access-Accept that also includes a `Termination-Action` attribute with the value `RADIUS-Request`. If the NAS performs the termination action by sending a new Access-Request upon termination of the current session, it **must** include the `State` attribute unchanged in that Access-Request.

In either usage, no interpretation by the client should be made. A packet may have only one `State` attribute.

13.1.23 `Termination-Action`

```
ATTRIBUTE Termination-Action 29 integer
```

Users:	`LR`
Hints:	`-R`
Huntgroups:	`-R`
Additivity:	Replace

Proxy propagated: No

```
VALUE      Termination-Action  Default             0
VALUE      Termination-Action  RADIUS-Request      1
```

This attribute indicates what action the NAS should take when the specified service is completed. It is only used in Access-Accept packets.

13.1.24 User-Name

```
ATTRIBUTE User-Name 1 string
```

Users: LR
Hints: -R
Huntgroups: LR
Additivity: Replace
Proxy propagated: Yes

This attribute indicates the name of the user to be authenticated or accounted. It is used in Access-Request and Accounting attributes. The length of the user name is usually limited by some arbitrary value. By default, Radius supports user names up to 32 characters long. This value can be modified by redefining the `RUT_USERNAME` macro in the '`include/radutmp.h`' file in the distribution directory and recompiling the program.

Some NASes have peculiarities about sending long user names. For example, the Specialix Jetstream 8500 24-port access server inserts a '`/`' character after the 10th character if the user name is longer than 10 characters. In such cases, we recommend applying rewrite functions in order to bring the user name to its normal form (see Section 4.12 [rewrite file], page 62).

13.1.25 User-Password

```
ATTRIBUTE User-Password 2 string
```

Users: L-
Hints: --
Huntgroups: --
Additivity: N/A
Proxy propagated: No

This attribute indicates the password of the user to be authenticated, or the user's input following an Access-Challenge. It is only used in Access-Request packets.

On transmission, the password is hidden. The password is first padded at the end with nulls to a multiple of 16 octets. A one-way MD5 hash is calculated over a stream of octets consisting of the shared secret followed by the request authenticator. This value is XORed with the first 16 octet segment of the password and placed in the first 16 octets of the String field of the User-Password attribute.

If the password is longer than 16 characters, a second one-way MD5 hash is calculated over a stream of octets consisting of the shared secret followed by the result of the first xor. That hash is XORed with the second 16 octet

segment of the password and placed in the second 16 octets of the string field of the User-Password attribute.

If necessary, this operation is repeated, with each XOR result being used along with the shared secret to generate the next hash to XOR the next segment of the password, up to no more than 128 characters.

13.1.26 Vendor-Specific

(This message will disappear, once this node revised.)

```
ATTRIBUTE Vendor-Specific 26 string
```

Users:	LR
Hints:	-R
Huntgroups:	-R
Additivity:	Append
Proxy propagated:	No

This attribute is available to allow vendors to support their own extended attributes not suitable for general usage.

13.2 Accounting Attributes

These are attributes the NAS sends along with accounting requests. These attributes can not be used in matching rules.

13.2.1 Acct-Authentic

```
ATTRIBUTE Acct-Authentic 45 integer
```

Users:	--
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	N/A

```
VALUE           Acct-Authentic          RADIUS          1
VALUE           Acct-Authentic          Local           2
VALUE           Acct-Authentic          Remote          3
```

This attribute may be included in an Accounting-Request to indicate how the user was authenticated, whether by Radius, the NAS itself, or another remote authentication protocol. Users who are delivered service without being authenticated should not generate accounting records.

13.2.2 Acct-Delay-Time

```
ATTRIBUTE Acct-Delay-Time 41 integer
```

Users:	--
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	N/A

This attribute indicates how many seconds the client has been trying to send this record for, and can be subtracted from the time of arrival on the server to find the approximate time of the event generating this Accounting-Request. (Network transit time is ignored.)

13.2.3 Acct-Input-Octets

```
ATTRIBUTE Acct-Input-Octets 42 integer
```

Users:	--
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	N/A

This attribute indicates how many octets have been received from the port over the course of this service being provided, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.2.4 Acct-Input-Packets

```
ATTRIBUTE Acct-Input-Packets 47 integer
```

Users:	--
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	N/A

This attribute indicates how many packets have been received from the port over the course of this service being provided to a framed user, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.2.5 Acct-Output-Octets

```
ATTRIBUTE Acct-Output-Octets 43 integer
```

Users:	--
Hints:	--
Huntgroups:	--
Additivity:	N/A
Proxy propagated:	N/A

This attribute indicates how many octets have been sent to the port in the course of delivering this service, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.2.6 Acct-Output-Packets

```
ATTRIBUTE Acct-Output-Packets 48 integer
```

Users:	--
Hints:	--
Huntgroups:	--

Additivity: N/A
Proxy propagated: N/A

This attribute indicates how many packets have been sent to the port in the course of delivering this service to a framed user, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.2.7 Acct-Session-Id

```
ATTRIBUTE Acct-Session-Id 44 string
```

Users: --
Hints: --
Huntgroups: --
Additivity: N/A
Proxy propagated: N/A

This attribute is a unique accounting ID to make it easy to match start and stop records in a log file. The start and stop records for a given session must have the same `Acct-Session-Id`. An Accounting-Request packet must have an `Acct-Session-Id`. An Access-Request packet may have an `Acct-Session-Id`; if it does, then the NAS must use the same `Acct-Session-Id` in the Accounting-Request packets for that session.

13.2.8 Acct-Session-Time

```
ATTRIBUTE Acct-Session-Time 46 integer
```

Users: --
Hints: --
Huntgroups: --
Additivity: N/A
Proxy propagated: N/A

This attribute indicates how many seconds the user has received service for, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.2.9 Acct-Status-Type

```
ATTRIBUTE Acct-Status-Type 40 integer
```

Users: --
Hints: --
Huntgroups: --
Additivity: N/A
Proxy propagated: N/A

```
VALUE    Acct-Status-Type    Start              1
VALUE    Acct-Status-Type    Stop               2
VALUE    Acct-Status-Type    Alive              3
VALUE    Acct-Status-Type    Accounting-On      7
VALUE    Acct-Status-Type    Accounting-Off     8
```

This attribute indicates whether this Accounting-Request marks the beginning of the user service (`Start`) or the end (`Stop`).

It may also be used to mark the start of accounting (for example, upon booting) by specifying `Accounting-On` and to mark the end of accounting (for example, just before a scheduled reboot) by specifying `Accounting-Off`.

A special value `Alive` or `Interim-Update` indicates the packet that contains some additional data to the initial `Start` record or to the last `Alive` record.

13.2.10 Acct-Terminate-Cause

```
ATTRIBUTE Acct-Terminate-Cause 49 integer
```

Users: --
Hints: --
Huntgroups: --
Additivity: N/A
Proxy propagated: N/A

```
VALUE    Acct-Terminate-Cause    User-Request            1
VALUE    Acct-Terminate-Cause    Lost-Carrier            2
VALUE    Acct-Terminate-Cause    Lost-Service            3
VALUE    Acct-Terminate-Cause    Idle-Timeout            4
VALUE    Acct-Terminate-Cause    Session-Timeout         5
VALUE    Acct-Terminate-Cause    Admin-Reset             6
VALUE    Acct-Terminate-Cause    Admin-Reboot            7
VALUE    Acct-Terminate-Cause    Port-Error              8
VALUE    Acct-Terminate-Cause    NAS-Error               9
VALUE    Acct-Terminate-Cause    NAS-Request             10
VALUE    Acct-Terminate-Cause    NAS-Reboot              11
VALUE    Acct-Terminate-Cause    Port-Unneeded           12
VALUE    Acct-Terminate-Cause    Port-Preempted          13
VALUE    Acct-Terminate-Cause    Port-Suspended          14
VALUE    Acct-Terminate-Cause    Service-Unavailable     15
VALUE    Acct-Terminate-Cause    Callback                16
VALUE    Acct-Terminate-Cause    User-Error              17
VALUE    Acct-Terminate-Cause    Host-Request            18
```

This attribute indicates how the session was terminated, and can only be present in Accounting-Request records where `Acct-Status-Type` is set to `Stop`.

13.3 Radius Internal Attributes

These are attributes used by GNU Radius during the processing of a request. They are never returned to the NAS. Mostly, they are used in matching rules.

13.3.1 Acct-Ext-Program

```
ATTRIBUTE Acct-Ext-Program 2008 string
```

Users: --
Hints: -R
Huntgroups: --
Additivity: Replace

Proxy propagated: N/A

The `Acct-Ext-Program` attribute can be used in RHS of an '`raddb/hints`' to require the execution of an external accounting program or filter. If the attribute value starts with a vertical bar ('`|`'), then the attribute specifies the filter program to be used. If it starts with a slash ('`/`'), then it is understood as the full pathname and arguments for the external program to be executed. Using any other character as the start of this string results in error.

The command line can reference any attributes from both check and reply pairlists using attribute macros (see Section 4.14 [Macro Substitution], page 64).

Before the execution of the program, `radiusd` switches to the uid and gid of the user `daemon` and the group `daemon`. You can override these defaults by setting variables `exec-program-user` and `exec-program-group` in configuration file to proper values (see Section 4.1.1 [The option statement], page 22).

The accounting program must exit with status 0 to indicate a successful accounting.

13.3.2 Acct-Type

```
ATTRIBUTE Acct-Type 2003 integer
```

Users: L-
Hints: -R
Huntgroups: -R
Additivity: Append
Proxy propagated: N/A

```
VALUE           Acct-Type               None    0
VALUE           Acct-Type               System  1
VALUE           Acct-Type               Detail  2
VALUE           Acct-Type               SQL     3
```

The `Acct-Type` allows one to control which accounting methods must be used for a given user or group of users. In the absence of this attribute, all currently enabled accounting types are used. See Chapter 7 [Accounting], page 81, for more information about accounting types.

