## Preface

- Audience
- Downloading Oracle Communications Documentation
- Documentation Accessibility

## 1 Understanding RADIUS Manager

- About the RADIUS Protocol
- What You Can Do with RADIUS Manager
- How RADIUS Manager Works
- RADIUS Manager Features and Functionality
- How RADIUS Manager Performs Authentication and Authorization
- How RADIUS Manager Performs Accounting
- Customizing RADIUS Manager Opcodes

## 2 Installing RADIUS Manager

- System Requirements
- Software Requirements
- Installing RADIUS Manager
- What’s Next?
- Uninstalling RADIUS Manager

## 3 Configuring RADIUS Manager

- Planning Checklist for RADIUS Implementations
- Overview of RADIUS Manager Configuration Tasks
- Connecting RADIUS Manager to BRM
- Connecting the RADIUS Manager Client to BRM
- Configuring RADIUS Implementations
  - Setting the IP Port Numbers
  - Setting the Log Level for Application Errors
  - Enabling the Debug Option
  - Selecting the Data Dictionary
  - Selecting a Data Dictionary When Using Different NAS Vendors
  - Defining the Process ID of the RADIUS Server
  - Configuring the stop_radius Script
  - Setting the Number of Threads and the Queue Size for Scalability
7 Logging RADIUS Requests

Logging Requests to Text Files ................................................................. 7-1
Using mod_logging ........................................................................ 7-1
Configuring mod_logging ................................................................. 7-2
Checking Requests ........................................................................ 7-2
Specifying the Destination File ........................................................ 7-2
Examples Using mod_logging .......................................................... 7-2
Creating Custom CDR Log Files .......................................................... 7-3
Scenarios for Creating Custom CDR Log Files ...................................... 7-3
Checklist of Customization Considerations for mod_cdr ...................... 7-4
Configuring mod_cdr ....................................................................... 7-5
Logging Different Types of RADIUS Messages ..................................... 7-5
Specifying the Attribute Order ........................................................ 7-5
Defining Field and Record Delimiters ............................................... 7-5
Specifying Delimiters for String Literals ............................................. 7-5
Defining Blank Fields ........................................................................ 7-6
Including Special Characters .......................................................... 7-6
Using cleartext or Encoded Password Entries ...................................... 7-6
Pre-appending the RADIUS Request Type .......................................... 7-6
Setting Limits on the Number of Records and Log Files ...................... 7-6
Configuring the Destination for Log Files .......................................... 7-7
Default Configuration for the Destination File ..................................... 7-7
Basic Customization of the Dest Field ............................................... 7-7
Adding a Numeric Counter to the dest Field ...................................... 7-7
Adding the NAS Option to the dest Field ........................................... 7-7
Adding Time-Based Rotation Schemes to the dest Field ................. 7-7
Guidelines for Setting Up Rotation Schemes .................................... 7-8
Managing Current Files ..................................................................... 7-9
Managing Archive and Backup Files .................................................. 7-9

8 Transforming Attribute Values

How the Transformation Module Works .............................................. 8-1
Configuring the Transformation Module ............................................. 8-1
Matching Attribute-Value Pairs in a Request ...................................... 8-1
Transforming Attributes ..................................................................... 8-1
Appending Values to Attributes ....................................................... 8-4
Copying Attribute Values .................................................................. 8-4
Example Using the Transformation Module ........................................ 8-4

9 Managing Requests for Wholesale Accounts

About the Wholesale Module ............................................................... 9-1
Specifying the Realm Attributes and Options .................................... 9-1
## 13 Setting Up iPass Roaming

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About iPass Roaming</td>
<td>13-1</td>
</tr>
<tr>
<td>iPass Checklist</td>
<td>13-1</td>
</tr>
<tr>
<td>Identifying and Forwarding Requests</td>
<td>13-1</td>
</tr>
<tr>
<td>Configuring mod_proxy</td>
<td>13-2</td>
</tr>
<tr>
<td>Recording Locally Serviced iPass Sessions</td>
<td>13-3</td>
</tr>
<tr>
<td>Handling Remote Authentication Requests</td>
<td>13-3</td>
</tr>
<tr>
<td>Handling Online and Batch Requests</td>
<td>13-4</td>
</tr>
<tr>
<td>Online Accounting</td>
<td>13-4</td>
</tr>
<tr>
<td>Batch Accounting</td>
<td>13-4</td>
</tr>
<tr>
<td>Logging Accounting Packets for Batch Accounting</td>
<td>13-4</td>
</tr>
<tr>
<td>Logging Accounting Packets for Online Accounting</td>
<td>13-5</td>
</tr>
<tr>
<td>iPass Call Detail Record File Definition</td>
<td>13-5</td>
</tr>
<tr>
<td>Example RADIUS Configuration File for iPass</td>
<td>13-6</td>
</tr>
<tr>
<td>Using the pin_ipass_loader Utility for Batch Accounting</td>
<td>13-8</td>
</tr>
<tr>
<td>Running pin_ipass_loader</td>
<td>13-8</td>
</tr>
<tr>
<td>Error Handling</td>
<td>13-8</td>
</tr>
</tbody>
</table>

## 14 Testing a RADIUS Configuration

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the rad_tester Utility</td>
<td>14-1</td>
</tr>
<tr>
<td>Configuring rad_tester</td>
<td>14-1</td>
</tr>
<tr>
<td>Configuring the Accounting and Authentication Ports</td>
<td>14-1</td>
</tr>
<tr>
<td>Defining the Client in the RADIUS Configuration File</td>
<td>14-2</td>
</tr>
<tr>
<td>Creating Input Packets</td>
<td>14-2</td>
</tr>
<tr>
<td>Simulating Multiple Users</td>
<td>14-2</td>
</tr>
<tr>
<td>Running rad_tester</td>
<td>14-2</td>
</tr>
<tr>
<td>Sample Authentication Request</td>
<td>14-3</td>
</tr>
<tr>
<td>Sample Start Accounting Request</td>
<td>14-3</td>
</tr>
<tr>
<td>Sample Stop Accounting Request</td>
<td>14-3</td>
</tr>
<tr>
<td>Sample Interim Accounting Request</td>
<td>14-3</td>
</tr>
<tr>
<td>Sending Input Packets to the RADIUS Server</td>
<td>14-4</td>
</tr>
<tr>
<td>About the Account Request Utility</td>
<td>14-4</td>
</tr>
<tr>
<td>Sending an Accounting-Off Request</td>
<td>14-4</td>
</tr>
<tr>
<td>Listing All Open Sessions</td>
<td>14-4</td>
</tr>
</tbody>
</table>

## 15 RADIUS Manager Utilities

<table>
<thead>
<tr>
<th>Utility</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin_ipass_loader</td>
<td>15-2</td>
</tr>
<tr>
<td>pin_radiusd_sig</td>
<td>15-4</td>
</tr>
<tr>
<td>pin_term_acct</td>
<td>15-5</td>
</tr>
<tr>
<td>rad_tester</td>
<td>15-7</td>
</tr>
</tbody>
</table>
This book describes how to configure Oracle Communications Billing and Revenue Manager (BRM) RADIUS Manager.

**Audience**

This document is intended for developers and system administrators.

**Downloading Oracle Communications Documentation**

Product documentation is located on Oracle Technology Network:

http://docs.oracle.com

Additional Oracle Communications documentation is available from Oracle E-Delivery:

http://edelivery.oracle.com

**Documentation Accessibility**

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at


**Access to Oracle Support**

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.
Understanding RADIUS Manager

This chapter provides an overview of Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager. Anyone who installs, configures, administers, or customizes RADIUS Manager should read this chapter.

**Important:** RADIUS Manager is an optional component, not part of base BRM.

Before using RADIUS Manager to set up a RADIUS configuration, you should be familiar with the RFC 2865: Remote Authentication Dial In User Service (RADIUS) and RFC 2866: RADIUS Accounting.

**Important:** RADIUS Manager is not fully compliant with RFC 2865 and 2866. Only some of the RFC 2865 and 2866 features are supported. However, to meet specific business requirements, you can modify RADIUS Manager policy opcodes and write custom RADIUS modules to add functionality. See "Customizing RADIUS Manager Opcodes" and "Creating Custom Modules".

### About the RADIUS Protocol

The RADIUS protocol is an industry standard protocol for authentication, authorization, and accounting (AAA). Terminal servers or Network Access Server (NAS) use the RADIUS protocol to communicate AAA requests to, and return results from, a database of customer information.

RADIUS Manager uses the RADIUS protocol to provide AAA services in the BRM environment.

### What You Can Do with RADIUS Manager

You use RADIUS Manager to perform the authentication, authorization and accounting services required when customers use your terminal server or Network Access Server (NAS) to connect to BRM.

RADIUS Manager performs these tasks:

- **Authentication:** Verifying a customers identity by checking the user name and password
- **Authorization:** Verifying a customers privileges for accessing the requested services
How RADIUS Manager Works

An organization connects a terminal server or Network Access Server (NAS), and the terminal server communicates with RADIUS Manager, as shown in Figure 1–1. Terminal servers authenticate incoming connections, authorize customers, then (if enabled) start accounting. To perform these operations, the NAS uses the RADIUS protocol to communicate with the pin_radiusd daemon. The pin_radiusd daemon accesses your BRM database. See “pin_radiusd_sig”.

Figure 1–1  RADIUS Manager Connectivity

The RADIUS protocol can authorize access requests to:

- BRM databases
- Text-based user files
- UNIX password files
- iPass databases
- Proxy databases
- VPDN databases

RADIUS Manager Features and Functionality

You can perform these tasks with RADIUS Manager:

- **Custom RADIUS settings**: You can customize your configuration of RADIUS Manager by setting the IP listen ports, debug options, data dictionary, vendor specific attributes (VSAs), number of threads, queue size, client list, secret, default return attributes, and configuration macros. For information on using these settings, see "Configuring RADIUS Manager".

- **Define the RADIUS workflow using general purpose modules**: RADIUS Manager provides a variety of standard modules to handle most RADIUS requests. Use these modules to add or remove functionality without rebuilding
How RADIUS Manager Performs Accounting

RADIUS Manager uses the following opcodes for accounting:

- **PCM_OP_TERM_IP_DIALUP_AUTHORIZE** assembles information from the NAS and a user’s account, and authorizes the user. It calls the PCM_OP_TERM_POL_AUTHORIZE policy opcode. You can customize the PCM_OP_TERM_POL_AUTHORIZE to calculate maximum authorized charges for prepaid rating.

How RADIUS Manager Performs Authentication and Authorization

RADIUS Manager uses the following opcodes for authenticating and authorizing customers:

- **PCM_OP_TERM_IP_DIALUP_AUTHORIZE** authenticates a user.

  This opcode verifies a user’s login and password. If a password is passed in to this opcode, it calls PCM_OP_ACT_FIND_VERIFY and PCM_OP_ACT_POL_SPEC_VERIFY to authenticate the password. If no password is passed in, this opcode assumes that CHAP authorization is used and it returns the unencrypted password for verification by the calling program.

- **PCM_OP_TERM_IP_DIALUP_AUTHENTICATE** assembles information from the NAS and a user’s account, and authorizes the user. It calls the PCM_OP_TERM_POL_AUTHORIZE policy opcode. You can customize the PCM_OP_TERM_POL_AUTHORIZE to calculate maximum authorized charges for prepaid rating.

How RADIUS Manager Performs Accounting

RADIUS Manager uses the following opcodes for accounting:
• PCM_OP_TERM_IP_DIALUP_ACCOUNTING_ON enables RADIUS Manager to tell BRM that it is ready for service. RADIUS Manager can be configured to mark the start of accounting by using the RADIUS protocol to handle Accounting-On requests. When BRM receives an Accounting-On request, it closes any open account associated with that RADIUS Manager opened before RADIUS Manager indicated it was active.

• PCM_OP_TERM_IP_DIALUP_ACCOUNTING_OFF records the end of accounting. The Terminal Server can be configured to mark the end of accounting by the use of the RADIUS protocol to handle Accounting-Off requests. This feature allows the Terminal Server to communicate with BRM that it is going out of service. When BRM receives an Accounting-Off request it closes all accounts associated with that Terminal Server so that accounts are not left open if the Terminal Server is going out of service.

• PCM_OP_TERM_IP_DIALUP_START_ACCOUNTING records the start of a previously authenticated IP dialup session.

This opcode calls PCM_OP_ACT_FIND to find the account. It then searches for current sessions and does one of the following actions:

– If a session exists and is a duplicate, no new session is created.
– If a session exists but is not a duplicate, the existing session is closed as an orphan session.
– If no session exists, calls PCM_OP_ACT_START_SESSION to start a new session.