13.3.3 Auth-Failure-Trigger

This attribute specifies an external program or a Scheme expression to be run upon an authentication failure. The handling of this attribute depends upon its value:

If the value of `Auth-Failure-Trigger` begins with '`/`', it is taken to contain a command line for invoking an external program. In this case `radiusd` invokes the program much the same way it does when handling `Exec-Program` attribute, i.e. the program is invoked with standard input closed, its standard output and standard error are captured and redirected to '*`radlog`*`/radius.stderr`' file, the return value of the program is ignored.

If the value of `Auth-Failure-Trigger` begins with '`(`', it is executed it as a `Scheme` expression. The return value of the expression is ignored.

This attribute is designed as a means to provide special handling for authentication failures. It can be used, for example, to increase failure counters and to block accounts after a specified number of authentication failures occurs. See Section 6.10 [Auth Probing], page 76, for the detailed discussion of its usage.

13.3.4 Auth-Data

```
ATTRIBUTE Auth-Data 2006 string
```

Users:	`L-`
Hints:	`-R`
Huntgroups:	`-R`
Additivity:	Replace
Proxy propagated:	N/A

The `Auth-Data` can be used to pass additional data to the authentication methods that need them. In version 1.6 of GNU Radius, this attribute may be used in conjunction with the `SQL` and `Pam` authentication types. When used with the `Pam` authentication type, this attribute holds the name of the PAM service to use. This attribute is temporarily appended to the authentication request, so its value can be referenced to as `%C{Auth-Data}`. See Section 4.11.2 [Authentication Server Parameters], page 54, for an example of of using the `Auth-Data` attribute in '`raddb/sqlserver`':

13.3.5 Auth-Type

```
ATTRIBUTE Auth-Type 1000 integer
```

Users:	`L-`
Hints:	`-R`
Huntgroups:	`-R`
Additivity:	Append
Proxy propagated:	No

```
VALUE      Auth-Type         Local              0
VALUE      Auth-Type         System             1
VALUE      Auth-Type         Crypt-Local        3
VALUE      Auth-Type         Reject             4
VALUE      Auth-Type         SQL                252
VALUE      Auth-Type         Pam                253
VALUE      Auth-Type         Accept             254
```

This attribute tells the server which type of authentication to apply to a particular user. It can be used in the LHS of the user's profile (see Chapter 6 [Authentication], page 71.)

Radius interprets values of `Auth-Type` attribute as follows:

`Local` The value of the `User-Password` attribute from the record is taken as a cleantext password and is compared against the `User-Password` value from the input packet.

`System` This means that a user's password is stored in a system password type. Radius queries the operating system to determine if the user name and password supplied in the incoming packet are O.K.

`Crypt-Local`
The value of the `User-Password` attribute from the record is taken as an MD5 hash on the user's password. Radius generates MD5 hash on the supplied `User-Password` value and compares the two strings.

`Reject` Authentication fails.

`Accept` Authentication succeeds.

`SQL`
`Mysql` The MD5-encrypted user's password is queried from the SQL database (Section 6.6 [SQL Auth], page 73). `Mysql` is an alias maintained for compatibility with other versions of Radius.

`Pam` The user-name–password combination is checked using PAM.

13.3.6 Crypt-Password

```
ATTRIBUTE Crypt-Password 1006 string
```

Users:	`L-`
Hints:	`--`
Huntgroups:	`--`
Additivity:	Append
Proxy propagated:	No

This attribute is intended to be used in user's profile LHS. It specifies the MD5 hash of the user's password. When this attribute is present, `Auth-Type = Crypt-Local` is assumed. If both `Auth-Type` and `Crypt-Password` are present, the value of `Auth-Type` is ignored.

See Section 13.3.5 [Auth-Type], page 180.

13.3.7 Exec-Program-Wait

```
ATTRIBUTE Exec-Program-Wait 1039 string
```

Users:	`-R`
Hints:	`-R--`
Huntgroups:	`Replace`
Additivity:	No
Proxy propagated:	

When present in the RHS, the `Exec-Program-Wait` attribute specifies the program to be executed when the entry matches. If the attribute value string starts with vertical bar ('`|`'), then the attribute specifies the filter program to be used. If it starts with slash ('`/`'), then it is understood as the full pathname and arguments for the external program to be executed. Using any other character as the start of this string results in error.

13.3.7.1 Running an External Program

The command line can reference any attributes from both check and reply pairlists using attribute macros see Section 4.14 [Macro Substitution], page 64.

Before the execution of the program, `radiusd` switches to uid and gid of the user `daemon` and the group `daemon`. You can override these defaults by setting the variable `exec-program-user` in the configuration file to a proper value. See Section 4.1.1 [The option statement], page 22.

The daemon will wait until the program terminates. The return value of its execution determines whether the entry matches. If the program exits with a nonzero code, then the match fails. If it exits with a zero code, the match succeeds. In this case the standard output of the program is read and parsed as if it were a pairlist. The attributes thus obtained are added to the entry's reply attributes.

Example.

Suppose the '`users`' file contains the following entry:

```
DEFAULT Auth-Type = System,
                Simultaneous-Use = 1
        Exec-Program-Wait = "/usr/local/sbin/telauth \
                            %C{User-Name} \
                            %C{Calling-Station-Id}"
```

Then, upon successful matching, the program '`/usr/local/sbin/telauth`' will be executed. It will get as its arguments the values of the `User-Name` and `Calling-Station-Id` attributes from the request pairs.

The '`/usr/local/sbin/telauth`' can, for example, contain the following:

```
#! /bin/sh

DB=/var/db/userlist

if grep "$1:$2" $DB; then
    echo "Service-Type = Login,"
    echo "Session-Timeout = 1200"
    exit 0
else
    echo "Reply-Message = \
          \"You are not authorized to log in\""
    exit 1
fi
```

It is assumed that '`/var/db/userlist`' contains a list of `username:caller-id` pairs for those users that are authorized to use login service.

13.3.7.2 Using an External Filter

If the value of `Exec-Program-Wait` attribute begins with '`|`', `radiusd` strips this character from the value and uses the resulting string as a name of the

predefined external filter. Such filter must be declared in '`raddb/config`' (see Section 4.1.10 [filters], page 38).

Example.

Let the '`users`' file contain the following entry:

```
DEFAULT Auth-Type = System,
                Simultaneous-Use = 1
        Exec-Program-Wait = "|myfilter"
```

and let the '`raddb/config`' contain the following[1]:

```
filters {
    filter myfilter {
        exec-path "/usr/libexec/myfilter";
        error-log "myfilter.log";
        auth {
            input-format "%C{User-Name}
                          %C{Calling-Station-Id}";
            wait-reply yes;
        };
    };
};
```

Then, upon successful authentication, the program `/usr/libexec/myfilter` will be invoked, if it hasn't already been started for this thread. Any output it sends to its standard error will be redirected to the file '`myfilter.log`' in the current logging directory. A string consisting of the user's login name and his calling station ID followed by a newline will be sent to the program.

The following is a sample `/usr/libexec/myfilter` written in the shell:

```
#! /bin/sh

DB=/var/db/userlist

while read NAME CLID
do
    if grep "$1:$2" $DB; then
        echo "0 Service-Type = Login, Session-Timeout = 1200"
    else
        echo "1 Reply-Message = \
              \"You are not authorized to log in\""
    fi
done
```

13.3.8 Exec-Program

```
ATTRIBUTE Exec-Program 1038 string
```

Users: -R
Hints: --

[1] In this example the `input-format` statement has been split on two lines to fit the page width. It must occupy a *single line* in the real configuration file.

Huntgroups: --
Additivity: Replace
Proxy propagated: No

When present in the RHS, the `Exec-Program` attribute specifies the full pathname and arguments for the program to be executed when the entry matches.

The command line can reference any attributes from both check and reply pairlists, using attribute macros (see Section 4.14 [Macro Substitution], page 64).

Before the execution of the program, `radiusd` switches to the uid and gid of the user `daemon` and the group `daemon`. You can override these defaults by setting variables `exec-program-user` and `exec-program-group` in configuration file to proper values Section 4.1.1 [The option statement], page 22.

The daemon does not wait for the process to terminate.

Example

Suppose the '`users`' file contains the following entry:

```
DEFAULT Auth-Type = System,
                Simultaneous-Use = 1
        Exec-Program = "/usr/local/sbin/logauth \
                        %C{User-Name} \
                        %C{Calling-Station-Id}"
```

Then, upon successful matching, the program '`/usr/local/sbin/logauth`' will be executed. It will get as its arguments the values of the `User-Name` and `Calling-Station-Id` attributes from the request pairs.

13.3.9 `Fall-Through`

```
ATTRIBUTE Fall-Through 1036 integer
```

Users: LR
Hints: LR
Huntgroups: --
Additivity: Append
Proxy propagated: No

```
VALUE      Fall-Through      No                  0
VALUE      Fall-Through      Yes                 1
```

The `Fall-Through` attribute should be used in the reply list. If its value is set to `Yes` in a particular record, that tells Radius to continue looking up other records even when the record at hand matches the request. It can be used to provide default values for several profiles.

Consider the following example. Let's suppose the '`users`' file contains the following:

```
johns   Auth-Type = SQL
                Framed-IP-Address = 11.10.10.251,
```

```
                Fall-Through = Yes

smith   Auth-Type = SQL
                Framed-IP-Address = 11.10.10.252,
                Fall-Through = Yes

DEFAULT NAS-IP-Address = 11.10.10.1
        Service-Type = Framed-User,
                Framed-Protocol = PPP
```

Then after successful matching of a particular user's record, the matching will continue until it finds the `DEFAULT` entry, which will add its RHS to the reply pairs for this request. The effect is that, if user '`johns`' authenticates successfully she gets the following reply pairs:

```
Service-Type = Framed-User,
Framed-Protocol = PPP,
Framed-IP-Address = 11.10.10.251
```

whereas user `smith` gets

```
Service-Type = Framed-User,
Framed-Protocol = PPP,
Framed-IP-Address = 11.10.10.252
```

Note that the attribute `Fall-Through` itself is never returned to the NAS.

13.3.10 Group

```
ATTRIBUTE Group 1005 string
```

Users:	L-
Hints:	L-
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

13.3.11 Hint

```
ATTRIBUTE Hint 1040 string
```

Users:	L-
Hints:	-R
Huntgroups:	-R
Additivity:	Append
Proxy propagated:	No

Use the `Hint` attribute to specify additional matching criteria depending on the hint (see Section 4.6 [hints file], page 50).

Let the '`hints`' file contain

```
DEFAULT         Prefix = "S", Strip-User-Name = No
                Hint = "SLIP"
```

and the '`users`' file contain

```
DEFAULT Hint = "SLIP",
                NAS-IP-Address = 11.10.10.12,
```

```
        Auth-Type = System
Service-Type = Framed-User,
        Framed-Protocol = SLIP
```

Then any user having a valid system account and coming from NAS '11.10.10.12' will be provided SLIP service if his user name starts with 'S'.

13.3.12 Huntgroup-Name

```
ATTRIBUTE Huntgroup-Name 221 string
```

Users:	L-
Hints:	-R
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

The **Huntgroup-Name** can be used either in the LHS of the '**users**' file record or in the RHS of the '**huntgroups**' file record.

When encountered in a LHS of a particular '**users**' profile, this attribute indicates the huntgroup name to be matched. Radius looks up the corresponding record in the '**huntgroups**' file. If such a record is found, each A/V pair from its reply list is compared against the corresponding pair from the request being processed. The request matches only if it contains all the attributes from the specified huntgroup, and their values satisfy the conditions listed in the huntgroup pairs.

For example, suppose that the authentication request contains the following attributes:

```
User-Name = "john",
User-Password = "guess",
NAS-IP-Address = 10.11.11.1,
NAS-Port-Id = 24
```

Let us further suppose that the '**users**' file contains the following entry:

```
john    Huntgroup-Name = "users_group",
                Auth-Type = System
        Service-Type = Login
```

and, finally, '**huntgroups**' contains the following entry:

```
users_group     NAS-IP-Address = 10.11.11.1
                NAS-Port-Id < 32
```

Then the authentication request will succeed, since it contains **NAS-Port-Id** attribute and its value is less than 32.

See Section 4.7 [huntgroups file], page 50.

13.3.13 Log-Mode-Mask

```
ATTRIBUTE Log-Mode-Mask 2007 integer
```

Users:	L-
Hints:	-R

Huntgroups: -R
Additivity: Append
Proxy propagated: N/A

```
VALUE    Log-Mode-Mask    Log-Auth          1
VALUE    Log-Mode-Mask    Log-Auth-Pass     2
VALUE    Log-Mode-Mask    Log-Failed-Pass   4
VALUE    Log-Mode-Mask    Log-Pass          6
VALUE    Log-Mode-Mask    Log-All           7
```

`Log-Mode-Mask` is used to control the verbosity of authentication log messages for given user or class of users. The meaning of its values is:

`Log-Auth` Do not log successful authentications.

`Log-Auth-Pass`
: Do not show the password with the log message from a successful authentication.

`Log-Failed-Pass`
: Do not show a failed password.

`Log-Pass` Do not show a plaintext password, either failed or succeeded.

`Log-All` Do not log authentications at all.

Technical details: After authentication, the server collects all `Log-Mode-Mask` attributes from the incoming request and LHS of the user's entry. The values of these attributes ORed together form a mask, which is applied via an XOR operation to the current log mode. The value thus obtained is used as effective log mode.

13.3.14 Login-Time

```
ATTRIBUTE Login-Time 1042 string
```

Users: L-
Hints: --
Huntgroups: --
Additivity: Append
Proxy propagated: No

The `Login-Time` attribute specifies the time range over which the user is allowed to log in. The attribute should be specified in the LHS.

The format of the `Login-Time` string is the same as that of UUCP time ranges. The following description of the time range format is adopted from the documentation for the Taylor UUCP package:

A time string may be a list of simple time strings separated with vertical bars '`|`' or commas '`,`'.

Each simple time string must begin either with a day-of-week abbreviation (one of '`Su`', '`Mo`', '`Tu`', '`We`', '`Th`', '`Fr`', '`Sa`'), or '`Wk`' for any day from Monday to Friday inclusive, or '`Any`' or '`Al`' for any day.

Following the day may be a range of hours separated with a hyphen, using 24-hour time. The range of hours may cross 0; for example '`2300-0700`'

means any time except 7 AM to 11 PM. If no time is given, calls may be made at any time on the specified day(s).

The time string may also be the single word '`Never`', which does not match any time.

Here are a few sample time strings with an explanation of what they mean.

'`Wk2305-0855,Sa,Su2305-1655`'
: This means weekdays before 8:55 AM or after 11:05 PM, any time Saturday, or Sunday before 4:55 PM or after 11:05 PM. These are approximately the times during which night rates apply to phone calls in the U.S.A. Note that this time string uses, for example, '`2305`' rather than '`2300`'; this will ensure a cheap rate even if the computer clock is running up to five minutes ahead of the real time.

'`Wk0905-2255,Su1705-2255`'
: This means weekdays from 9:05 AM to 10:55 PM, or Sunday from 5:05 PM to 10:55 PM. This is approximately the opposite of the previous example.

'`Any`'
: This means any day. Since no time is specified, it means any time on any day.

13.3.15 `Match-Profile`

```
ATTRIBUTE Match-Profile 2004 string
```

Users:	`LR`
Hints:	`-R`
Huntgroups:	`-R`
Additivity:	Append
Proxy propagated:	No

The `Match-Profile` attribute can be used in LHS and RHS lists of a user profile. Its value is the name of another user's profile (target profile). When `Match-Profile` is used in the LHS, the incoming packet will match this profile only if it matches the target profile. In this case the reply pairs will be formed by concatenating the RHS lists from both profiles. When used in the RHS, this attribute causes the reply pairs from the target profile to be appended to the reply from the current profile if the target profile matches the incoming request.

For example:

```
IPPOOL  NAS-IP-Address = 10.10.10.1
                Framed-Protocol = PPP,
                Framed-IP-Address = "10.10.10.2"

IPPOOL  NAS-IP-Address = 10.10.11.1
                Framed-Protocol = PPP,
```

```
                Framed-IP-Address = "10.10.11.2"

guest   Auth-Type = SQL
                Service-Type = Framed-User,
        Match-Profile = IPPOOL
```

In this example, when user `guest` comes from NAS `10.10.10.1`, he is assigned IP `10.10.10.2`, otherwise if he is coming from NAS `10.10.11.1` he is assigned IP `10.10.11.2`.

13.3.16 Menu

```
ATTRIBUTE Menu 1001 string
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	No

This attribute should be used in the RHS. If it is used, it should be the only reply item.

The `Menu` attribute specifies the name of the menu to be presented to the user. The corresponding menu code is looked up in the '`RADIUS_DIR/menus/`' directory (see Section 4.13 [menus directory], page 62).

13.3.17 Pam-Auth

```
ATTRIBUTE Pam-Auth 1041 string
```

Users:	L-
Hints:	-R
Huntgroups:	-R
Additivity:	Append
Proxy propagated:	No

The `Pam-Auth` attribute can be used in conjunction with

```
Auth-Type = Pam
```

to supply the PAM service name instead of the default '`radius`'. It is ignored if `Auth-Type` attribute is not set to `Pam`.

13.3.18 Prefix

```
ATTRIBUTE Prefix 1003 string
```

Users:	L-
Hints:	L-
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

The `Prefix` attribute indicates the prefix that the user name should contain in order for a particular record in the profile to be matched. This attribute should be specified in the LHS of the '`users`' or '`hints`' file.

For example, if the ‘users’ file contained

```
DEFAULT Prefix = "U", Auth-Type = System
                Service-Type = Login-User
```

then the user names ‘Ugray’ and ‘Uyoda’ would match this record, whereas ‘gray’ and ‘yoda’ would not.

Both Prefix and Suffix attributes may be specified in a profile. In this case the record is matched only if the user name contains both the prefix and the suffix specified.

See Section 13.3.27 [Suffix], page 193, and Section 13.3.26 [Strip-User-Name], page 192.

13.3.19 Proxy-Replied

```
ATTRIBUTE Proxy-Replied 2012 integer
```

Users:	L-
Hints:	L-
Huntgroups:	L-
Additivity:	Replace
Proxy propagated:	N/A

```
VALUE      Proxy-Replied     No                  0
VALUE      Proxy-Replied     Yes                 1
```

radiusd adds this attribute to the incoming request if it was already processed by a remote radius server.

13.3.20 Realm-Name

(This message will disappear, once this node revised.)