• PCM_OP_TERM_IP_DIALUP_UPDATE_ACCOUNTING updates a previously started IP dialup session.

• PCM_OP_TERM_IP_DIALUP_STOP_ACCOUNTING closes out a previously started IP dialup session.

This opcode calls PCM_OP_ACT_FIND to find the account. It then searches for current sessions and does one of the following actions:

– If more than one session is found, returns an error.
– If a session is open, or if a session was orphaned, closes the session.
– If no session is found, creates and closes a session.

Customizing RADIUS Manager Opcodes

Use the following opcodes to customize RADIUS Manager:

• Use the PCM_OP_TERM_POL_ACCOUNTING policy opcode to set the event type (such as /event/session/dialup/ascend) and extract extra fields from PIN_FLD_ARGS and PIN_FLD_INHERITED_INFO. You can then add these fields to an extended /event/session/dialup event.

The default implementation does nothing.

PCM_OP_TERM_IP_DIALUP_START_ACCOUNTING, PCM_OP_TERM_IP_DIALUP_STOP_ACCOUNTING, and PCM_OP_TERM_IP_DIALUP_UPDATE_ACCOUNTING call this opcode.

• Use the PCM_OP_TERM_POL_AUTHORIZE policy opcode to merge attributes from the NAS and a user’s account into a list to be returned to the NAS.
Use the PCM_OP_TERM_POL_REVERSE_IP policy opcode to find the user of a given IP address.

This opcode maps the IP address to the relevant account and service objects. It finds open event session for given IP address and returns user information based on that session.
This chapter explains how to install the Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager software.

---

**Important:** RADIUS Manager is an optional feature that requires a separate license.

---

Before you read this chapter, you should be familiar with BRM concepts and architecture. See "Introducing BRM" and "BRM System Architecture" in BRM Concepts and "Understanding RADIUS Manager Modules".

### System Requirements

RADIUS Manager is available for the HP-UX IA64, Solaris, AIX, and Linux for terminal servers using the RADIUS protocol. For information on disk space requirements for the HP-UX IA64, Solaris, AIX, and Linux operating systems, see "Disk Space Requirements" in BRM Installation Guide.

For a typical production system, install RADIUS Manager on the same machine as a CM.

### Software Requirements

Before installing RADIUS Manager, you must install:

- Third-Party software, which includes the PERL libraries and JRE required for installing BRM components. See "Installing the Third-Party Software" in BRM Installation Guide.
- BRM. See "Putting Together Your BRM System" in BRM Installation Guide.

### Installing RADIUS Manager

---

**Note:** If you have already installed the product, features that are already installed cannot be reinstalled without uninstalling them first. To reinstall a feature, uninstall it and then install it again.

---

To install RADIUS Manager:

1. Download the software to a temporary directory (`temp_dir`).
Installing RADIUS Manager

2. Go to the directory where you installed the Third-Party package and source the `source.me` file.

```bash
source source.me.sh
```

```csh
source source.me.csh
```

3. Go to the `temp_dir` directory and enter this command:

```bash
7.4_RadiusMgr_platform_32_opt.bin
```

4. Follow the instructions displayed during installation. The default installation directory for RADIUS Manager is `/opt/portal/7.4`.

```bash
Note: You can use the `-console` parameter to run the installation in command-line mode. To enable a graphical user interface (GUI) installation, install a GUI application such as X Windows and set the DISPLAY environment variable before you install the software.
```

5. Go to the directory where you installed the RADIUS Manager package and source the `source.me` file:

```bash
source source.me.sh
```

```csh
source source.me.csh
```

6. Go to the `BRM_home/setup` directory and run the `pin_setup` script.

---

**Note:**

- If you download to a Windows workstation, use FTP to copy the `.bin` file to a temporary directory on your UNIX server.
- You must increase the heap size used by the Java Virtual Machine (JVM) before running the installation program to avoid "Out of Memory" error messages in the log file. For information, see "Increasing Heap Size to Avoid 'Out of Memory' Error Messages" in BRM Installation Guide.

**Caution:** You must source the `source.me` file to proceed with installation, otherwise "suitable JVM not found" and other error messages appear.

**Note:** You can use the `-console` parameter to run the installation in command-line mode. To enable a graphical user interface (GUI) installation, install a GUI application such as X Windows and set the DISPLAY environment variable before you install the software.

**Note:** The installation program does not prompt you for the installation directory if BRM or RADIUS Manager is already installed on the machine and automatically installs the package at the `BRM_home` location.
7. (Oracle only) If you use Oracle for your BRM database and your event tables are partitioned, run the `partition_utils` utility with the `-o update` parameter from the `BRM_home/apps/partition_utils` directory:

```
perl partition_utils.pl -o update
```

For more information, see "Updating Partitions" and "partition_utils" in *BRM System Administrator’s Guide*.

Your RADIUS Manager installation is now complete.

**What’s Next?**

See "Configuring RADIUS Manager".

**Uninstalling RADIUS Manager**

To uninstall RADIUS Manager, run the `BRM_home/uninstaller/RadiusMgr/uninstaller.bin`. 

---

**Note:** The `pin_setup` script starts all required BRM processes.
This chapter describes how to configure Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager. Anyone who installs, configures, administers, or customizes RADIUS Manager should read this chapter.

**Caution:** Only administrators with advanced RADIUS skills should attempt to implement custom configurations.

Before configuring RADIUS Manager, you must install BRM:

- "Putting Together Your BRM System" in *BRM Installation Guide*
- "Installing BRM" in *BRM Installation Guide*
- Installing RADIUS Manager

**Planning Checklist for RADIUS Implementations**

Before you begin the configuration process, plan the features of your custom RADIUS implementation. This planning is critical to a successful RADIUS implementation. While each configuration is unique, there are basic questions you must answer for each configuration. Use the following checklist to ensure that these issues are considered in your planning process.

**Note:** This chapter assumes that you are already familiar with the RADIUS protocol. If any terms in this checklist are unfamiliar, see RFC 2865: Remote Authentication Dial In User Service (RADIUS) and RFC 2866: RADIUS Accounting.

**Define the RADIUS Architecture**

- Which data dictionary do I use? See "Defining the Data Dictionary".
- Do I need to merge data dictionaries from multiple Network Access Server (NAS) vendors? See "Editing the Data Dictionary".
- How many threads and what queue size are appropriate for my configuration? See "Setting the Number of Threads and the Queue Size for Scalability".
- What time zone setting do I use? See "Specifying the Time Zone".
- Do I need to set any vendor-specific attributes for my NAS? See "Adding Vendor Specific Attributes to the Data Dictionary".
Overview of RADIUS Manager Configuration Tasks

Authentication and Authorization Methods
- Do I need one authentication method for internal users? See "About Local and Remote Authentication Requests".
- Do I need multiple authentication methods for remote users? See "About Cascading Authentication".
- Do I set up proxying to forward requests from multiple NASs? See "Using the Proxy Module to Forward Incoming Requests to an External RADIUS Server".
- Are all of my NASs listed in the client list? See "Defining the RADIUS Client List".
- Have I configured the listen ports correctly? See "Setting the IP Port Numbers".

Accounting
- Should I use stop-only accounting to improve performance? See "Specifying How Accounting-On Events Are Handled after a NAS Failure".
- Do I turn on debug options? See "Enabling the Debug Option".
- At what level do I want to log error messages? See "Setting the Log Level for Application Errors".
- Do I create field-delimited log files that can be read by Universal Event (UE) Loader or other applications? See "Creating Custom CDR Log Files".

Overview of RADIUS Manager Configuration Tasks
To configure RADIUS Manager, complete these configuration tasks:
- Set the link between RADIUS Manager and BRM in the CM pin.conf file. See "Connecting RADIUS Manager to BRM".
- Set the connection between the RADIUS server and BRM in the RADIUS pin.conf file. See "Connecting the RADIUS Manager Client to BRM".
- Configure the RADIUS server by editing the RADIUS pin.conf file. See "Configuring RADIUS Implementations".
- Start the RADIUS daemon. See "Starting and Stopping the RADIUS Daemon".

Connecting RADIUS Manager to BRM
Use the CM (Connection Manager) configuration (pin.conf) file to link RADIUS Manager to BRM. This operation is automatic if RADIUS Manager is installed on the computer where the CMs reside. If you are installing RADIUS Manager on a computer that does not have BRM installed, you must make the following changes to the CM configuration file (BRM_home/sys/cm/pin.conf), where BRM_home is the directory in which BRM components are installed.

Solaris, Linux, and HP-UX IA64
```
cm fm_module ../../lib/fm_term.so fm_term_config - pin
cm fm_module ../../lib/fm_term_pol.so fm_term_pol_config - pin
```

AIX
```
cm fm_module ../../lib/fm_term.a fm_term_config - pin
cm fm_module ../../lib/fm_term_pol.a fm_term_pol_config - pin
```
Connecting the RADIUS Manager Client to BRM

Use the RADIUS server configuration file (BRM_home/apps/radius/pin.conf) to connect the RADIUS server to BRM. This file contains the standard BRM connection entries and editing instructions.

Configuring RADIUS Implementations

Use the RADIUS configuration file (BRM_home/apps/radius/config) to define the features of your custom RADIUS implementation. This file has three sections:

- **$CONFIG** - Includes the core server settings ($CORE) as well as global module settings.
- **$DEFINE** - Allows you to define macros that are referenced elsewhere in the configuration file.
- **$MODULES** - Includes settings for each module instance.

**Important:** Before you begin customizing RADIUS settings, make a copy of BRM_home/apps/radius/config.tsm, and rename the file. The file name required by the RADIUS server is config.

This following sections explain how to configure settings for the $CORE and $DEFINE sections of the RADIUS configuration file. The $MODULES descriptions are in the configuration documentation for each type of module.

Setting the IP Port Numbers

Set the IP port numbers to which the RADIUS server listens by defining the listen entry in the $CORE section of the RADIUS configuration file (BRM_home/apps/radius/config). Add one listen entry for each port to which you want the server to listen. You do not need to distinguish between authentication and accounting ports. The RADIUS server can listen to more than two ports. The default authorization port is 1812 and the default accounting port is 1813.

For example:

```plaintext
listen {
    port = 1812
}
listen {
    port = 1813
}
```

Setting the Log Level for Application Errors

Use the log entry in the $CORE section of the RADIUS configuration file (BRM_home/apps/radius/config) to set the file name, path, and log level for logging application-level errors, warnings, and informational messages.

The default log file path is /var/portal7.4/radius.pinlog and the default log level is 2.

Log level 1 logs error messages only. Log level 2 logs error and warning messages. Log level 3 logs error, warning, and informational messages.

For example:

```plaintext
log {
```
Enabling the Debug Option

Use the `debug` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to enable the debug option and log output to `stdout`. Only `all` is supported. Enabling this option allows you to see which `flists` the CM and the `pin_radiusd` utility are using to communicate.

If you use the `start_radius` script, messages are logged to the debug log file `/var/portal/7.4/radius/radius.log`.

For example:

```plaintext
debug {
    #all
}
```

Selecting the Data Dictionary

Use the following procedure to select a data dictionary:

1. Choose one of the supplied dictionaries and copy it to the `BRM_home/apps/radius` directory.

Two dictionaries ship with RADIUS Manager:

- `dictionary-RFC2865` is an RFC-compliant dictionary.
- `dictionary-ascend-4.5` works with Ascend 4.5.

2. Set the file name and path of the RADIUS dictionary file with the `dictionary` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`):

   ```plaintext
dictionary=/opt/portal/7.4/apps/radius/dictionary_name
```

   where `dictionary_name` is the dictionary file you copied in step 1.

   **Important:** Ensure that you specify the same dictionary file for the NAS and RADIUS Manager.

3. (Optional) Modify these dictionary files. See "Editing the Data Dictionary".
Selecting a Data Dictionary When Using Different NAS Vendors
If you must use NAS from different vendors, you have these options:

- If all of the NAS are RFC 2865 compliant then you can use dictionary RFC2865. This is the preferred solution. Don’t forget to update the dictionary file with any vendor-specific attributes associated with the NAS.
- If your NAS is not RFC 2865 compliant, you can merge data dictionary files. For instructions, see "Editing the Data Dictionary".

Defining the Process ID of the RADIUS Server
Use the `pidfile` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to set the file name and path of the file containing the process ID of the currently running RADIUS server. The default name and location is `/var/portal/7.4/radius/radiusd.pid`

For example:
```
pidfile=/var/portal/7.4/radius/pin_radiusd.pid
```

Configuring the stop_radius Script
You use the `stop_radius` script to stop the RADIUS daemon. To use the script, you must change three parameters.

1. Open the `stop_radius` script (`BRM_home/bin/stop_radius`) with a plain text editor.
2. Change the default values for the following parameters:

   - `RADIUS_HOST=` `host_name`
   - `RADIUS_PORT=` `port_number`
   - `RADIUS_SECRET=` `secret`

   See "pin_radiusd_sig".
3. Close the file.

   **Note:** For information on how to use this script, see "Starting and Stopping the RADIUS Daemon".

Setting the Number of Threads and the Queue Size for Scalability
The number of selected threads is a major scalability factor. Usually, increasing the number of threads increases performance. Setting the number of threads too low decreases throughput. Setting the number too high wastes memory and causes unnecessary context switching to manage the extraneous processes.

There is no one criterion for setting the number of threads. Many factors impact the number of threads required, such as the cache size of each CPU, memory size, and swap size. Systems can handle as many as eight threads per CPU. On production systems, set these values higher.

Setting the Number of Threads
Use the `n_threads` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to set the number of threads used by the RADIUS server to service incoming requests. Set the number of threads to process requests.

For example:
Configuring RADIUS Implementations

n_threads=4

---

**Note:** Set the value of the `dm_trans_be_max` entry in your BRM database Data Manager `pin.conf` file to match the value of the `n_threads` entry. The number must match the `num_connects` entry in the RADIUS `pin.conf` file.

---

See "Determining the Required Number of Back Ends" in *BRM System Administrator’s Guide*.

**Adjusting the Queue Size**
Adjust the queue (`req_queue`) size to match the number of threads (`n_threads`). `req_queue` sets the size of the queue used by the RADIUS server to service incoming requests. For high-volume usage, set `req_queue` to 200 times the value of `n_threads`.

For example:

```
req_queue=40
```

**Defining the RADIUS Client List**
Use the `client` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to define the list of RADIUS clients that are permitted to connect to the server. You can have multiple `client` entries. Make sure the NAS is listed in the client list.

For example:

```
client {
    addr = nas1
    addr = nas2
}
```

**Defining the Secret**
The `secret` entry is a common password shared between the RADIUS server and the NAS. It is used by the RADIUS protocol for security. Use the `secret` entry in the RADIUS configuration file (`BRM_home/apps/radius/config`) to set the password shared between the client NAS and RADIUS Manager.

---

**Important:** The `secret` entry must be included with each NAS listing in the client section of the RADIUS configuration file. This entry must match on the NAS and RADIUS Manager.

---

For example:

```
client {
    addr = nas1
    secret = testing123
}
```

**Setting Limits on the Number of Processes**
Use the `fdlimit` entry in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to set the process limit on the number of files handles. See UNIX man page `setrlimit(2)` for more information.
For example:

```
fdlimit=64
```

### Setting the Return Attributes

You must define which attributes are returned from the RADIUS server to the NAS in the authentication packet. These return attributes are set in the `$DEFINE` section in the RADIUS configuration file (`BRM_home/apps/radius/config`). To add or modify entries in the reply packet, edit the `send` section.

For example:

```
...
$DEFINE {
   send {
       Service-Type=Framed
       Framed-IP-Netmask=255.255.255.0
       Framed-IP-Address=255.255.255.254
   }
}
...
```

### Defining Configuration Macros

To avoid repeating common elements, use the `$DEFINE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to define entries that serve as configuration macros for the RADIUS configuration file. This portion of the `$DEFINE` section shows the preset elements available with RADIUS Manager:

```
$DEFINE {
   basic-ip-settings {
       send {
           Service-Type=Framed
           Framed-IP-Netmask=255.255.255.0
           Framed-IP-Address=255.255.255.254
       }
   }
}
```

### Including Configuration Macros

After configuration macros are set in the `$DEFINE` module, use the `$INCLUDE` entry to add them, as needed, to the RADIUS configuration file.

For example:

```
$INCLUDE = basic-ip-settings
```

### Retrieving Performance Statistics

You can gather internal statistics on the performance of queues and connections. This data is exported at specified intervals to a log file. Use the `instrumentation` section of the RADIUS configuration file (`BRM_home/apps/radius/config`) to enable this feature.

- To enable or disable the instrumentation feature, use the `status` entry.
- To specify the name of the log file, use the `file` entry.
- To specify in seconds the interval at which performance data is written to the log file, use the `interval` entry.

```
Instrumentation{
```
#status = disabled
#status = enabled
file = radius.statlog
interval = 10

instrument {
    queue
    ##threads
    ##packets
    ##performance
    ##request
    ##response
    ##fd
    ##module
    connections
    ##clients
    ##proxy
}

**Sample Log File for Queues**

=================================================================================
Logging Queue Stats. (Wed Jul 11 16:00:21 2001)
=================================================================================
current Active      = 0
current Size        = 1
current Waiting     = 1
high Size           = 1
low Size            = 0
max Size            = 300
high Active         = 0
high Waiting        = 1
average Size        = 1
duplicates detected = 0
queue overflows     = 0
last request spent 0ms in queue
average time requests spent in queue = 0ms
updated queue stats 1 times this interval
logging interval = 10 seconds
=================================================================================

**Sample Log File for Connections**

=================================================================================
Logging Connections Stats. (Wed Jul 11 16:00:21 2001)
=================================================================================
current Active    = 0
current Connects  = 4
current Dead      = 0
current Free      = 4
max Active        = 0
max Connects      = 4
max Dead          = 0
max Free          = 4
updated connection stats 1 times
logging interval = 10 seconds
=================================================================================
Configuring IP Service for an Account

You configure the following IP options for a customer account in Customer Center:

- The protocol the customer will use (PPP or SLIP).
- The IP address of the customer’s host machine (xxx.xxx.xxx.xxx).
- Whether or not header compression should be used.
- Any protocol extensions (attribute-value pairs that control the IP connection).

Sample Configuration File

This example shows sample settings for the $DEFINE and $CONFIG sections of the RADIUS configuration file ($RM_home/apps/radius/config):

```
$DEFINE {
  basic-ip-settings {
    send {
      Service-Type= Framed
      Framed-IP-Netmask=255.255.255.0
      Framed-IP-Address=255.255.255.254
    }
  }
}

$CONFIG {

  $CORE {
    listen {
      port = 1812
    }
    listen {
      port = 1813
    }
    log {
      file=/var/portal/__VERSION__/radius/radius.pinlog
      level=2
    }
    debug {
      #all
    }
    dictionary=/opt/portal/__VERSION__/apps/radius/dictionary
    pidfile=/var/portal/__VERSION__/radius/pin_radiusd.pid
    n_threads=4
    req_queue=40
    fdlimit=64
    client {
      addr = nas1
      secret = testing123
    }
    #options = Ignore-Acct-Auth
  }
```

```
Starting and Stopping the RADIUS Daemon

**Note:** You normally stop and start the RADIUS daemon to activate changes you made to the RADIUS configuration file (`BRM_home/apps/radius/config`). Some configuration changes can be activated dynamically (without restarting the RADIUS daemon). See "Reconfiguring Your RADIUS Server without Stopping Operation".

To start or stop the RADIUS daemon:

**Important:** Ensure that the RADIUS configuration file and dictionary file are in the `BRM_home/apps/radius` directory.

- To start the RADIUS daemon, run the `BRM_home/bin/start_radius` script.
- To stop the RADIUS daemon, run the `BRM_home/bin/stop_radius` script.

**Important:** If this is the first time you are running the `stop_radius` script, ensure that the parameters in the script are configured for your system. See "Configuring the stop_radius Script".

Reconfiguring Your RADIUS Server without Stopping Operation

Use the `pin_radiusd_sig` utility to reconfigure many RADIUS server functions without stopping operation. RADIUS server reconfiguration is implemented by making changes to the configuration file and then sending a reconfiguration event to the server. See "pin_radiusd_sig".

About the `pin_radiusd_sig` Utility

`pin_radiusd_sig` reconfigures RADIUS server functions while the server is running, extracts status information about the RADIUS server, and restarts the RADIUS server. `pin_radiusd_sig` reconfigures the following RADIUS server functions:

- Log file and log level.
- Debugging on/off.
- Number of threads.
- Size of the request queue.
- Listening ports.
- NAS clients and secrets.

The following RADIUS server features cannot be dynamically reconfigured:

- The data dictionary.
- Modules incorporated into the server.
- `mod_pin` configuration parameters:
  - `timezone`
What’s Next?

- network
- charge-on-reboot
- CM connection information

What’s Next?

After you configure the core settings and define configuration macros in the $CORE and $DEFINE modules of the RADIUS configuration file, you need to configure the modules used by RADIUS Manager for handling RADIUS requests. See "Understanding RADIUS Manager Modules".
This chapter describes how to select, edit, and customize the RADIUS data dictionary file.

**About the Data Dictionary**

The data dictionary includes a list of the attribute-value pairs which are used by Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager to perform AAA and other operations. The RADIUS dictionary uses standard attributes prescribed by the RADIUS protocol in Request for Comments (RFC) 2865 and 2866. The default location of the data dictionary file is `$BRM_home/apps/radius/dictionary_file`, where `BRM_home` is the directory in which BRM components are installed.

Attribute-value pairs are sent from the NAS to RADIUS Manager as RADIUS requests. Each attribute-value pair consists of an attribute with a value for that attribute.

The syntax for attribute-value pairs:

**Syntax:**

```
<attribute_declaration> <attribute_name> <attribute_number> <data_type>
```

**Examples:**

```
ATTRIBUTE NAS-IP-Address 4 ipaddr
ATTRIBUTE User-Name 1 string
```

---

**Important:** Attribute numbers must be unique.

---

**Defining the Data Dictionary**

Use the `dictionary` entry in the `SCORE` section of the RADIUS configuration file (`BRM_home/apps/radius/`) to set the file name and path of the RADIUS dictionary file. See "Selecting the Data Dictionary".

**Editing the Data Dictionary**

In special cases where you are using NAS from multiple vendors, or must add vendor specific attributes to the dictionary file, you may need to edit the data dictionary.

Only merge data dictionary files as a last resort. Each attribute value pair in the dictionary must be unique. When you merge data dictionaries review the file carefully to ensure that no two attributes have the same name. If duplicate attributes exist, modify the attribute names so that they are unique.
Adding Vendor Specific Attributes to the Data Dictionary

To use a vendor specific attribute (VSA), you must define the attribute and vendor code size in your dictionary file.

The syntax for defining a vendor-specific attribute in your RADIUS dictionary file:

```
VENDORATTR <vendor_id> <attr_name> <vendor_type> <type> [struct defn.]
VENDORATTR 9Example_VSA_struct 89 struct
ip:ipaddr,vector:data[10],id:short
```

**vendor_id**
Number used to identify the NAS or gateway vendor. These numbers are assigned by the Internet Advisory Board (IAB). Check your vendor’s documentation for details. Some common vendor identification numbers are:

- 9 (Cisco)
- 311 (Microsoft)
- 429 (3Com/USR)

**attr_name**
Name of the attribute, User-Name for example.

**vendor_code**
Identification number assigned to the attribute in the dictionary.

**data_type**
Any one of several supported data types.

- **UnsignedInt**
  32 bit unsigned value in big endian order (high byte first).
- **Integer**
  32-bit value in big endian order (high octet first).
- **String**
  0-253 octets
- **Ipaddr**
  4 octets in network octet order
- **Binary**
  0-254 octets
- **Password**
  \((n * 16) \geq 16\) octets. This field is encrypted according to the User-Password definition in RFC 2865.
- **Short** – 16-bit value
- **Octet** – 8-bit value
- **Data** – 0 to 253 octets, whose length is fixed when you define it. This data type must be used inside a **struct**; it can’t be used as a stand-alone data type.
- **Struct** – A composite data type that can use the other supported data types as subfields. A **struct** can’t contain another **struct**.
Adding the Vendor Code Size to Your Data Dictionary

You must also add the `VENDOR_CODE_SIZE` field to your dictionary file before using a VSA.

The syntax for entering a `VENDOR_CODE_SIZE` field in your RADIUS dictionary file:

```
VENDOR_CODE_SIZE <vendor_id> <code_size>
```

- **VENDOR_CODE_SIZE**
  Keyword that specifies the size of the vendor code field in the VSA.

- **vendor_id**
  Number used to identify the NAS or gateway vendor. These numbers are assigned by the Internet Advisory Board (IAB). Some common vendor identification numbers are:
  - Cisco 9
  - Microsoft 311
  - 3Com/USR 429

- **code_size**
  Size of the vendor code contained in the VSA. This value must be 1, 2 or 4 octets.

If you use one of the following NAS vendors, you can copy the appropriate `VENDOR_CODE_SIZE` entry to your dictionary file. If your NAS vendor isn’t listed, check the vendor’s documentation for details or use the default size. The default size is 1 octet.