```
ATTRIBUTE Realm-Name 2013 string
```

Users:	L-
Hints:	L-
Huntgroups:	L-
Additivity:	Append
Proxy propagated:	No

13.3.21 Replace-User-Name

```
ATTRIBUTE Replace-User-Name 2001 string
```

Users:	LR
Hints:	LR
Huntgroups:	--
Additivity:	Append
Proxy propagated:	No

```
VALUE      Replace-User-Name  No                  0
VALUE      Replace-User-Name  Yes                 1
```

Use this attribute to modify the user name from the incoming packet. The Replace-User-Name can reference any attributes from both LHS and RHS pairlists using attribute macros (Section 4.14 [Macro Substitution], page 64).

For example, the '`users`' entry

```
guest   NAS-IP-Address = 11.10.10.11,
                Calling-Station-Id != ""
                Auth-Type = Accept
        Replace-User-Name = "guest#%C{Calling-Station-Id}",
                Service-Type = Framed-User,
                Framed-Protocol = PPP
```

allows the use of PPP service for user name `guest`, coming from NAS '`11.10.10.11`' with a nonempty `Calling-Station-Id` attribute. A string consisting of a '`#`' character followed by the `Calling-Station-Id` value is appended to the user name.

13.3.22 `Rewrite-Function`

```
ATTRIBUTE Rewrite-Function 2004 string
```

Users:	LR
Hints:	LR
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

The `Rewrite-Function` attribute specifies the name of the rewriting function to be applied to the request. The attribute may be specified in either pairlist in the entries of the '`hints`' or '`huntgroups`' configuration file.

The corresponding function should be defined in '`rewrite`' as

```
integer name()
```

i.e., it should return an integer value and should not take any arguments.

See Section 4.12 [Packet rewriting rules], page 62, Section 4.6 [hints file], page 50; Section 4.7 [huntgroups file], page 50.

13.3.23 `Scheme-Acct-Procedure`

```
ATTRIBUTE Scheme-Acct-Procedure 2010 string
```

Users:	--
Hints:	-R
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	N/A

The `Scheme-Acct-Procedure` attribute is used to set the name of the Scheme accounting procedure. See Section 10.3.3 [Accounting with Scheme], page 117, for information about how to write Scheme accounting procedures.

13.3.24 `Scheme-Procedure`

```
ATTRIBUTE Scheme-Procedure 2009 string
```

Users:	-R
Hints:	-R--
Huntgroups:	Append

Additivity: N/A
Proxy propagated:

The `Scheme-Procedure` attribute is used to set the name of the Scheme authentication procedure. See Section 10.3.2 [Authentication with Scheme], page 116, for information about how to write Scheme authentication procedures.

13.3.25 Simultaneous-Use

```
ATTRIBUTE Simultaneous-Use 1034 integer
```

Users: L-
Hints: -R
Huntgroups: -R
Additivity: Append
Proxy propagated: No

This attribute specifies the maximum number of simultaneous logins a given user is permitted to have. When the user is logged in this number of times, any further attempts to log in are rejected.

See Section 6.9 [Multiple Login Checking], page 74.

13.3.26 Strip-User-Name

```
ATTRIBUTE Strip-User-Name 1035 integer
```

Users: LR
Hints: LR
Huntgroups: -R
Additivity: Append
Proxy propagated: No

```
VALUE      Strip-User-Name   No                  0
VALUE      Strip-User-Name   Yes                 1
```

The value of `Strip-User-Name` indicates whether Radius should strip any prefixes/suffixes specified in the user's profile from the user name. When it is set to `Yes`, the user names will be logged and accounted without any prefixes or suffixes.

A user may have several user names for different kind of services. In this case differentiating the user names by their prefixes and stripping them off before accounting would help keep accounting records consistent.

For example, let's suppose the '`users`' file contains

```
DEFAULT Suffix = ".ppp",
                Strip-User-Name = Yes,
                Auth-Type = SQL
        Service-Type = Framed-User,
                Framed-Protocol = PPP

DEFAULT Suffix = ".slip",
                Strip-User-Name = Yes,
                Auth-Type = SQL
```

```
        Service-Type = Framed-User,
                Framed-Protocol = SLIP
```

Now, user '`johns`', having a valid account in the SQL database, logs in as '`johns.ppp`'. She then is provided the PPP service, and her PPP session is accounted under user name '`johns`'. Later on, she logs in as '`johns.slip`'. In this case she is provided the SLIP service and again her session is accounted under her real user name '`johns`'.

13.3.27 Suffix

```
ATTRIBUTE Suffix 1004 string
```

Users:	L-
Hints:	L-
Huntgroups:	LR
Additivity:	Append
Proxy propagated:	No

The `Suffix` attribute indicates the suffix that the user name should contain in order for a particular record in the profile to be matched. This attribute should be specified in LHS of the '`users`' or '`hints`' file.

For example, if the '`users`' file contained

```
DEFAULT Suffix = ".ppp", Auth-Type = System,
                Strip-User-Name = Yes
        Service-Type = Framed-User,
                Framed-Protocol = PPP
```

then the user names '`gray.ppp`' and '`yoda.ppp`' would match this record, whereas '`gray`' and '`yoda`' would not.

Both `Prefix` and `Suffix` attributes may be specified in a profile. In this case the record is matched only if the user name contains both the prefix and the suffix specified.

See Section 13.3.18 [Prefix], page 189, and Section 13.3.26 [Strip-User-Name], page 192.

13.3.28 Termination-Menu

```
ATTRIBUTE Termination-Menu 1002 string
```

Users:	-R
Hints:	--
Huntgroups:	--
Additivity:	Replace
Proxy propagated:	No

This attribute should be used in the RHS. If it is used, it should be the only reply item.

The `Termination-Menu` specifies the name of the menu file to be presented to the user after finishing his session. The corresponding menu code is looked up in the '`RADIUS_DIR/menus/`' directory (see Section 4.13 [menus directory], page 62).

14 Reporting Bugs

It is possible you will encounter a bug in one of the Radius programs. If this happens, we would like to hear about it. As the purpose of bug reporting is to improve software, please be sure to include maximum information when reporting a bug. The information needed is:

- Conditions under which the bug appears.
- Version of the package you are using.
- Compilation options used when configuring the package.
- If the bug is found in `radiusd` daemon, run '`radiusd -v`' and include the output it produces.
- Contents of Radius configuration directory ('`/usr/local/etc/raddb`' or whatever you have set it to while configuring).
- Log messages produced.

Send your report to `bug-gnu-radius@gnu.org`. Allow us a couple of days to answer.

15 Where to Get Information about GNU Radius

The two places to look for news regarding GNU Radius are the Radius homepage at `http://www.gnu.org/software/radius` and the Radius project page at `http://savannah.gnu.org/projects/radius`.

The following mailing lists are related to GNU Radius:

`info-gnu-radius@gnu.org`
: This list distributes announcements and progress reports on GNU Radius. This is a moderated list. Please do not send bug reports or requests for help to this list; there exist special mailing lists for these purposes. To subscribe to the list, visit `http://mail.gnu.org/mailman/listinfo/info-gnu-radius`.

`help-gnu-radius@gnu.org`
: This list is the place for users and installers of GNU Radius to ask for help. The list is not moderated, but postings are allowed for list members only. To subscribe to the list, visit `http://mail.gnu.org/mailman/listinfo/help-gnu-radius`.

`bug-gnu-radius@gnu.org`
: This list distributes bug reports, bug fixes, and suggestions for improvements in Radius. User discussion of Radius bugs also occurs here. The list is not moderated; postings are allowed for anybody. To subscribe to the list, visit `http://mail.gnu.org/mailman/listinfo/bug-gnu-radius`.

How to Obtain Radius

GNU Radius is *free software*; this means that everyone is free to use it and free to redistribute it on certain conditions. GNU Radius is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GNU Radius that they might get from you. The precise conditions are found in the GNU General Public License that comes with Radius and also appears following this section.

One way to get a copy of GNU Radius is from someone else who has it. You need not ask for our permission to do so, or tell any one else; just copy it. If you have access to the Internet, you can get the latest distribution version of GNU Radius by anonymous FTP. It is available at `ftp://ftp.gnu.org/pub/gnu/radius`

Radius Glossary

Throughout this document the following terms are used:

RADIUS (small capitals)
: The Remote Authentication Dial In User Service protocol as described in RFC 2138, 2865, and 2866.

NAS
: A network access server, that is, a computer or a special device designed to provide access to the network. For example, it can be a computer connected to the network and equipped with several modems. Such a NAS will allow a user connecting to one of its modems to access the network.

Service
: A service, such as PPP, SLIP, or telnet, provided to a user by the NAS.

Session
: Each instance of a service. Sessions start when the service is first provided and close when the service is ended. A user may be allowed to have multiple sessions active simultaneously.

Session ID
: The session identifier: a string of characters uniquely identifying the session.

A/V pair
: Attribute-value pair: see Section 2.1 [Attributes], page 7.

Dial-in or dial-up user
: A user connecting to a service through the modem line.

User database
: A database where a RADIUS server keeps information about users, their authentication information, etc.

User's profile
: A record in the user database describing a particular user for purposes of authentication and authorization, i.e., how the user should be authenticated as well as which services he is allowed to be provided and parameters of these services.

Acknowledgements

I would like to acknowledge Oswaldo Aguirre and Francisco Obispo, who invested a lot of time and effort to debug and test the program. They also wrote `web-radact` — a web interface to the radius database.

Alexandre Oliva provided a lot of good advice and offered valuable help in testing Radius on various platforms.

The following people provided many helpful comments, bug reports and patches: Dustin Mitchell, Jody Owens, Andrey Y. Mosienko, Oleg Gawriloff, Adrian P. van Bloois, Michael Samuel, Michael Smirnov, Andrey Pavlenko, Michael Weiser, Eric Salomé, Clement Gerouville, Dave Restall, Vlad Lungu, Robert Abbate, Jaime Tellez Sanchez, Cornel Cristea, Krzysztof Kopera, and David Friedman.

Additional people need to be thanked for their assistance in producing this manual. Lisa M. Goldstein coordinated its preparation and Joseph C. Fineman and Daniel Barowy did a remarkable job of editing.

And of course, thanks to Richard M. Stallman for founding the FSF and starting the GNU project.

Appendix A New Configuration Approach (draft)

(This message will disappear, once this node revised.)

This document presents a draft describing new approach for processing RADIUS requests. It is intended as a *request for comments*, and, in the long run, as a guide for GNU Radius developers. In its current state it is far from being complete. Please check `http://www.gnu.org/software/radius/manual` for updated versions. Feel free to send your comments and suggestions to `bug-gnu-radius@gnu.org`.

A.1 A brief description of Currently Used Approach

When I started to write GNU Radius, back in 1998, I had two major aims. The first and primary aim was to create a flexible and robust system that would follow the principle of Jon Postel:

> Be liberal in what you accept and conservative in what you send.

This, I believe, is the main principle of any good software for Internet.

The second aim was to be backward compatible with the implementations that already existed back then. This seemed to be important (and the time has proved it was), because it would allow users to easily switch from older radius daemon to GNU Radius.

An important part of every complex program is its configuration file. Traditional implementations of RADIUS servers (beginning from Livingston Radius) used a configuration suite consisting of several files, usually located in '`/etc/raddb`' subdirectory. Its main components were:

'`dictionary`'
: A file containing translations of symbolic names of radius attributes and attribute values to their integer numbers as specified by RADIUS protocol.

'`hints`'
: This file was intended to separate incoming requests in groups, based on the form of their login name. Traditionally such separation was performed basing on common *prefixes* and/or *suffixes* of login names.

'`huntgroups`'
: The purpose of this file was to separate incoming requests depending on their source, i.e. on the NAS that sent them and the port number on that NAS. It also served as a sort of simplified *access control list*.

'`users`'
: This file contained a users database. It described criteria for authentication and *reply pairs* to be sent back to requesting NASes.

Among these files, the first two were used for requests of any kind, whereas '`users`' was used only for `Access-Request` packets.

Though this configuration system suffered from many inconsistencies, the *second aim* required GNU Radius to use this approach.

To compensate for its deficiencies and to fulfill the *first aim*, this configuration system was extended, while preserving its main functionality. A number of additional *internal attributes* were added, that control `radiusd` behavior. A new language was created whose main purpose was to modify incoming requests (see Section 10.2 [Rewrite], page 98). The support for *GNU's Ubiquitous Intelligent Language for Extensions* (see Section 10.3 [Guile], page 115) was added, that allowed to further extend GNU Radius functionality.

The present operation model[1] of GNU Radius and its configuration file system[2] emerged as a result of the two development aims described above. Since 1998 up to present, GNU Radius users contributed a lot of ideas and code to the further development of the system.

However, it became obvious that this system presents strong obstacles to the further development. The next section addresses its deficiencies.

A.2 Deficiencies of Current Operation Model and Configuration Suite

The main deficiencies are inherited with the traditional configuration file suite. The rules for processing each request are split among three files, each of which is processed differently, despite of their external similarity. The administrator has to keep in mind a set of exotic rules when configuring the system[3]. When matching incoming requests with configuration file entries (*LHS*, see Section 2.3 [Matching Rule], page 11), some attributes are taken verbatim, whereas others are used to control `radiusd` behavior and to pass additional data to other rules (see Section 13.3 [Radius Internal Attributes], page 178). The things become even more complicated when RADIUS realms come into play (see Section 2.4.2.1 [Proxy Service], page 13). Some attributes are meaningful only if used in a certain part of a certain configuration file rule.

So, while being a lot more flexible than the approach used by other RADIUS implementations, the current system is quite difficult to maintain.

Another deficiency is little control over actions executed on different events. For example, it is often asked how can one block a user account after a predefined number of authentication failures? Currently this can only

[1] See Chapter 2 [Operation], page 7.

[2] See Chapter 4 [Configuration Files], page 21.

[3] '`Hints`' is processed for each request... Authentication requests first pass '`hints`', then '`huntgroups`', then '`users`'... Accounting requests use only '`hints`' and '`huntgroups`'... '`Huntgroups`' entries may also be used (sometimes inadvertently) to create ACL rules, etc, etc...

be done by writing an external authentication procedure (either in Scheme, using Guile, or as a standalone executable, using `Exec-Program-Wait`). The proper solution would be to have a set of user-defined triggers for every RADIUS event (in this case, for authentication failure).

Another commonly asked question is how to make `radiusd` execute several SQL queries when processing a request. While GNU Radius is not supposed to compensate for deficiencies of some SQL implementations that do not allow for nested queries, such a feature could come quite handy.

A.3 Proposed Solution

(This message will disappear, once this node revised.)

Processing of incoming requests is controlled by *request-processing program*. Request-processing program is a list-like structure, consisting of *instructions*.

A.3.1 Request-processing Instruction

Request-processing program consists of *instructions*. There are seven basic instruction types:

`grad_instr_conditional_t`
: This instruction marks a branch point within the program.

`grad_instr_call_t`
: Represents a *call* of a subprogram

`grad_instr_action_t`
: Invokes a Rewrite function

`grad_instr_proxy_t`
: Proxies a request to the remote server

`grad_instr_forward_t`
: Forwards a request to the remote server

`grad_instr_reply_t`
: Replies back to the requesting NAS.

Consequently, an instruction is defined as a union of the above node types:

`grad_instr_t` [Instruction]

```
enum grad_instr_type
{
  grad_instr_conditional,
  grad_instr_call,
  grad_instr_return,
  grad_instr_action,
  grad_instr_reply,
  grad_instr_proxy,
  grad_instr_forward
};

typedef struct grad_instr grad_instr_t;

struct grad_instr
{
  enum grad_instr_type type;
  grad_instr_t *next;
  union
    {
      grad_instr_conditional_t cond;
      grad_instr_call_t call;
      grad_instr_action_t action;
      grad_instr_reply_t reply;
      grad_instr_proxy_t proxy;
      grad_instr_forward_t forward;
    } v;
};
```

`Type` member contains type of the instruction. The evaluator uses `type` to determine which part of `union v`, holds instruction-specific data.

`Next` points to the next instruction. The evaluator will go to this instruction unless the present one changes the control flow.

Finally, `v` contains instruction-specific data. These will be discussed in the following subsections.

A.3.2 grad_instr_conditional

(This message will disappear, once this node revised.)

`grad_instr_conditional_t` *cond iftrue iffalse* [Instruction]

```
struct grad_instr_conditional
{
  grad_entry_point_t cond;  /* Entry point to the compiled
                               Rewrite condition */
  grad_instr_t *iftrue;     /* Points to the ``true'' branch  */
  grad_instr_t *iffalse;    /* Points to the ``false'' branch  */
};
typedef struct grad_instr_conditional grad_instr_conditional_t;
```

Instructions of type `grad_instr_conditional_t` indicate branching. Upon encountering an `grad_instr_conditional_t`, the engine executes a Rewrite expression pointed to by `cond`. If the expression evaluates to

`true`, execution branches to instruction `iftrue`. Otherwise, if `iffalse` is not `NULL`, execution branches to that instruction. Otherwise, the control flow passes to `grad_instr_t.next`, as described in the previous section.

RPL representation

`COND` *`expr if-true`* [*`if-false`*] [RPL defun]

expr Textual representation of Rewrite conditional expression or its entry point.

if-true RPL expression executed if *expr* evaluates to `t`.

if-true Optional RPL expression that is executed if *expr* evaluates to `nil`.

Example

`COND` with two arguments:

```
(COND "%[User-Name] ~= \"test-.*\""
      (REPLY Access-Reject ("Reply-Message" . "Test accounts disabled")))
```

`COND` with three arguments:

```
(COND "%[Hint] == "PPP" && authorize(PAM)"
      (REPLY Access-Accept
             ("Service-Type" . "Framed-User")
             ("Framed-Protocol" . "PPP"))
      (REPLY Access-Reject
             ("Reply-Message" . "Access Denied")))
```

A.3.3 grad_instr_call

(This message will disappear, once this node revised.)

`grad_instr_call_t` *entry* [Instruction]

```
struct grad_instr_call {
       grad_instr_t *entry;
};
typedef struct grad_instr_call grad_instr_call_t;
```

Instructions of type `grad_instr_call` instruct the engine to *call* the given subprogram. The engine pushes the current instruction to the return point stack and branches to instruction `entry`. Execution of the subprogram ends when the engine encounters an instruction of one of the following types: `grad_instr_return`, `grad_instr_reply` or `grad_instr_proxy`.

If `grad_instr_return` is encountered, the engine pops the instruction from the top of the return point stack and makes it current instruction, then it branches to the `next` node.

If `grad_instr_reply` or `grad_instr_proxy` is encountered, the engine, after executing corresponding actions, finishes executing the program.

RPL representation

`CALL` *`list`* [RPL defun]
`CALL` *`defun-name`* [RPL defun]

In the first form, the argument *list* is the RPL subprogram to be executed.

In the second form *defun-name* is a name of the RPL subprogram defined by `defun`.

Examples

First form:

```
(CALL (ACTION "myfun(%[User-Name])")
      (REPLY Access-Reject
             ("Reply-Message" . "Access Denied")))
```

Second form:

```
(CALL process_users)
```

A.3.4 grad_instr_return

(This message will disappear, once this node revised.)

An instruction of type `grad_instr_return` indicates a return point from the subprogram. If encountered in a subprogram (i.e. a program entered by `grad_instr_call` node), it indicates return to the calling subprogram (see the previous subsection). Otherwise, if `grad_instr_return` is encountered within the main trunk, it ends evaluating of the program.

Instructions of this type have no data associated with them in union `v`.

RPL representation

`RETURN` [RPL defun]

Examples

```
(RETURN)
```

A.3.5 grad_instr_action

(This message will disappear, once this node revised.)

`grad_instr_reply_t` *expr* [Instruction]

```
struct grad_instr_action {
       grad_entry_point_t expr;    /* Entry point to the compiled
                                      Rewrite expression */
};
typedef struct grad_instr_action grad_instr_reply_t;
```

The machine executes a Rewrite expression with entry point `expr`. Any return value from the expression is ignored.

RPL representation

`ACTION` *`expr`* [RPL defun]
`ACTION` *`entry-point`* [RPL defun]

Examples

```
(ACTION "%[NAS-IP-Address] = request_source_ip()")
```

A.3.6 grad_instr_reply

(This message will disappear, once this node revised.)

`grad_instr_reply_t` *return_code* [Instruction]

```
struct grad_instr_reply {
       u_char reply_code;         /* Radius request code */
};
typedef struct grad_instr_reply grad_instr_reply_t;
```

`grad_instr_reply` instructs `radiusd` to send to the requesting NAS a reply with code `reply_code`. Any reply pairs collected while executing the program are attached to the reply.

After executing `grad_instr_reply` instruction, the engine stops executing of the program.

Any execution path will usually end with this instruction.

RPL representation

`REPLY` *`reply-code`* [*`attr-list`*] [RPL defun]

Arguments:

reply-code Radius reply code.

attr-list List of A/V pairs to be added to the reply. Each A/V pair is represented as a `cons`: (*`name-or-number . value`*).

Example