```
VENDOR_CODE_SIZE 9 1 # Cisco
VENDOR_CODE_SIZE 429 4 # 3Com/USR
VENDOR_CODE_SIZE 311 1 #MICROSOFT
VENDOR_CODE_SIZE DEFAULT 1 # Default vendor code size
```

When the dictionary is initialized, these values are read and stored in BRM.

Using the struct Data Type in a VSA

Usually a predefined vendor code exists for a VSA. In special cases, where no such vendor code exists, or where you as the administrator require additional flexibility in defining attribute parameters, the struct data type can be used.

**Caution:** Only advanced administrators should attempt to use this feature.

The format of the first five fields in all VSA definitions are the same for all supported data types. When you use the struct data type, the fields following the `data_type` field are a comma-separated list of field-name and field-type pairs.

This example shows three subfields in the `struct`:

- **IP address** field named `ip`
- Field of type `data` with a length of 16 bits called `vector`
- Field of type `short` named `tunnel`
Editing the Data Dictionary

---

**Note:** Each VSA definition must fit on one line. Definitions will not work if they are split over more than one line.

---

VENDORATTR 429 VPN 123 struct ip:ipaddr,vector:data[16],tunnel:short

**Limitations to Using the Struct Data Type in a VSA**

Some data types are not allowed to be used in a struct.

These data types are permitted:

- int
- short
- octet
- data
- ipaddress

These data types are not permitted:

- struct
- string
- binary
- password
Understanding RADIUS Manager Modules

This chapter presents concepts that are essential to understanding the standard modules available with Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager.

Introduction to Modules

Eight general purpose modules types are included with RADIUS Manager. These utilities handle RADIUS requests. Each module type has a unique name, such as mod_pin, mod_null, and mod_text, along with code that implements its unique behavior. Each module type is implemented as a pair of C++ classes: a module master class, and a module worker class.

There are three important module concepts: module type, module master class, and module worker class.

About Module Types

Module types are defined in the RADIUS configuration file (BRM_home/apps/radius/config, where BRM_home is the directory in which BRM components are installed) by using the type element. You define global configuration settings module types in the $CONFIG section of the RADIUS configuration file (BRM_home/apps/radius/config). This ensures that the same configuration is used across all modules of that type.

This example specifies the configuration of all modules of type mod_pin:

$CONFIG {
  ...
  mod_pin {
    timezone=28800
    network=my-network
    charge-on-reboot=NC
    domain-separator='@'
  }
  ...
}

About Module Master Classes

For each module type, there is a module master class implementation. A module master class is instantiated for each entry of a module type RADIUS configuration file (BRM_home/apps/radius/config). For example, if the RADIUS configuration file contains entries for two modules of type mod_logging, two module master classes of type mod_logging are created.
The module master is responsible for:

- Reading and interpreting global module type configuration information
- Reading and interpreting module specific configuration information
- Instantiating (creating) module workers as required.

**About Module Worker Classes**

There is a module worker class implementation for each module type. A module worker object of the appropriate class is instantiated for each module entry in the RADIUS configuration file. (BRM_home/apps/radius/config) for each worker thread. For example, if the RADIUS configuration file contains entries for two modules of type `mod_logging`, and two worker threads are configured, four module worker objects of type `mod_logging` are created.

Module workers are responsible for processing requests as they arrive.

**Understanding External Communication and Support Module Types**

The eight general use modules provided with RADIUS Manager can divided into two general categories: external communication modules and support modules.

**About External Communication Modules**

External communication, or terminating module types, check an incoming request and either accept the request or pass it on to the next module instance in the chain. If the request is accepted, it is processed. After the request is processed successfully, the required attributes are appended to the request, and the request is sent to the NAS. The external communication modules include:

- **mod_pin.** Performs standard BRM authentication and accounting. Also provides support for VPDN and iPass integration.
- **mod_text.** Authenticates requests using plain-text user, login, and password information.
- **mod_unixpwd.** Authenticates requests using a UNIX password.
- **mod_proxy.** Forwards requests to another RADIUS server.
- **mod_null.** Selectively replies ACK, NAK, ignore, or discard requests. Handles incoming authentication and accounting requests before special processing of the packets is performed. Used as the last module in a module chain.

**About Support Modules**

Support, or non terminating, modules check an incoming request, and either accept the request or pass it on to the next module worker class in the chain. If the request is accepted, it is processed. After the request is processed successfully, the request is passed on to the next module worker class in the chain. The module may or may not alter the contents or attributes of the request. Unlike terminating modules, non-terminating modules never reply to the NAS. The non-terminating modules include:

- **mod_logging.** Logs RADIUS requests to text files.
- **mod_cdr.** Logs RADIUS requests using a record oriented, field-separated format.
- **mod_transform.** Transforms attribute-values using a basic transformation function.
■ **mod_unit.** Scales unsigned integer values to signed integer values.

### About Module Chains

Modules are strung together in a series to form module chains. The module chain performs a series of functions as defined by the specific worker modules configured into the chain.

When a request is received by the RADIUS server, it is placed in the queue. The next available thread picks up the request and processes it within the module chain. How the request is processed depends on:

- which modules are configured into the chain
- the order in which modules are configured
- whether the module is an external communication module or a support module

In order for the module chain to work as intended each module in the chain must be configured to pass a request to the next module using specific instructions for accepting, failing, or ignoring the request.

### How RADIUS Requests Are Processed Using check and send Requests

Each module in the chain includes its own instructions for how to handle a RADIUS request. The primary components of these instructions are the **check** and **send** elements. These are defined in the $DEFINE section of the RADIUS configuration file.

#### About the check Element

All worker class modules use the **check** element. The **check** element attempts to match attribute-value pairs in the request with specific criteria:

- If the request meets the criteria set in the **check** element, then the request is processed.
- If the request does not meet the criteria set in the **check** element, then the check fails. A failed request is passed on to the next module worker class in the chain.

This example shows the syntax for a **check** element. In order for this request to be processed, the request must be from the IP address 10.1.2.3, and the user name must end with @domain.com.

```plaintext
check {
    NAS-IP-Address = 10.1.2.3
    User-Name = "**@domain.com"
}
```

#### About the send Element

The attributes set in the **send** element determine which attribute-value pairs are appended to the reply to the NAS. Only external communication module worker classes, which communicate with the NAS, use the **send** element.

This example shows the syntax for a **send** element. In this example, the reply packet will be appended to contain the Framed-MTU and Ascend-Idle-Limit attributes.

```plaintext
send {
    Framed-MTU=892
    Ascend-Idle-Limit = 600
}
```
This chapter describes how to use the authentication, authorization and accounting (AAA) modules in Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager. Features of these AAA modules are also described:

- BRM authentication and authorization module - mod_pin
- Plain text authentication module - mod_text
- Password authentication module - mod_unixpwd
- Forwarding authentication requests to an external RADIUS server - mod_proxy
- Selectively responding to RADIUS requests - mod_null

Before reading this chapter you should be familiar with "Understanding RADIUS Manager Modules".

**Authentication Concepts**

To design module chains which process AAA requests, you need to understand these concepts:

- local authentication
- remote authentication
- cascading authentication

**About Local and Remote Authentication Requests**

Any local authentication request is a known request since you authenticate using the BRM database. RADIUS Manager uses the module, mod_pin, to perform authentication, authorization and accounting functions against your BRM database. All local BRM users log in with their user name. This is the default authentication method.

An authentication request from a remote server is an unknown request since AAA functions cannot be performed against the BRM database. BRM supports both VPDN and iPass roaming for remote users. Remote users log in using their full email address, including the user name, @ symbol, and domain name. Two types of authentication are available for managing unknown requests:

- **Password authentication**. This method uses the mod_unixpwd module for password authorization.
Plain text password authentication. Uses the mod_text module to perform authentication against a text list of user names and passwords.

Unknown authentication requests require multiple authentication methods and are handled using a cascading authentication scheme.

About Cascading Authentication

Remote, or unknown, authentication requests require that you use more than one authentication method. To do this you must configure the RADIUS server to perform cascading authentication. This allows authentication requests to cascade from the most restrictive authentication check to the least restrictive check. Cascading authentication ensures that all requests are processed by the appropriate module.

For example, a check for *@isp.com should be performed before a check for *@*. If the less restrictive *@* check is placed before the more restrictive *@isp.com check, the request will never reach the more restrictive check. This is important for unknown authentication requests using iPass and VPDN where your RADIUS server does not perform the authentication. For information, see "Using a Virtual Private Dialup Network (VPDN)".

About Configuring Modules for Cascading Authentication

Three module configuration entries directly affect how requests are processed through a module chain which uses cascading authentication. These entries define how requests are passed from one module to another:

- Ignore-No-Entry: Allows a failed accounting request to pass to the next module in the chain. This entry is supported by all modules except the proxy authentication module (mod_proxy).
- Ignore-No-Response: This entry is used by the proxy authentication module (mod_proxy) It allows a failed authentication request to pass on to the next module worker class in the chain.
- Class: The Class attribute is appended to accounting requests during authentication. Use it to indicate which module should process the accounting request. The Class attribute is specified in the RFC 2865 and should be supported by your NAS.

Managing Cascading Authentication Requests with the Class Attribute

Use the Class attribute whenever you use multiple authentication methods. The Class attribute ensures that accounting requests are handled efficiently.

For example, if the authentication request is from an iPass user, the reply packet is appended to contain the Class=iPass attribute-value pair. When the corresponding accounting request is received by the RADIUS server, the module worker classes checks the Class attribute to determine whether it should process the accounting request, or pass it to the next module worker.

This example shows the syntax for appending the Class attribute. In this example, the authentication request is for an iPass account. The send element is used to append the attribute-value pair iPass to the corresponding accounting request.

```plaintext
...
#Authentication Request#
{
    type = mod_text
    check {
        Pseudo-Request-Type = Authentication-Request
```
Using the BRM Authentication and Accounting Module

The module **mod_null** must always be the last module in the chain. Otherwise, incoming authentication and accounting requests are misdirected before special processing of the packets is performed. For information on **mod_null**, see "Selectively Responding to RADIUS Requests with the null Module".

### Using the BRM Authentication and Accounting Module

The **mod_pin** module provides authentication and accounting support, and performs standard check processing for RADIUS requests. The **mod_pin** module supports:

- User accounts with a `/service/ip` service
- VPDN authorization and accounting.
- iPass integration.

### How the BRM Authentication Module Processes Requests

An authentication request is sent to the BRM database to authenticate a user name.

- If the user name is found and authentication succeeds, the request is sent to BRM. The request includes any `send` attributes required to find the session setup information for the user.
- If the user name is not found, the `Ignore-No-Entry` option specifies whether processing continues or **NAK** is returned.
- If the user name is found, but authentication fails, **NAK** is returned.
- If the authorization succeeds, an **ACK** is returned, including any attributes returned from the opcode.
- If authorization fails, **NAK** is returned and the user is denied access.

### Configuring Global Settings for the BRM Authentication Module

This section describes the global settings that can be defined for the BRM authentication module (**mod_pin**). You configure these settings in the `$CONFIG` section of the RADIUS configuration file (`BRM_home/apps/radius/config`, where `BRM_home` is the directory in which BRM components are installed).

### Specifying the Time Zone

Set the `timezone` entry to define the time zone where your RADIUS Manager is physically located. RADIUS Manager sends the information in the `TIMEZONE_ID` string to the BRM server. The BRM server uses the time zone for rating purposes. If the
time zone is not set, the default setting NULL allows real-time rating to use the RADIUS server's time zone. In the case where the RADIUS Manager, NAS, and end user are in different time zones, RADIUS Manager uses the time zone of the NAS.

The time zone string should use standard Java TimeZone ID format. Wherever possible, use the unambiguous long descriptive format. (The short abbreviation is error-prone.)

---

**Note:** A list of TimeZone IDs can be obtained by downloading the public domain time zone database (commonly known as "zoneinfo") `tzdata2000d.tar.gz` from `ftp://elsie.nci.nih.gov/pub`. Since there is no ISO or any other international standard for time zone names, this database can be considered as the standard. Once this file has been downloaded and extracted, the third column in the `zone.tab` file is the Timezone ID.

---

timezone-id = "America/Los_Angeles"

If no standard TimeZone IDs are available for a particular time zone, then RADIUS Manager can accept encoded custom time zone strings in the format "GMT+14." You must set the number of seconds + (West) or - (East) of GMT.

**Specifying How Accounting-On Events Are Handled after a NAS Failure**

The `charge_on_reboot` entry controls how Accounting-On events are handled after a NAS system crash. Normally, a user is billed for the period between session start and session end. If the NAS crashes, the session is unexpectedly terminated without the standard session end calls.

The `charge_on_reboot` entry allows you to define how Accounting-On events are billed once the NAS has been restarted. Two values are available, charge and no charge.

- If no charge (nc) is selected, then the user will not be charged for the period of the system crash. Session billing begins when the NAS is restarted and the Accounting-On events are sent. No charge (nc) is the default.
- Selecting the charge (c) value will charge the user for the period between the original session start and the NAS restart when the Accounting-On events are sent to BRM. Selecting this option bills the customer for the entire period including the time of the system crash.

```
charge-on-reboot=NC
```

**Specifying the Network Name**

You can use the `network` entry to list the network name. This entry is used only when you need to track audit information on multiple networks. If the network name is supplied, it is stored in the `/event/session/dialup` storable object.

**Specifying the Domain Name Separators**

Use the `domain-separator` entry to define the separator between the user name and domain name in an address like `joe@company.com`. The default is "@". This information is used by the `Use-Domain-Name` option.

```
domain-separator="@"
```
Example of mod_pin global Settings

```
... 
mod_pin {
    timezone-id = "America/Los_Angeles"
    network=my-network
    charge-on-reboot=NC
    domain-separator="@"
}
```

Configuring Instances of the BRM Authentication Module

This section describes the settings you can configure for instances of **mod_pin** in the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config). The mandatory **type** entry is **mod_pin**.

Specifying How RADIUS Requests Are Processed

Use the **check** and **send** requests specify how RADIUS requests are processed.

```
type=mod_pin
check {
    attribute = value ...
    ...
}
send {
    attribute = value ...
    ...
}
```

---

**Note:** You should configure **mod_pin** module after configuring all the other modules, as it prepares the RADIUS response and sends the response back to the client. Due to this, post processing of opcode is not possible when the **mod_pin** is being used.

---

Allowing a Request to Be Searched by Multiple Databases

Use the **options-Ignore-No-Entry** entry when multiple authentication methods are configured in a cascading chain. If this value is set and a matching user name is not found in the BRM database, the request is sent to the next authentication module in the module chain. This allows an access request to be searched by several different databases.

You can also use the **Ignore-No-Entry** option when authentication needs to be performed only on the domain name (The domain name is the string after the "@" sign in the **User-Name** attribute.) If there is no "@" sign, the login name to validate against is empty ("")

```
options {
    Ignore-No-Entry
}
```

Tracking Access Requests in BRM but Using Another Database for Authentication

Use the **pass-through** entry when multiple authentication methods are configured in a cascading chain. When the **pass-through** value is set, the request is first sent to the BRM database. If a matching user name and password are found, the request is recorded and forwarded to the next module in the chain.
Use this entry when you need to track access requests in BRM, but use another database for authentication. No response is sent back to the NAS, so that the mod_pin module behaves as a support or non-terminating module.

**Important:** When the pass-through value is selected, mod_proxy must be the next module in the chain.

### Authenticating against a Fixed Login String

Use the login entry with organizational models, such as VPDN and iPass, when you need to authenticate using only the domain name of the login name. This entry can be set to either a fixed string, like "ipass", or a string-replacement transformation can be applied to it. The default is the value in the User-Name attribute.

```plaintext
login = [fix_string | "Replace/target/replaceStr/*"]
```

### Recording the User Login for Audit Purposes

Use the desc entry when you need to track usage information for organizational accounts. Setting this entry provides audit information in organizational settings where many users are using a single login account.

For example, if all users from an organization are using the account user@corporate.com, but you need to know which users actually use the account, then you can record user names by setting the desc entry. If a desc value is specified, the actual user login name is recorded as a free text description and stored in the PIN_FLD_DESC field in the input flists.

```plaintext
desc = "... %u ...
```

**Note:** The %u setting maps to the user login and is the only available option.

### Routing and Tracking Service Requests by Subtype

Use the service_type entry to track and route subtypes of the /service/ip request type. The service_type entry is useful when an Internet service provider (ISP) offers multiple service types and it is necessary to track requests for each different service type (such as /service/ip/email and /service/ip/broadband). The service_type entry can also be used when you need to route different types of IP service requests to different NAS for authentication.

You can use this entry to define any service type. It is usually used for organizational models such as VPDN. The default value is /service/ip.

```plaintext
service_type = /service/[ip | vpdn | other_services]
```

### Example settings for an instance of mod_pin

This example shows configuration settings for an instance of mod_pin:

```plaintext
type=mod_pin
check {
   attribute = value ...
   ...
}
send {
   attribute = value ...
   ...
```
Sample Code Showing How the BRM Authentication Module Processes Requests

This sample code is configured to authenticate all requests against BRM. If the Access-Request fails, NAK is not returned. The request is passed on so other authentication mechanisms, such as VPDN or iPass, can be tried.

```plaintext
# Authenticate against Portal
auth_portal {
  type=mod_pin
  $INCLUDE=Basic-PPP
  check {
    Psuedo-Request-Type=Access-Request
  }
  send {
    # Send Class=PIN so we can match up accounting requests.
    # By returning it to the NAS, the NAS is required to return
    # this attribute in Accounting requests.
    Class=PIN
  }
  # If the user isn't in Portal, don't stop
  options=Ignore-No-Entry
}

# Store accounting info in Portal.
# Note the match on Class=PIN to ensure that only sessions
# which were authenticated by Portal have the accounting
# data stored in Portal.
acct_portal {
  type=mod_pin
  check {
    Psuedo-Request-Type=Accounting-Request
    Class=PIN
  }
}
```

Authenticating Access Requests Using Plain Text

The mod_text module lets you authenticate access requests using plain-text user login and password verification. Only Access-Request requests are considered by this module, other requests are ignored.

How the Plain Text Authentication Module Processes Requests

- Standard check processing is performed against a request.
- If the user name is not found, the Ignore-No-Entry option determines if processing continues or NAK is returned.
- If a match for the user name is found, the User-Name value is compared with the elements of the password.
- If a password match is found, ACK is returned along with any send attributes.
If no password match is found, **NAK** is returned.

**Configuring the Plain Text Authentication Module**

No global configuration settings are supported for the **mod_text** module. You configure each instance of the **mod_text** module in the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config). The mandatory **type** entry is **mod_text**.

**Defining the List of User Names and Passwords**

In the **user** block include a list of **username=username** and **password=password** pairs which specify the cleartext password for each user name. If the password has special characters, enclose them in quotation marks.

```plaintext
user {
  username = userxxxx
  password = passxxxx
}
```

**Specifying How the Plain Text Authentication Module Processes Requests**

Use the **check** and **send** requests to specify how RADIUS requests are processed. See "How RADIUS Requests Are Processed Using check and send Requests".

```plaintext
check {
  attribute = value ... 
  ... 
  send {
    attribute = value ... 
    ... 
```

**Allowing Requests to Be Searched by Multiple Databases**

Use the **options-Ignore-No-Entry** entry to allow an access request to be searched by several different databases. You can also use the **Ignore-No-Entry** option when authentication needs to be performed only on the domain name.

```plaintext
options {
  Ignore-No-Entry
}
```

**Example Settings for mod_text**

```plaintext
text_auth {
  type=mod_text
  check {
    attribute = value ... 
    ... 
  }
  user {
    username = userxxxx
    password = passxxxx
  }
  ... 
  send {
    attribute = value ... 
    ... 
  }
  options {
    Ignore-No-Entry
```
Using Password Authentication Module

The mod_unixpwd module performs authentication against a password file. Only Access-Request requests are processed by this module; other requests are ignored.

---

**Note:** CHAP authentication is not supported by this module because CHAP requires access to clear-text passwords. Attempting to use password authentication with CHAP introduces conflicting encryption methods and fails the procedure.

---

How Passwords Are Used for Authentication

When an access request is sent to the mod_unixpwd module a check is performed against a password database:

- If the user name is not found, the Ignore-No-Entry option specifies whether processing continues or NAK is returned.
- If a match is found, the domain name is stripped from the User-Name attribute, and checked against the password database.
- If a matching User-Name is found, the user password is checked against the password database.
- If the password also matches, ACK is returned along with any send attributes, otherwise NAK is returned.

---

**Note:** For security reasons, the User-Name value "root" always fails.

---

Configuring Password Authentication

No global settings are supported for the mod_unixpwd module. You configure each instance of the mod_unixpwd module in the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config). The mandatory type entry is mod_unixpwd.

Specifying How RADIUS Requests Are Processed

The check and send requests specify how RADIUS requests are processed.

Allowing Requests to Be Searched by Multiple Databases

Use the options-Ignore-No-Entry entry to allow an access request to be searched by several different databases. You can also use the Ignore-No-Entry option when authentication needs to be performed only on the domain name.

```bash
options {
    Ignore-No-Entry
}
```

Example Settings for the mod_unixpwd

This configuration is supported.

```bash
type=mod_unixpwd
```
check {
  attribute = value ...
  ...
}
send {
  attribute = value ...
  ...
}
options {
  Ignore-No-Entry
}

Using the Proxy Module to Forward Incoming Requests to an External RADIUS Server

The **mod_proxy** module forwards incoming requests to an external RADIUS server. All request types are accepted by this module. This module does not support retransmission because it assumes that the request source performs retransmission. Only one **mod_proxy** module can be used within a cascading authentication configuration.

This module is useful in a variety of scenarios. For example, if you have a legacy RADIUS server and do not want to perform data translation, you can forward requests from RADIUS Manager to the legacy server for authentication. This module is also useful with iPass roaming, since it allows requests to be forwarded from a remote server back to the home ISP for processing.

How the Proxy Module Forwards Requests to Another Server

You can configure the **mod_proxy** module to proxy (forward) requests of a given type from one server to another server.

- When a response is received from the external server, that response is passed back to the client after appending any **send** attributes.
- If no response is received within the specified time-out period, the request is ignored or returns NAK according to the **Ignore-No-Response** option.
- The original packets are not forwarded unmodified. A new RADIUS packet is created based on a new authenticator containing the shared secret for the destination and on the contents of the input request.

Configuring Global Settings for the Proxy Module

You configure these global settings in the **$CONFIG** block of the RADIUS configuration file (**BRM_home/apps/radius/config**). Information about both the remote authorization and remote accounting destination server must be set in the server blocks of each **mod_proxy** instance.

Specifying a Name for the Destination Server

The **name** is a label used by BRM to reference the destination server(s). Referencing this name in a module configuration provides access to the server information listed in the server block.

```plaintext
name = remote-auth
```
Specifying the IP Address of the Destination Server
Use the `addr` entry to list the IP address or host name of the server to which requests are to be forwarded.

```plaintext
addr = remote-host
```

Specifying the Authorization and Accounting Ports
Use the `port` entry to set the IP port to which access requests should be sent. If you define the authorization port using the `authport` entry then you must also define a port for forwarding accounting requests. Use the `acctport` entry to define the accounting port.

Example for authorization port:

```plaintext
server {
  name = remote-auth
  addr = remote-host
  port = 1812
}
```

Example for accounting port

```plaintext
server {
  name = remote-acct
  addr = remote-host
  port = 1813
}
```

Setting the RADIUS Secret
Use the `secret` entry to define the secret shared with the destination server. Make sure this secret is also listed in the client block of your data dictionary. For information, see "Defining the Secret".

```plaintext
secret = testing123
```

Specifying the Number of Seconds till Time Out
Use the `timeout` entry to specify the number of seconds until `mod_proxy` gives up trying to receive a response from the external server.

```plaintext
timeout = 10
```

Example of Global Settings for `mod_proxy`

```plaintext
mod_proxy {
  server {
    name = remote-auth
    addr = remote-host
    port = 1812
    secret = testing123
    timeout = 10
  }
  server {
    name = remote-acct
    addr = remote-host
    port = 1813
    secret = testing123
    timeout = 10
  }
}
```

Configuring Instances of the Proxy Module
Use these options to define the settings for each instance of the `mod_proxy` module.
Specifying How to Process RADIUS Requests
The check and send requests specify how RADIUS requests are processed. See "How RADIUS Requests Are Processed Using check and send Requests".

```
proxy-auth {
    check {
        Pseudo-Request-Type = Access-Request
    }
    dest=server1
    send {
        Class = "proxy"
    }
}
```

Allowing Requests to Be Searched by Multiple Databases
Use the options-Ignore-No-Entry entry to allow an access request to be searched by several different databases. You can also use the Ignore-No-Entry option when authentication needs to be performed only on the domain name.