```
(REPLY Access-Accept
       ("Service-Type" . "Framed-User")
       ("Framed-Protocol" . "PPP"))
```

A.3.7 grad_instr_proxy

(This message will disappear, once this node revised.)

`grad_instr_proxy_t` *realm* [Instruction]

```
struct grad_instr_proxy
{
  grad_realm_t realm;
};
typedef struct grad_instr_proxy grad_instr_proxy_t;
```

This instruction tells radius to proxy the request to the server defined in `realm`. In other words, the engine executes `proxy_send`. Further processing of the program is stopped.

RPL representation

`PROXY` *`realm-name`* [RPL defun]

Realm-name is name of the realm as defined in '`raddb/realms`'.

Examples

.

A.3.8 grad_instr_forward

(This message will disappear, once this node revised.)

`grad_instr_forward_t` *server_list* [Instruction]

```
struct grad_instr_forward
{
  grad_list_t server_list;
};
typedef struct grad_instr_forward grad_instr_forward_t;
```

This node *forwards* the request to each servers from `server_list`. Forwarding differs from proxying in that the requests are sent to the remote servers *and* processed locally. The remote server is not expected to reply. See Section 4.1.3 [auth], page 28, for more information on this subject.

In contrast to `grad_instr_proxy`, this instruction type does not cause the execution to stop.

Elements of `server_list` are of type `grad_server_t`.

Currently forwarding is performed by `forward_request` function ('`forward.c`'), which could be used with little modifications. Namely, it will be rewritten to get server list as argument, instead of using static variable `forward_list`. Consequently, the functions responsible for creating and initializing this static variable will disappear along with the variable itself. .

A.3.9 RPL representation

`FORWARD` *`server-list`* [RPL defun]

A.4 Changes to Rewrite Language

(This message will disappear, once this node revised.)

A.5 Support for Traditional Configuration Files.

(This message will disappear, once this node revised.)

Within the new configuration system, the traditional "trio" '`hints-huntgroups-users`' will be translated to the following program:

```
(defprog main
 (CALL hints)
 (CALL huntgroups)
 (COND "request_code() == Access-Request"
       (CALL users))
 (REPLY Access-Reject
        (Reply-Message . "\nAccess denied\n")))
```

For example, consider the following configuration:

```
# raddb/hints:
DEFAULT Prefix = "PPP" Hint = PPP
```

This will produce the following program:

```
(defprog hints
 (COND "%[Prefix] == \"PPP\"")
       (ACTION "%[Hint] = \"PPP\""))
```

```
#raddb/huntgroups
DEFAULT NAS-IP-Address = 10.10.4.1      Suffix = "staff"
DEFAULT NAS-IP-Address = 10.10.4.2      Huntgroup-Name = "second"
```

Will produce

```
(defprog huntgroups
 (COND "%[NAS-IP-Address] == 10.10.4.1 && !(%[Suffix] == \"staff\")"
       (REPLY Access-Reject
              ("Reply-Message" . "Access Denied by Huntgroup")))
 (COND "%[NAS-IP-Address] == 10.10.4.2"
       (ACTION "%[Huntgroup-Name] = \"second\"")))
```

Finally, ‘`users`’:

```
#raddb/users
DEFAULT Hint = "PPP",
               Auth-Type = PAM
        Service-Type = Framed-User,
               Framed-Protocol = PPP

DEFAULT Huntgroup-Name = "second",
               Auth-Type = PAM
        Service-Type = "Authenticate-Only",
               Reply-Message = "Authentity Confirmed"
```

will produce