```
options {
    Ignore-No-Entry
}
```

Allowing Failed Requests to Be Forwarded
Use the Ignore-No-Response option to pass an authentication request that cannot be validated on to the next module. The mod_proxy module waits till the end of the timeout period for a response from the server. After the timeout period, the request is sent to the module in the chain.

Use this option when you wish to configure a back up authentication system. If a remote server is down, this option forwards the request to another server for authentication.

```
options {
    Ignore-No-Response
}
```

Using a Non-BRM Database for Authentication
Use the pass-through entry when multiple authentication methods are configured in a cascading chain. When the pass-through value is set, the request is first sent to the BRM database. If a matching user name and password are found, the request is recorded and forwarded to the next module in the chain. Use entry when you need to track access requests in BRM, but use another database for authentication. No response is sent back to the NAS, so that the mod_pin module behaves as a support or non-terminating module. When the pass-through value is selected, mod_proxy must be the next module in the chain.

```
Using the Proxy Module to Forward Incoming Requests to an External RADIUS Server
```

Specifying the Destination for Forwarded Requests
Use the dest entry to specify a single destination to which requests should be forwarded. The server details are retrieved from the type-specific configuration.

```
dest=server1
```

Sample Code for Forwarding Requests with mod_proxy
These sample code show how to forward Access-Request and Accounting-Stop packets.
# mod_proxy configuration to proxy Access-Request packets to server1
$CONFIG {
    mod_proxy {
        server {
            name = server1
            addr = zap
            port = 1814
            secret = testing123
            timeout = 10
        }

        server {
            name = server2
            addr = zap
            port = 1815
            secret = testing123
            timeout = 10
        }

        server {
            name = server3
            addr = zap
            authport = 1814
            acctport = 1815
            secret = testing123
            timeout = 10
        }
    }
}

$MODULES {
    # mod_proxy configuration to proxy Access packets to server1
    proxy-auth {
        check {
            Pseudo-Request-Type = Access-Request
        }
        dest=server1
        send {
            Class = "proxy"
        }
    }

    # mod_proxy configuration to proxy Accounting-Stop packets to server2
    proxy-acct {
        check {
            Pseudo-Request-Type = Accounting-Request
        }
        dest=server2
        check {
            Acct-Status-Type=Stop
        }
    }

    # mod_proxy configuration to proxy Access and Accounting packets to server3
    proxy-host {
        dest=server3
    }
}
Selectively Responding to RADIUS Requests with the null Module

The mod_null module allows selected requests to receive an ACK or NAK reply. This module accepts Access-Request and Accounting-Request packets but ignores the requests. It is useful as the last module in the module chain, or to put the RADIUS server in the promiscuous mode. Promiscuous mode is an emergency alternative which ACKs all requests when the database is down.

Note: The last module in the chain must be mod_null, otherwise incoming authentication or accounting requests would be NAK or ACK respectively, before special processing of the packets.

Configuring mod_null

No global configuration settings are supported for the mod_null module. You configure each module instance of the mod_null module in the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config). The mandatory type entry is mod_null.

Specifying How to Process RADIUS Requests

The check and send requests specify how RADIUS requests are processed. See "How RADIUS Requests Are Processed Using check and send Requests".

check {
    attribute = value ...
    ...
}
send {
    attribute = value ...
    ...
}

Defining the Action to Take for Matching Requests

Use the action block to define the list of possible actions to take for matching requests:

- **ack**: Creates an Access-Accept or Accounting-Response packet for sending to the client, including any send attributes. Unrecognized request types are discarded. Valid for Access-Request and Accounting-Request only.
- **nak**: Creates an Access-Reject or Accounting-Response packet for sending to the client, including any send attributes.

Note: Since all accounting responses are processed, the action of NAK for accounting requests is equivalent to specifying ACK. This behavior simplifies the configuration of support module entries. Unrecognized request types are discarded.

- **discard**: Discards a request and takes no other action. Use this option to prevent a request from being processed by another module. This entry is valid for any request type.
- **ignore**: Ignores a request and takes no action. This entry is valid for any request type.
Example Showing Settings for an Instance of mod_null

```
type=mod_logging
check {
    attribute = value ...
    ...
}
send {
    attribute = value ...
    ...
}
action {
    ack | nak | discard | ignore
}
```

Sample Code Showing How mod_null Processes Requests

The example below show how to NAK all Framed-User authentication requests and drop Accounting-Start requests:

```
# mod_null configuration to NAK all Framed-User authentication requests
disable_framed {
    type=mod_null

    action=nak
    check {
        Pseudo-Request-Type = Access-Request
        User-Service = Framed-User
    }
    send {
        Reply-Message = "All Framed-User logins temporarily disabled"
    }
}

# mod_null configuration to drop Accounting-Start requests
drop_acct_start {
    action=ack
    check {
        Pseudo-Request-Type = Accounting-Request
        Acct-Status-Type=Start
    }
}
```
This chapter describes how to configure the logging modules available with Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager.

**Caution:** Only administrators with advanced RADIUS skills should attempt to implement custom configurations.

Before reading this chapter you should be familiar with:

- Understanding RADIUS Manager Modules
- Configuring RADIUS Manager

RADIUS Manager provides two logging modules. The most basic logging module is `mod_logging`, used to log RADIUS requests to text files. The module `mod_cdr` enables you to create custom CDR log files.

### Logging Requests to Text Files

Use the `mod_logging` module to log RADIUS requests to text files. This is the simplest logging module available with RADIUS Manager. There is less overhead using this module than with custom log files. The resulting log files are human readable so that they can be used for debugging purposes. This module allows you to mask attributes, such as user name and password, for security reasons.

All RADIUS request types are accepted by this module. Once requests are processed they are passed to the next module in the module chain. Requests are logged to the log files you specify.

### Using mod_logging

The `mod_logging` module can be configured to log requests of any type. This is a support module, so no response is sent back to the client NAS.

- Each element of the incoming request is checked against the corresponding element in `mod_logging` block of the RADIUS configuration file `BRM_home/apps/radius/config`, where `BRM_home` is the directory in which BRM components are installed.
- If a match is found, the request is logged according to the configuration parameters.
- Searching continues with the next configuration element in the request. This permits a single request to be logged to multiple destinations.
Requests are logged to the specified files using a fixed, text-based format which includes the date, time, and source of the request.

Configuring mod_logging

No global ($CONFIG block) configurations are supported. Configure each instance of the mod_logging module in the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config). The mandatory type entry is mod_logging.

Checking Requests

Use the check element to search for matching elements. The check element matches attribute-value pairs in the request. If the value pair in the request matches the criteria set in the check element, the request is processed. If any attribute in the check element is not matched in the request, then the check fails. After the check fails, the request is passed on to the next module in the chain.

Masking Attributes

The mask entry allows you to mask out attributes. Masking allows you block out attributes for security reasons. In general, it is a good idea to block out the user name and password as well as any other sensitive user information. Any attribute in the data dictionary can be masked out. Masked attributes are shown as "XXXX" in the log file.

Specifying the Destination File

Use the dest element to specify the destination file where RADIUS requests are to be logged. If the directory you specify doesn't exist, it is created. Any instance of %n in this value are expanded with the NAS identifier. This uses the standard algorithm:

- If the NAS-Identifier exists, use it;
- Otherwise, if the NAS-IP-Address exists, use it;
- Otherwise, use the address of the source of the request.

---

Note: No options are available for this module.

---

Examples Using mod_logging

The following example logs Accounting-Stop requests to a detail file.

```
# logging configuration to log all Accounting-Stop requests to a detail file
logging_stop {
    type=mod_logging
dest="/opt/portal/radius/%n/detail*example"
check {
    Acct-Status-Type=Stop
    Pseudo-Request-Type=Accounting-Request
}
}
```

```
# logging configuration to log all requests to a debugging file
# mask out the User-Password attribute (it shows as "XXXX")
logging_all {
    type=mod_logging
dest="/tmp/logs/%n"
mask {
    User-Password
}
In a scenario where the accounting request from a particular terminal server should just be logged and not processed, a `mod_null` module worker class should be added to the chain to perform the reply action. The following example shows how to configure the module to reply.

```ini
## Just want to log the accounting record to file
## Need to add a mod_null after mod_logging to reply to the NAS
just_log {
    type=mod_logging
    dest="/tmp/logs/%n"
    check {
        NAS-Identifier = "TESTING NAS"
    }
}

## Note that the check attribute is identical to [just_log] module worker class
just_log_reply {
    type=mod_null
    check {
        NAS-Identifier = "TESTING NAS"
    }
    action = ack
}
```

### Creating Custom CDR Log Files

Use the module `mod_cdr` to create custom call detail record (CDR) files for logging RADIUS events. RADIUS events such as access and accounting requests can be logged in a record oriented, field separated format. The format, name, and location of the log file, as well as any rotation schemes, can all be customized. This allows you to design log files which may be processed later using scripts, cron jobs, or other applications such as the UE Loader.

In addition, this module enables you to:

- Send RADIUS messages to different CDR log files.
- Use naming conventions and rotation schemes for log files across RADIUS invocations.
- Configure delimiters.
- Use string literals.
- Include multiple instances of `mod_cdr` to handle different scenarios and types of RADIUS messages.
- Use dynamic reconfiguration.

**Limitations of `mod_cdr`:**

- Does not create a response to send to a client NAS.
- Does not support masking of attributes.

### Scenarios for Creating Custom CDR Log Files

Custom log files generated by `mod_cdr` are useful in many different scenarios. The module, `mod_cdr`, is particularly useful in cases where log files need to be processed at a later time. For example:
Recovering usage information. If BRM goes down while RADIUS is sending data, then the mod_cdr module lets you continue to log incoming requests. Later when the system is back up, this information can be imported into BRM using an application such as UE Loader using the mod_cdr generated log file.

Moving usage information. If a customer wants to move usage data from one BRM installation to another and the usage information from the original site came through a RADIUS system using mod_cdr, then the usage information can be imported using the log file generated by mod_cdr.

Checklist of Customization Considerations for mod_cdr

The mod_cdr module offers you great flexibility in configuring log files. Because of this, it’s important to consider your overall system needs before beginning the configuration process. Use the following checklist when evaluating your unique system requirements.

Destination of the Log File

- Where do I want the log files written?
- Will other applications, scripts, or cron jobs need to read the log files? Does this effect where the log files are archived?

How Log Files Will Be Used

- Who or what applications are consumers of the log files?
- Do I need to match the configuration of my log files with another application such as UE Loader?
- What is the standard operating procedure for moving mod_cdr output to any consumer applications?
- Will a consumer application need to read the _CURRENT file while it’s being written?

Number and Size of Log Files

- Do I want to set a maximum number of records per log file?
- Do any consumer applications I plan to use have a limitation on the numbers of records per log file and does this match with my configuration for mod_cdr?

Rotation Schemes

- Do I want to periodically rotate log files?
- Do I want to implement a time-based rotation scheme (such as hours, days, months)?
- How does the time-based rotation scheme I’ve chosen impact the size and number of log files generated?
- If I choose to rotate by a maximum number of records, do I want to continue incrementing to a large number or begin overwriting at a specified number of records?

Filtering

- Do I want to separate records by NAS?
- Do I want to filter RADIUS messages by type (such as Stop and Start requests)?
- Do I need to log different types of RADIUS messages to different files?
Configuring mod_cdr

Use these instructions to create a custom configuration of mod_cdr in the RADIUS configuration file. The mandatory module type setting is mod_cdr.

Logging Different Types of RADIUS Messages

If you wish to log only specific types of RADIUS messages, set the check type field. This example shows the format of the check entry.

```plaintext
check {
  Acct-Status-Type=Stop
  Pseudo-Request-Type=Accounting-Request
}
```

Specifying the Attribute Order

The attribute order defines how attributes appear in the resulting log files. List the RADIUS attributes in the order they should be logged. If an entry in the attribute order list is not a RADIUS attribute, then a warning message is logged to the appropriate log file.

```plaintext
attribute-order {
  # attribute-order {
  User-Name
  User-Password
  "This is an example of a string literal"
  NAS-Identifier
  NAS-Port
  Pseudo-Request-Source
  Pseudo-Request-Type
}
```

Defining Field and Record Delimiters

Delimiter values separate fields, records, and string literals. Only single character delimiters are supported. If the log files are to be processed by another application, such as UE Loader, then these must match the special delimiters required by that application.

```
Note: Take care when entering special characters such as the end of line. Precede the end of line with a \\.
```

```plaintext
field-delimiter=" "
record-delimiter="\n"
```

Specifying Delimiters for String Literals

Any entries that are not in the RADIUS dictionary are treated as string literals. If you wish to use a string literal which is the same as a dictionary entry, enclose the attribute in escape character delimited quotes. If you include string literals, then define the start and end delimiters for string literals.

```plaintext
string-literal-start-delimiter="[
string-literal-end-delimiter="]"
```
Defining Blank Fields

If you wish to add special characters for blank fields then, mod_cdr will replace the blank field with a blank-field entry. If not specified, then the blank-field is left empty.

blank-field="-"

Including Special Characters

If you wish to include special characters set insert-escape-character entry. This field can be set to true or false. When set to true, fields with a special delimiter character in them get the escape character "\" inserted in front of them. The default is false.

insert-escape-char=true

Using cleartext or Encoded Password Entries

Use the password-cleartext field to define whether you want password fields to be encoded or not. If set to true, then password fields are put in the log file as a clear text password. If set to false, then the password field is encoded. The default is true.

password-cleartext=true

Pre-appending the RADIUS Request Type

You can pre-append the RADIUS request type by setting the prepend-type entry to true. This is useful when you need to analyze or sort log files by types of RADIUS request.

This field can be set to true or false and maps two types of RADIUS packets, Pseudo_Request and Account_Status_Type into one easy to read field. The default is false. If set to true, the record will have the RADIUS type appended in front of the record. Values can include:

- ACCESS_REQUEST
- START_ACCOUNTING_REQUEST
- STOP_ACCOUNTING_REQUEST
- ACCOUNTING_ON_REQUEST
- ACCOUNTING_OFF_REQUEST
- ACCOUNTING_REQUEST_OTHER, OTHER

Setting Limits on the Number of Records and Log Files

In the rotate-log block set the entries for max-record and max-increment values.

- The max-record field defines the maximum number of records to be included in a log file.
- The max-increment field defines the maximum number of records defined for the numeric counter (%N value). When the number of log files (%N value) exceeds the max-increment value, then the counter starts back at 1.

---

Note: If either max-record or max-increment is set to -1, then the feature is disabled.

---

rotate-log {
Configuring the Destination for Log Files

The `dest` entry is used to define the file name and optionally the path name of the destination file. The `dest` entry is highly configurable and is core to the operation of `mod_cdr`. In its simplest configuration, the `dest` entry requires only the file name of the destination file. However, by combining the options listed below, this entry can be used to define a full path and filename which specifies the numeric counter, the NAS option, and the rotation scheme for log files.

**Default Configuration for the Destination File**
The default for the `dest` field is:

```
dest=logs/cdr_log.%N
```

This default setting archives log files to the `log` directory in a file named `cdr_log`. A numeric counter indicates the number of the log file. The resulting file will be named `call_cdr.log.1`. On the next rotation, the file will be named `call_cdr.log.2` and so on.

**Basic Customization of the Dest Field**
When customizing the `dest` field, the only required entry is the file name for the destination file.

```
dest=ThisIsAFileName
```

The full path name can also be added.

```
dest=logdirectory/ThisIsAFileName
```

**Adding a Numeric Counter to the dest Field**
A numeric counter can be added to the destination file name. The `%N` option is a numeric counter designating the log file rotation number. The `%N` option may appear anywhere within the `dest` field.

```
dest=logdirectory/ThisIsAFileName%N
```

**Adding the NAS Option to the dest Field**
If messages from different NAS need to be recorded in separate log files then use the `%n` option. When the `%n` option is used the NAS identifier gets substituted as part of the destination file name.

```
dest=logdirectory/ThisIsAFileName%n
```

**Adding Time-Based Rotation Schemes to the dest Field**
Log files may be rotated based on time-based rotation schemes. Table 7-1 describes the options available.

```
max-record=100000
max-increment-value=10
}
```
Configuring mod_cdr

By using the time based rotation options listed above, you can configure mod_cdr to rotate log files according to prescribed time periods such as minutes, hours, days, months, etc.

In the example below the log files will be stored in the /var/portal/7.4/radius directory. The destination log file name includes the NAS option, a numeric counter, and the log files will be rotated each day of the year (using the %j option.)

dest="/var/portal/7.4/radius/cdr/call_cdr_%n.%j.%N.log"

Table 7–1 Options for Time-Based Rotation Schemes

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Abbreviated weekday name</td>
</tr>
<tr>
<td>%A</td>
<td>Full weekday name</td>
</tr>
<tr>
<td>%b</td>
<td>Abbreviated month name</td>
</tr>
<tr>
<td>%B</td>
<td>Full month name</td>
</tr>
<tr>
<td>%c</td>
<td>Local date and time representation</td>
</tr>
<tr>
<td>%d</td>
<td>Day of the month</td>
</tr>
<tr>
<td>%H</td>
<td>Hour - 24 hours clock (00-23)</td>
</tr>
<tr>
<td>%I</td>
<td>Hour - 12 hour clock (01-12)</td>
</tr>
<tr>
<td>%j</td>
<td>Day of the year (001-366)</td>
</tr>
<tr>
<td>%m</td>
<td>Month (01-12)</td>
</tr>
<tr>
<td>%M</td>
<td>Minute (00-59)</td>
</tr>
<tr>
<td>%p</td>
<td>Local equivalent of AM or PM</td>
</tr>
<tr>
<td>%S</td>
<td>Second (00-59)</td>
</tr>
<tr>
<td>%U</td>
<td>Week number of the year (00-51) Sunday as first day of week</td>
</tr>
<tr>
<td>%w</td>
<td>Weekday (0-6) Sunday is 0</td>
</tr>
<tr>
<td>%W</td>
<td>Week number of the year (00-51) Sunday as first day of week</td>
</tr>
<tr>
<td>%x</td>
<td>Local date presentation</td>
</tr>
<tr>
<td>%X</td>
<td>Local time presentation</td>
</tr>
<tr>
<td>%y</td>
<td>Year without century (00-99)</td>
</tr>
<tr>
<td>%Y</td>
<td>Year with century</td>
</tr>
</tbody>
</table>

By using the time based rotation options listed above, you can configure mod_cdr to rotate log files according to prescribed time periods such minutes, hours, days, months, etc.

In the example below the log files will be stored in the /var/portal/7.4/radius directory. The destination log file name includes the NAS option, a numeric counter, and the log files will be rotated each day of the year (using the %j option.)

dest="/var/portal/7.4/radius/cdr/call_cdr_%n.%j.%N.log"

Guidelines for Setting Up Rotation Schemes

When setting up rotation schemes use these general guidelines.

- If the dest string does not have a %N option or time stamp pattern, then the log file never gets rotated.
- If the dest string does not have %N option but does includes a time stamp, then the file is rotated when the time pattern is changed.
- If the dest string has a %N option, but does not have a time stamp, then the file is rotated when the maximum number of records (max_record) is exceeded.
- If the dest string has both a %N option and a time stamp, then the file gets rotated when the maximum number of records (max_record) is reached or when the time pattern changes, whichever changes first.
Managing Current Files

The current file is the log file which is currently being written. It is designated with a special suffix _CURRENT.

<%N>.<Module_Name>._CURRENT

- `<%N>` is the current number of rotation file
- `<Module_Name>` is the module name.
- `__CURRENT` indicates that this is the current log file.

After the current log file is complete, it is renamed and archived. A new log file is created and given the special _CURRENT suffix. Once the log file has been renamed and archived, you can copy or move the file to another location or process it. It’s not recommended that you move the _CURRENT file unless you are sure that the file won’t grow anymore. If the __CURRENT CDR file does not exist, then mod_cdr won’t be able to recreate previously set values such as %N across RADIUS server invocations.

**Note:** Changing the dest pattern during a RADIUS invocation may create an orphan __CURRENT file. If this happens, be sure to clear the orphan __CURRENT file.

Managing Archive and Backup Files

After the current (_CURRENT) file has been rotated out, it becomes an archive file. It is moved into the cdr_archive directory and renamed using a special time-based suffix.

For example:

dest="/var/someDir/%n/CDR.%d"

If the value of %d (designating the day of the month) is 05, and the NAS is 150.151.152.153, then, before mod_cdr writes to a file, it checks to see whether a file with this name already exists:

/var/someDir/150.151.152.153/cdr_archive/cdr.05.Jan-05-1999.10.05

- If an archive file by this name does not exist, then mod_cdr creates the file and writes the data to this file.
- If an archive file by this name already exists, then the existing archive file is copied to the backup directory (cdr_backup) and renamed. The existing archive file is then overwritten.
- If a backup archive file by this name already exists in the cdr_backup directory, then that backup file is overwritten.
Transforming Attribute Values

This chapter explains how to transform string type attribute values using the mod_transform module. All request types are accepted by the mod_transform module.

**Important:** Only attributes of RADIUS type string can be transformed. Transformation of other attribute types is not supported.

How the Transformation Module Works

It is sometimes useful to perform operations on a RADIUS packet without having to write a custom module. For example, you may wish to strip out leading or trailing characters, add the domain name to a user name, or copy an attribute value from one attribute to another.

Use the mod_transform module to perform these types of transformations. Multiple transformations can be performed on one or more attributes in the packet. When a transformation is completed, the packet along with the transformed attributes are passed to the next module worker class in the chain.

Configuring the Transformation Module

No global configuration options are supported for this module. Use the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config, where BRM_home is the directory in which BRM components are installed) for all module specific configurations. The mandatory `type` element for this module is `mod_transform`.

Matching Attribute-Value Pairs in a Request

Use the `check` element to match attribute-value pairs in a request:

- If the attribute-value pair in the request matches the criteria set in the `check` element, the request is processed.
- If any attribute in the `check` element is not matched in the request, the `check` fails. After the `check` fails, the request is passed on to the next module worker class in the chain.

The `send` element does not apply to this module.

Transforming Attributes

In the `transform` block, define the list of transform operations to be performed on an attribute. Transformations are carried out in the order specified. Use the `transform`
element to define a module specific transformation. Within a transform block, the operations are carried out in the order specified. The syntax is:

\[ \text{Attribute name} = \text{transform function} \]

These transformation functions are supported.

- Search and replace.
  Syntax: \texttt{Replace/target	extunderscore string/replace	extunderscore string/}
- Change case.
  Syntax: \texttt{ChangeCaseUpper | ChangeCaseLower}
- Strip leading space.
  Syntax: \texttt{StripLeading}
- Strip trailing space.
  Syntax: \texttt{StripTrailing}
- Add an attribute.
  Syntax: \texttt{add}
- Copy an attribute.
  Syntax: \texttt{copy}

---

**Note:** Syntax for these transformation functions is case sensitive.

Example:

This example shows the syntax for each transformation function as it appears in the configuration file. The next example shows the results.

```
transform1 {
    type = mod_transform
    transform {
        User-Name = StripLeading
    }
}
transform2 {
    type = mod_transform
    transform {
        User-Name = StripTrailing
    }
}
transform3 {
    type = mod_transform
    transform {
        User-Name = ChangeCaseUpper
    }
}
transform4 {
    type = mod_transform
    transform {
        User-Name = ChangeCaseLower
    }
}
transform5 {
```
Configuring the Transformation Module

Transforming Attribute Values

```plaintext
type = mod_transform
transform {
    User-Name = Replace/$/@portal.com/
}
}
transform6 {
    type = mod_transform
transform {
        User-Name = Replace/^/user_/
    }
}
transform7 {
    type = mod_transform
transform {
        User-Name = Replace/example//
    }
}
transform8 {
    type = mod_transform
    add {
        Class = "MyClass"
    }
}
transform9 {
    type = mod_transform
    copy {
        Reply-Message = Class
    }
}

Results:

THREAD: [1] got request:
User-Name = * bbexample *

[transform1]: Resulting packet after transform:
User-Name = bbexample

[transform2]: Resulting packet after transform:
User-Name = bbexample

[transform3]: Resulting packet after transform:
User-Name = bbexample

[transform4]: Resulting packet after transform:
User-Name = bbexample

[transform5]: Resulting packet after transform:
User-Name = bbexample@portal.com

[transform6]: Resulting packet after transform:
User-Name = user_bbexample@portal.com

[transform7]: Resulting packet after transform:
User-Name = user_bb@portal.com

[transform8]: Resulting packet after transform:
User-Name = user_bb@portal.com
Class = MyClass

[transform9]: Resulting packet after transform:
```

Appending Values to Attributes

Use the add element to append an attribute value to the incoming request. The syntax is:

```
Attribute_name = attribute_value
```

Copying Attribute Values

Use the copy element to copy the value of attribute_2 to attribute_1.