```
(defprog users
 (COND "%[Hint] == "PPP" && authorize(PAM)"
       (REPLY Access-Accept
             (Service-Type . Framed-User)
             (Framed-Protocol . PPP))
       (REPLY Access-Reject
             (Reply-Message . "Access Denied")))
 (COND "%[Huntgroup-Name] == \"second\" && authorize(PAM)"
       (REPLY Access-Accept
              (Service-Type . "Authenticate-Only")
              (Reply-Message . "Authentity Confirmed"))))
```

A.6 New Configuration Files

(This message will disappear, once this node revised.)

Appendix B GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within

that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque

copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If

there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire

aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

$

$INCLUDE (dictionary) 40

%

%raddb-path 131

_

_ 114

A

A/V pair 7, 201
Accept Authentication Type 71
access-denied 37
'access.deny' file 53
account-closed 37
Accounting directory 5
Accounting requests 9
Accounting service parameters 30
Accounting Types 81
Accounting with Scheme 117
acct 38
acct statement 30
Acct-Authentic 175
Acct-Delay-Time 175
acct-dir 22
Acct-Ext-Program 178
acct-function-name 117
Acct-Input-Octets 176
Acct-Input-Packets 176
Acct-Output-Octets 176
Acct-Output-Packets 176
Acct-Session-Id 177
Acct-Session-Time 177
Acct-Status-Type 177
Acct-Terminate-Cause 178
Acct-Type 179
acl 32
ACTION 211
Additivity of an attribute 7
ALIAS 43
allow 32
Analyzing SNMP output 104
Attribute 7, 201
ATTRIBUTE 41
Attribute-value pair 7
Attribute-Value pair 201
auth 28, 38
Auth-Data 180
Auth-Failure-Trigger 179
auth-function 116
Auth-Type 180
auth_db 54
auth_failure_query 54
auth_query 54
auth_success_query 54
Authentication 16
authentication probes 76
Authentication requests 8
Authentication service parameters 28
Authentication with Scheme 116
avl-delete 118
avl-match? 118
avl-merge 118

B

BEGIN 42
BEGIN-VENDOR 42
break 154
buildbm 130
Built-in functions, Rewrite 111

C

CALL 210
Callback-Id 166
Callback-Number 166
Called-Station-Id 166
Calling-Station-Id 167
category 26
channel 26
CHAP 71
CHAP-Password 166
check 104
Checking UNIX finger output 104
checkrad-assume-logged 28
Class 167
clid 125
Client Configuration 135
Client Package 135
'client.conf' 135
'clients' file 45
common 38

`community` ... 32
`compare-atribute-flag` ... 28, 30
Comparing the requests ... 67
`COND` ... 209
Configuration directory ... 5
Configuration files (radiusd) ... 21
`continue` ... 154
Controlling Authentication Probes ... 76
`Crypt-Password` ... 181
Custom Accounting Types ... 83
Custom Authentication Types ... 73
Customizing accounting service ... 30
Customizing authentication server ... 28
Customizing Radiusd Guile interface ... 35
Customizing reply messages ... 37
Customizing SNMP server ... 32

D

`d` ... 93
Data directory ... 5
Data types, Rewrite ... 105
'`datadir`', directory for shared data files ... 5
DBM: enabling ... 32
`debug` ... 35, 93
Debugging ... 89
Declarations, Rewrite ... 108
`delay` ... 124
Deleting hung user sessions ... 127
`deny` ... 32
`detail` ... 28, 30
Detailed Request Accounting ... 81
`dgettext` ... 114
Dial-in user ... 201
Dial-up user ... 201
`dict-entry` ... 118
'`dictionary`' file ... 40
Disabling user accounts ... 71
`dngettext` ... 114
`doauth` ... 54
duplicate requests, checking ... 12
`duration` ... 124

E

Enabling DBM ... 32
Encrypted Password Authentication Type ... 72
`END` ... 42
`END-VENDOR` ... 42
`eval` ... 35
`Exec-Program` ... 183
`exec-program-user` ... 22
`Exec-Program-Wait` ... 181
`exit` ... 154
`expect` ... 155
`Extended comparison` ... 68
Extended Comparison ... 67
Extensions ... 95

F

`Fall-Through` ... 184
FDL, GNU Free Documentation License ... 215
`field` ... 112
`file` ... 26, 32
`filter` ... 38
`filters` ... 38
Filters ... 95
`FORWARD` ... 212
`framed-address` ... 124
`Framed-Compression` ... 167
`Framed-IP-Address` ... 168
`Framed-IP-Netmask` ... 168
`Framed-MTU` ... 168
`Framed-Protocol` ... 169
`Framed-Route` ... 169
`Framed-Routing` ... 169

G

`g` ... 91
`gc-interval` ... 35
`gecos` ... 124
`getopt` ... 153
`gettext` ... 114
`Group` ... 185
`group_query` ... 54
`gsub` ... 113
`Guest accounts, setting up` ... 71
`guile` ... 35, 91
Guile ... 115
Guile interface ... 131
Guile interface configuration ... 35
Guile, representation of Radius data ... 115

H

`Hint` ... 185
Hints ... 14, 50
`'hints' file` ... 50
`hook` ... 105

htonl ... 112
htons ... 112
Huntgroup-Name ... 186
Huntgroups ... 15, 50
'huntgroups' file ... 50

I

ident ... 32
Identifiers, Rewrite ... 108
Idle-Timeout ... 169
index ... 111
inet_aton ... 112
inet_ntoa ... 112
input ... 154
input-format ... 38
Invoking Scheme authentication function ... 117
Invoking the radius daemon ... 17
IP pools for MAX Ascend ... 105

L

Label, Matching Rule ... 11
length ... 111
level ... 26
LHS, Matching Rule ... 11
listen ... 28, 30
load ... 35
load-module ... 35
load-path ... 35
Local Password Auth ... 71
Log directory ... 5
log-dir ... 22
Log-Mode-Mask ... 186
logging ... 24
Logging ... 85
Logging category ... 26
Logging channel ... 26
Logging hook ... 25
logging statement ... 27
Logging, 'config' statement ... 24
login ... 124
Login verification functions ... 104
Login-Time ... 187
logit ... 112

M

master-read-timeout ... 22
master-write-timeout ... 22
Match-Profile ... 188
Matching Rule ... 11
MAX Ascend, broken passwords ... 45
max-nas-count ... 32
max-port-count ... 32
max-processes ... 22
max-requests ... 22, 28, 30, 32
Menu ... 189
menu, syntax ... 62
'menus' file ... 62
'menus', configuration subdirectory ... 62
message ... 37
Messages: configuring ... 37
mlc ... 39
Multiple Login Checking ... 74
multiple-login ... 37

N

Naming conventions ... 5
NAS ... 7, 201
NAS types, standard ... 49
nas-address ... 124
NAS-Identifier ... 170
NAS-IP-Address ... 170
nas-port ... 124
NAS-Port-Id ... 170
NAS-Port-Type ... 171
nas.scm ... 163
'naslist' file ... 45
'naslist' file ... 47
'nastypes' file ... 47
'nastypes' file ... 49
'nastypes' file, syntax of ... 47
network ... 32
Network Access Server ... 7, 201
newline ... 123
ngettext ... 114
'NOREALM', special realm name ... 51
ntohl ... 112
ntohs ... 112

O

option ... 22
orig-login ... 124
outfile ... 35

P

PAM Authentication Type ... 73
Pam-Auth ... 189

pam_radius.so 164
password-expire-warning 28, 37
password-expired 37
perms 32
port 28, 30, 32
port-type 124
Prefix 189
prefix-hook 25, 26
print 154
print-auth 26
print-category 26
print-cons 26
print-failed-pass 26
print-level 26
print-milliseconds 26
print-pass 26
print-pid 26
print-priority 26
print-tid 26
Problem Tracking 87
process-idle-timeout 22
Processing requests 12
Propagation of an attribute 7
Properties of an attribute 7
PROPERTY 44
protocol 124
PROXY 212
Proxy Service 13
Proxy-Replied 190
Proxying 13

Q

q 91
qprn 113
query-nas 91
quit 94
quote_string 113

R

r 92
rad-add-server 134
rad-client-add-server 133
rad-client-list-servers 132
rad-client-retry 133
rad-client-set-server 132
rad-client-source-ip 133
rad-client-timeout 133
rad-closelog 119
rad-dict-name->attr 118
rad-dict-name->value 119
rad-dict-pec->vendor 119
rad-dict-value->name 118
rad-format-code 133
rad-format-pair 133
rad-format-reply-msg 134
rad-get-server 132
rad-list-servers 134
rad-log-close 119
rad-log-open 119
rad-openlog 119
rad-print-pairs 133
rad-read-no-echo 133
rad-rewrite-execute 119
rad-rewrite-execute-string 119
rad-select-server 134
rad-send 132
rad-send-internal 132
rad-server-list 131
rad-syslog 119
rad-utmp-putent 119
'radacct', accounting directory 5
radauth 129
radctl 130
'raddb' 5
'raddb/access.deny' file 53
'raddb/client.conf' 135
'raddb/clients' file 45
'raddb/config' file 22
'raddb/hints' file 50
'raddb/huntgroups' file 50
'raddb/menus', configuration subdirectory 62
'raddb/naslist' file 45
'raddb/realms' file 51
'raddb/rewrite', configuration file 62
'raddb/sqlserver' file 53
'raddb/users' file 52
radgrep 128
Radius daemon invocation 17
Radius dictionary 40
Radius-Specific Scheme Functions 118
radiusd 17
Radiusd configuration 22
Radiusd configuration files 21
radiusd-user 22
radlast 126
radlast, options 126
'radlog' 5
radping 129
radscm 131
radsession 162
radtest 136

`radwho` ... 121
`radwho`, command line options ... 121
`radwho`, format strings ... 123
`radwho`, predefined formats ... 125
`radzap` ... 127
`rd` ... 93
`realm` ... 125
`Realm-Name` ... 190
`realm-quota` ... 37
Realms ... 14
'`realms`' file ... 51
`'realms' file` ... 52
Regular Expressions, Rewrite ... 110
Reject Authentication Type ... 71
`Replace-User-Name` ... 190
`REPLY` ... 211
`Reply-Message` ... 171
Request ... 8
request queue, configuring ... 69
`request-cleanup-delay` ... 28, 30, 32
`request-define` ... 93
`request-print` ... 94
`request_code` ... 115
`request_code_string` ... 113
`request_id` ... 115
`request_source_ip` ... 115
`request_source_port` ... 115
Requests, accounting ... 9
Requests, authentication ... 8
`resolve` ... 22
`return` ... 154
`RETURN` ... 210
`rewrite` ... 35
Rewrite ... 98
`Rewrite functions` ... 100
Rewrite identifiers ... 108
Rewrite language settings ... 35
Rewrite, applying functions ... 99
Rewrite, attribute creation functions ... 104
'`rewrite`', configuration file ... 62
Rewrite, data types ... 105
Rewrite, Logging Hook Functions ... 105
Rewrite, login verification functions ... 103
Rewrite, quick start introduction ... 98
Rewrite, symbols ... 106
Rewrite, syntax of the language ... 105
Rewrite, syntax overview ... 98
Rewrite, usage ... 99
`Rewrite-Function` ... 191
`rewrite-stack` ... 92
Rewriting incoming requests ... 99
RHS, Matching Rule ... 11
`rindex` ... 112
`rp` ... 94
`rs` ... 92
Rule Tracing ... 87
`run-rewrite` ... 92
Run-time options (radiusd) ... 22

S

`s` ... 92
`Scheme accounting function` ... 118
`Scheme authentication function` ... 116
`Scheme authentication function, invocation` ... 117
`Scheme-Acct-Procedure` ... 191
`Scheme-Procedure` ... 191
`second-login` ... 37
`send` ... 155
Service ... 201
`Service-Type` ... 171
Session ... 201
Session ID ... 201
`session-id` ... 124
`Session-Timeout` ... 173
`set` ... 154
`shift` ... 154
Simultaneous logins, checking for ... 74
`Simultaneous-Use` ... 192
`snmp` ... 32
SNMP service parameters ... 32
`source` ... 92
`source-ip` ... 22
`sql` ... 39
SQL Accounting ... 83
SQL accounting query templates ... 59
SQL accounting query templates, writing of ... 59
SQL Authentication Type ... 73
'`sqlserver`' file ... 53
`State` ... 173
Statements, Rewrite ... 109
`storage` ... 32
`strip-names` ... 28
`Strip-User-Name` ... 192
`substr` ... 112
`Suffix` ... 193
`suffix-hook` ... 25, 26
Symbols, Rewrite ... 106
Syntax of '`nastypes`' ... 47
`syslog` ... 26
`system` ... 39

System Accounting 81
System Authentication Type 72

T

t 92
tab 123
Termination-Action 173
Termination-Menu 193
Test Mode 90
textdomain 114
time 124
time-to-live 28, 30, 32
timespan 92
timespan-violation 37
tolower 113
toupper 113

U

unquote_string 113
usedbm 32
User Profiles 16
User-Name 174
User-Password 174
username-chars 22
'users' file 52
'users' file 53
Utility Programs 121
utmp-entry 119

V

VALUE 44
VENDOR 41
Vendor-Specific 175

W

wait-reply 38
Writing SQL accounting query templates 59

Short Contents

Introduction to Radius 1
1 Naming Conventions 5
2 How Radius Operates 7
3 How to Start the Daemon 17
4 Radius Configuration Files 21
5 Request Comparison Methods 67
6 Authentication 71
7 Accounting 81
8 Logging 85
9 Problem Tracking 87
10 Extensions 95
11 Utility Programs 121
12 Client Package 135
13 Attribute List 165
14 Reporting Bugs 195
15 Where to Get Information about GNU Radius 197
How to Obtain Radius 199
Radius Glossary 201
Acknowledgements 203
A New Configuration Approach (draft) 205
B GNU Free Documentation License 215
Index 223

Table of Contents

Introduction to Radius **1**
Overview 1

1 Naming Conventions **5**

2 How Radius Operates **7**
2.1 Attributes 7
2.2 RADIUS Requests 8
2.2.1 Authentication Requests 8
2.2.2 Accounting Requests 9
2.3 Matching Rule 11
2.4 Processing Requests 12
2.4.1 Checking for Duplicate Requests 12
2.4.2 Proxying 13
2.4.2.1 Proxy Service 13
2.4.2.2 Realms 14
2.4.3 Hints 14
2.4.4 Huntgroups 15
2.4.5 User Profiles 16

3 How to Start the Daemon. **17**

4 Radius Configuration Files **21**
4.1 Run-Time Configuration Options — '`raddb/config`' 22
4.1.1 `option` block 22
4.1.2 `logging` block 24
4.1.2.1 Logging hooks 25
4.1.2.2 `category` statement 26
4.1.2.3 `channel` statement 26
4.1.2.4 Example of the `logging` statement 27
4.1.3 `auth` statement 28
4.1.4 `acct` statement 30
4.1.5 `usedbm` statement 32
4.1.6 `snmp` statement 32
4.1.7 `rewrite` statement. 35
4.1.8 `guile` statement 35
4.1.9 `message` statement 37
4.1.10 `filters` statement 38
4.1.11 `mlc` statement 39
4.2 Dictionary of Attributes — '`raddb/dictionary`' 40

4.2.1 Comments ... 40
4.2.2 $INCLUDE Statement ... 40
4.2.3 VENDOR Statement ... 41
4.2.4 ATTRIBUTE statement ... 41
4.2.5 Blocks of Vendor-Specific Attributes ... 42
4.2.6 ALIAS statement ... 43
4.2.7 PROPERTY statement ... 44
4.2.8 VALUE Statement ... 44
4.3 Clients List — 'raddb/clients' ... 45
4.3.1 Example of 'clients' file ... 45
4.4 NAS List — 'raddb/naslist' ... 45
4.4.1 Example of 'naslist' file ... 47
4.5 NAS Types — 'raddb/nastypes' ... 47
4.5.1 Syntax of 'raddb/nastypes' ... 47
4.5.2 Example of nastypes file. ... 49
4.5.3 Standard NAS types ... 49
4.6 Request Processing Hints — 'raddb/hints' ... 50
4.6.1 Example of 'hints' file ... 50
4.7 Huntgroups — 'raddb/huntgroups' ... 50
4.7.1 Example of 'huntgroups' file. ... 50
4.8 List of Proxy Realms — 'raddb/realms' ... 51
4.8.1 Example of 'realms' file ... 52
4.9 User Profiles — 'raddb/users' ... 52
4.9.1 Example of 'users' file ... 53
4.10 List of Blocked Users — 'raddb/access.deny' ... 53
4.11 SQL Configuration — 'raddb/sqlserver' ... 53
4.11.1 SQL Client Parameters ... 54
4.11.2 Authentication Server Parameters ... 54
4.11.3 Authorization Parameters ... 56
4.11.4 Accounting Parameters ... 58
4.11.4.1 Writing SQL Accounting Query Templates ... 59
4.12 Rewrite functions — 'raddb/rewrite' ... 62
4.13 Login Menus — 'raddb/menus' ... 62
4.13.1 A menu file syntax. ... 62
4.13.2 An example of menu files ... 62
4.14 Macro Substitution ... 64

5 Request Comparison Methods ... 67

5.1 Extended Comparison ... 67
5.1.1 An example of extended comparison configuration ... 68
5.1.2 List of attributes that can be declared comparable. ... 68
5.2 Fine-Tuning the Request Queue ... 69

6 Authentication ... 71

6.1 Accept Authentication Type ... 71
6.2 Reject Authentication Type ... 71
6.3 Local Password Authentication Type ... 71
6.4 Encrypted Password Authentication Type ... 72
6.5 System Authentication Type ... 72
6.6 SQL Authentication Type ... 73
6.7 PAM Authentication Type ... 73
6.8 Defining Custom Authentication Types ... 73
6.9 Multiple Login Checking ... 74
6.9.1 Retrieving Session Data ... 74
6.9.2 Verifying Active Sessions ... 75
6.10 Controlling Authentication Probes ... 76

7 Accounting ... 81

7.1 System Accounting ... 81
7.2 Detailed Request Accounting ... 81
7.3 sql Accounting ... 83
7.4 Defining Custom Accounting Types ... 83

8 Logging ... 85

9 Problem Tracking ... 87

9.1 Rule Tracing ... 87
9.2 Debugging ... 89
9.3 Test Mode ... 90

10 Extensions ... 95

10.1 Filters ... 95
10.1.1 Getting Acquainted with Filters ... 95
10.1.2 Declaring the Filter ... 95
10.1.3 Invoking the Filter from a User Profile ... 96
10.1.4 Adding Reply Attributes ... 96
10.1.5 Accounting Filters ... 97
10.1.6 Invoking the Accounting Filter ... 98
10.2 Rewrite ... 98
10.2.1 Syntax Overview ... 98
10.2.2 Quick Start ... 98
10.2.3 Interaction with Radius ... 99
10.2.4 Rewriting Incoming Requests ... 99
10.2.4.1 Examples of Various Rewrite Functions ... 100
10.2.5 Login Verification Functions ... 103
10.2.5.1 Examples of Login Verification Functions ... 104
10.2.6 Attribute Creation Functions ... 104

10.2.7 Logging Hook Functions ... 105
10.2.8 Full Syntax Description ... 105
10.2.8.1 Rewrite Data Types ... 105
10.2.8.2 Rewrite Symbols ... 106
10.2.8.3 Rewrite Identifiers ... 108
10.2.8.4 Rewrite Declarations ... 108
10.2.8.5 Rewrite Statements ... 109
10.2.8.6 Regular Expressions ... 110
10.2.8.7 Rewrite Built-in Functions ... 111
10.3 Guile ... 115
10.3.1 Data Representation ... 115
10.3.2 Authentication with Scheme ... 116
10.3.3 Accounting with Scheme ... 117
10.3.4 Radius-Specific Functions ... 118

11 Utility Programs ... 121
11.1 `radwho` ... 121
11.1.1 `radwho` Command Line Options ... 121
11.1.2 `radwho` Format Strings ... 123
11.1.3 `radwho` Predefined Formats ... 125
11.2 `radlast` ... 126
11.2.1 `radlast` Command Line Options ... 126
11.3 `radzap` ... 127
11.4 `radgrep` ... 128
11.5 `radping` ... 129
11.6 `radauth` ... 129
11.7 `radctl` ... 130
11.8 `builddbm` ... 130
11.9 `radscm`: A Guile Interface to Radius Functions ... 131

12 Client Package ... 135
12.1 Client Configuration ... 135
12.2 radtest ... 136
12.2.1 Invoking radtest ... 136
12.2.2 Literal Values ... 137
12.2.2.1 Numeric Values ... 137
12.2.2.2 Character Strings ... 137
12.2.2.3 Lists of A/V pairs ... 139
12.2.3 Reserved Keywords ... 139
12.2.4 Variables ... 139
12.2.4.1 Using Variables ... 139
12.2.4.2 Variable Assignments ... 140
12.2.4.3 Dereferencing Variables ... 140
12.2.4.4 Accessing Elements of A/V Pair Lists ... 142
12.2.4.5 Assignment Options ... 142

12.2.4.6 Built-in Variables 143
12.2.5 Positional Parameters 143
12.2.6 Expressions 144
12.2.6.1 Arithmetic Operations 144
12.2.6.2 String Operations 145
12.2.6.3 Operations on A/V Lists 145
12.2.6.4 Comparison Operations 146
12.2.6.5 Boolean Operations 147
12.2.6.6 Conversion Between Data Types 147
12.2.6.7 Function Calls 148
12.2.6.8 Operator Precedence (How Operators Nest) 148
12.2.7 Function Definitions 149
12.2.8 Interacting with Radius Servers 149
12.2.9 Conditional Statements 150
12.2.10 Loops 152
12.2.11 Built-in Primitives 153
12.2.12 Sample Radtest Program 155
12.3 radsession 162
12.4 `nas.scm` 163
12.5 `pam_radius.so` 164

13 Attribute List 165
13.1 Authentication Attributes 165
13.1.1 `CHAP-Password` 166
13.1.2 `Callback-Id` 166
13.1.3 `Callback-Number` 166
13.1.4 `Called-Station-Id` 166
13.1.5 `Calling-Station-Id` 167
13.1.6 `Class` 167
13.1.7 `Framed-Compression` 167
13.1.8 `Framed-IP-Address` 168
13.1.9 `Framed-IP-Netmask` 168
13.1.10 `Framed-MTU` 168
13.1.11 `Framed-Protocol` 169
13.1.12 `Framed-Route` 169
13.1.13 `Framed-Routing` 169
13.1.14 `Idle-Timeout` 169
13.1.15 `NAS-IP-Address` 170
13.1.16 `NAS-Identifier` 170
13.1.17 `NAS-Port-Id` 170
13.1.18 `NAS-Port-Type` 171
13.1.19 `Reply-Message` 171
13.1.20 `Service-Type` 171
13.1.21 `Session-Timeout` 173
13.1.22 `State` 173
13.1.23 `Termination-Action` 173

13.1.24 User-Name ... 174
13.1.25 User-Password ... 174
13.1.26 Vendor-Specific ... 175
13.2 Accounting Attributes ... 175
13.2.1 Acct-Authentic ... 175
13.2.2 Acct-Delay-Time ... 175
13.2.3 Acct-Input-Octets ... 176
13.2.4 Acct-Input-Packets ... 176
13.2.5 Acct-Output-Octets ... 176
13.2.6 Acct-Output-Packets ... 176
13.2.7 Acct-Session-Id ... 177
13.2.8 Acct-Session-Time ... 177
13.2.9 Acct-Status-Type ... 177
13.2.10 Acct-Terminate-Cause ... 178
13.3 Radius Internal Attributes ... 178
13.3.1 Acct-Ext-Program ... 178
13.3.2 Acct-Type ... 179
13.3.3 Auth-Failure-Trigger ... 179
13.3.4 Auth-Data ... 180
13.3.5 Auth-Type ... 180
13.3.6 Crypt-Password ... 181
13.3.7 Exec-Program-Wait ... 181
13.3.7.1 Running an External Program ... 182
13.3.7.2 Using an External Filter ... 182
13.3.8 Exec-Program ... 183
13.3.9 Fall-Through ... 184
13.3.10 Group ... 185
13.3.11 Hint ... 185
13.3.12 Huntgroup-Name ... 186
13.3.13 Log-Mode-Mask ... 186
13.3.14 Login-Time ... 187
13.3.15 Match-Profile ... 188
13.3.16 Menu ... 189
13.3.17 Pam-Auth ... 189
13.3.18 Prefix ... 189
13.3.19 Proxy-Replied ... 190
13.3.20 Realm-Name ... 190
13.3.21 Replace-User-Name ... 190
13.3.22 Rewrite-Function ... 191
13.3.23 Scheme-Acct-Procedure ... 191
13.3.24 Scheme-Procedure ... 191
13.3.25 Simultaneous-Use ... 192
13.3.26 Strip-User-Name ... 192
13.3.27 Suffix ... 193
13.3.28 Termination-Menu ... 193

14 Reporting Bugs 195

15 Where to Get Information about GNU Radius 197

How to Obtain Radius 199

Radius Glossary 201

Acknowledgements 203

Appendix A New Configuration Approach (draft) 205

A.1 A brief description of Currently Used Approach 205
A.2 Deficiencies of Current Operation Model and Configuration Suite 206
A.3 Proposed Solution 207
A.3.1 Request-processing Instruction 207
A.3.2 grad_instr_conditional 208
A.3.3 grad_instr_call 209
A.3.4 grad_instr_return 210
A.3.5 grad_instr_action 210
A.3.6 grad_instr_reply 211
A.3.7 grad_instr_proxy 211
A.3.8 grad_instr_forward 212
A.3.9 RPL representation 212
A.4 Changes to Rewrite Language 212
A.5 Support for Traditional Configuration Files 212
A.6 New Configuration Files 214

Appendix B GNU Free Documentation License 215

B.1 ADDENDUM: How to use this License for your documents 222

Index 223