- If the attribute_2 doesn’t exist in the packet, the operation is not performed.
- If attribute_1 is not present, an attribute with the value of attribute_2 is created and appended to the packet, else the value of attribute_1 is changed.

The syntax is:

```
attribute_1 = attribute_2
```

Example Using the Transformation Module

This example shows how the user name attribute is transformed by stripping out the domain name, appending isp.com, and prepending the "user" to the user name. A new attribute, the calling station id, is then added and copied to the user name attribute.

Example:

```
# mod_transform to strip leading and trailing space, change to uppercase
# From the User-Name attribute, removes domain "domain.com", appends isp.com and prepends "user" to username.
# from all packets from 164.123.10.1
mod_transform {
    check {
        Pseudo-Request-Source = 164.123.10.1
        User-Name = "**@domain.com"
    }

    add {
        Calling-Station-Id = "408-111-2222"
    }

    copy {
        User-Name = Calling-Station-Id
    }

    transform {
        User-Name = StripLeading
        User-Name = StripTrailing
        User-Name = ChangeCaseUpper
        User-Name = Replace/"@domain.com"/
        User-Name = Replace/"user_"/
        User-Name = Replace/$/@isp.com/"
    }
}
```
Managing Requests for Wholesale Accounts

This chapter describes how to use the Wholesale Module to manage requests for wholesale accounts.

---

**Caution:** Only administrators with advanced RADIUS skills should attempt to implement custom configurations.

---

Before reading this chapter you should be familiar with:

- Understanding RADIUS Manager Modules
- Configuring RADIUS Manager

### About the Wholesale Module

The wholesale module can improve the performance of request processing for large wholesale accounts by dividing up account requests and sending them to subordinate accounts for processing. The wholesale module defines a wholesale account as a realm.

### Specifying the Realm Attributes and Options

To configure the wholesale module, specify realm information in the **Per Module** section of the RADIUS config file.

- To specify the name of the realm, use the **Name** entry.
- To specify the attribute containing the realms, use the **Attribute** entry.
- To specify the pattern for extracting the realm name from the attribute, use the **Pattern** entry.

**Example**

```plaintext
mod_wholesale {
  RealmAttributeOptions {
    name = realm1
    attribute = Class
    pattern = "WC=*@/Realm/@*"
  }
  RealmInfo {
    realm-name = "ISP1"
    name-range = 500
    name-pattern = "ISP1%d"
  }
}
```

Caution: Only administrators with advanced RADIUS skills should attempt to implement custom configurations.
If we use this RealmAttributeOptions then an incoming request having the attribute Class=WC=*@ISP1@* will be routed to one the accounts ranging from ISP1 to ISP499.

In the MODULES section of the config file use the `wholesale` block to specify the realm attribute options.

```plaintext
wholesale {
    status = disabled
    type = mod_wholesale

    use-realm-attr-options = realm3

    add {
        AAA = User-Name
        Custom-Realm = %r
        Custom-User-Name = %u
        Custom-Type = "ISPXYZ"
        Custom-Literal = "User:%u,Realm:%r"
    }
    options = Ignore-No-Entry
}
```

Table 9–1 lists the descriptions for example realm attribute options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA = User-Name</td>
<td>Copy one attribute to another. For example, copy the user_name to AAA.</td>
</tr>
<tr>
<td>Custom-Realm = %r</td>
<td>Store realm name using %r.</td>
</tr>
<tr>
<td>Custom-User-Name = %u</td>
<td>Store the original user name using %u.</td>
</tr>
<tr>
<td>Custom-Type = &quot;ISPXYZ&quot;</td>
<td>Store a literal string into an attribute.</td>
</tr>
<tr>
<td>Custom-Literal = &quot;User:%u,Realm:%r&quot;</td>
<td>Combination of the above options</td>
</tr>
</tbody>
</table>
Creating Custom Modules

This chapter describes how to create your own RADIUS module. Extending the RADIUS server is a task for advanced users with special requirements. Creating new modules requires C or C++ programming skills and an understanding of the RADIUS protocol and thread programming.

**Caution:** Only administrators with advanced RADIUS skills should attempt to implement custom configurations.

Before creating a custom module you should be familiar with:

- Understanding RADIUS Manager Modules
- Configuring RADIUS Manager
- Using the Authentication and Authorization Modules

**About Creating Custom Modules**

Before creating your own module, be sure that the task you wish to accomplish cannot be performed by one of the modules supplied with the RADIUS Manager.

You may write a custom module which authenticates and modifies incoming packets to handle special conditions. You also might want to modify logins to provide greater flexibility in managing requests. With custom modules, you can:

- Add or remove part of a login.
- Authenticate a request against an alternate database.
- Handle requests with identical logins.

Use `mod_example` as a template for your new module. Customize the required sections, and link your custom modules with the core functionality module libraries to produce new RADIUS server binaries.

**Note:** The RADIUS manager framework does not expose the connection pool APIs to the custom RADIUS modules. Any custom RADIUS module that calls an opcode by calling the Connection Manager (CM) needs to manage its own connection pool.
Checklist for Creating Custom Modules

This checklist provides an overview of tasks that must be performed when you create a custom module. To create a custom module:

1. Review the Module Class Model as implemented by RADIUS Manager. See "About the Module Class Model".

2. After RADIUS Manager is installed, locate the mod_example template in BRM_home/source/apps/radius, where BRM_home is the directory in which BRM components are installed.

3. Copy the mod_example.cpp and mod_example.h files. Store the copies in the BRM_home/source/apps/radius directory.

4. Add the new module to the definitions file (BRM_home/source/apps/radius/moddef.cpp) file.

5. Add functionality to the new module, by modifying the module.h and module.cpp files.

6. Add an object for the new module to the Makefile:

   OBJS = moddef.o <mod_new>.o

7. Build your custom module and link it with the libraries to produce a new radiusid executable:

   make -f makefile

Note: When you run the makefile, the following warning message displays. You can ignore the warning.

   Warning 921: "/usr/include/stdlib.h", line 606 # A trailing comma is not allowed in an enumeration.

   AR_ARENA_OPEN=0, /* open a session for obtaining memo

8. Stop the RADIUS server, if it is running.

9. Copy pin_radiusid from the current directory to BRM_home/bin.

10. Edit the RADIUS configuration file (BRM_home/apps/radius/config) to add the new module to the module chain.

11. Start the RADIUS server.

12. Test the new module.

Modifying the Definitions File

Modify the definitions file (BRM_home/source/apps/radius/moddef.cpp) file by declaring the module master creation function and adding the name of your module to the module definition table.

Add this line:

   {<mod_examplename>, MDF_NONE, RadiusModuleNew_create}

To this section of the moddef.cpp file:

   . . .

   extern RadiusModule *RadiusModuleTransform_create(string theName, string
Adding Functionality to a Custom Module

To add functionality to a custom module you must understand the C++ API interface and the Module Class Module.

About the C++ API Interface to RADIUS Modules

To create a new module type, declare new C++ classes which inherit from a set of base classes, and then implement the module type-specific functionality. The virtual base class for module masters is RadiusModule. See modbase.h for detailed information about the RadiusModule.

The module masters are instantiated by a special module definition table. See moddef.cpp for detailed information about the module definition table. The API support files contain specific information about using each file.

Support Header Files

The support header files (module.h) are self-documenting. See the following files for details:

In source/apps/radius:
- moddef.h

In source/apps/radius/include/general:
- applog.h - Application log functions
- debuglog.h - Debug log functions
- ipconvert.h - IP address translation utilities
- list.h - List processing functions
- lstring.h - String class
- strutil.h - String processing functions
- usersconfig.h - How to handle configuration files

In source/apps/radius/include/modules:
- checksend.h - Check and send support for modules
- modbase.h - Definitions of module classes
About the Module Class Model

Before writing a custom module, you must understand the Module Class Model and the support APIs.

This section describes the C++ API interface to module masters and module workers. A virtual base class inheritance model is used. This means that creating a new module type involves declaring new C++ classes which inherit from a set of base classes and then implement the module type’s specific functionality.

About Configuring Modules

The modules, which are configured in the RADIUS configuration file (BRM_home/apps/radius/config), form an ordered list. When a request is received from the core server, it is processed by each module in turn until one of the modules indicates that it has completely processed the request. Note that the behavior of a module is determined by its configuration, especially its type.

Each module indicates what should happen next by setting a "return value":

- MRT_CONTINUE indicates that the request should be passed to the next module.
- MRT_COMPLETE indicates that the request has been successfully processed. The response has been filled in by the module and should be sent to the requesting client (NAS).
- MRT_ERROR indicates that an error has occurred and the request should be discarded.
- MRT_DISCARD indicates that the request should be discarded. When an individual module receives a request, it performs the following tasks:
  - Checks to see that the request matches its check sections. If not, it returns MRT_CONTINUE.
  - Optionally makes changes to the incoming request by adding, deleting, or modifying attributes. (See mod_transform for an example).
  - Optionally makes changes to the outgoing response by setting the response type, and adding, deleting, or modifying attributes.
  - Optionally adding attributes from any send sections to the outgoing response.
  - Returns MRT_CONTINUE, MRT_COMPLETE or MRT_ERROR as appropriate.
If the end of the module list is reached and no module has returned MRT_COMPLETE, the request is discarded.

**Note:** Configure the mod_pin module after configuring all the other modules because mod_pin prepares the RADIUS response and sends it to the client.

### About the Module Master

The virtual base class for module masters is RadiusModule. The salient parts are shown here from modbase.h. See modbase.h for detailed information.

```cpp
class RadiusModule {
    protected:
        ModuleDefFlags flags;
        string name;
        string type;
        int ref_count;

        pthread_mutex_t mutex;
        ConfigEntry *shared_config; // A copy of $CONFIG->(name)
        ConfigEntry *worker_config; // A copy of $MODULES->(name)

    /*
    ** use()
    **
    ** Increments the reference count.
    */
    void use();

    public:
    /*
    ** RadiusModule()
    **
    ** Create a module master.
    ** The default operation initialises name, type and flags from
    ** the parameters and sets the reference count to 1.
    */
    RadiusModule(string name, string type, ModuleDefFlags flags);

    /*
    ** ~RadiusModule()
    **
    ** Destroy the module master.
    ** Automatically called when the reference count reaches 0 (see use(),
    ** unuse())
    */
    virtual ~RadiusModule();

    /*
    ** unuse()
    **
    ** Decrement the reference count and if it reaches 0, destroys the object.
    */
    void unuse();

    /*
    ** newConfig()
    **
    */
}
```
** Called when the configuration manager detects that
** new configuration information is available.
**
** Default implementation searches for the entry $MODULES->(name),
** where (name) is the name of this module entry, and stores
** a copy of that part of the configuration tree in 'worker_config'.
** It also searches for $CONFIG->(name) and stores the result (if found)
** in shared_config.
*/
virtual void newConfig(const ConfigEntry *top_config);

/*
** createWorker()
**
** User-defined function which creates a module worker.
** The worker object should take a COPY of its configuration,
** since during a reconfig, the worker may need to complete the
** current request before it dies.
**
** Must call use() to increment the reference count on this object.
*/
virtual RadiusModuleWorker *createWorker(int theThreadId) = 0;

/*
** lock()
** unlock()
**
** Synchronise access to this object.
*/
void lock();
void unlock();
};

** Methods Defined in Derived Classes for the Master Module**

These methods may be defined in the derived class. See the source for mod_example
as an example.

**Constructor**

You must provide a constructor in order to initialize the module master. The base class
constructor must be called.

Example:

```cpp
RadiusModuleExample::RadiusModuleExample(string name_, string type_,
ModuleDefFlags flags_)
: RadiusModule(name_, type_, flags_)
{}
```

**Destructor**

Destroys the object. Generally doesn't need to do anything.

Example:

```cpp
RadiusModuleExample::~RadiusModuleExample()
{}
```
newConfig()

This method is called both at startup and when a reconfiguration event occurs. It is the responsibility of the module master to extract the appropriate configuration from the configuration tree. The default implementation keeps a reference to both the module type configuration (shared_config) and the per-module configuration (worker_config). Thus, most modules do not require a special implementation of this method.

createWorker()

This method creates a new module worker and increments the reference count. Typically, this method simply creates a new module worker of the appropriate type. The module worker is responsible for taking a copy of both the per-module configuration (worker_config) and the module type configuration (shared_config), when appropriate. It must also call use() to ensure that the reference count is incremented.

Example:

```c
RadiusModuleWorker *
RadiusModuleExample::createWorker(int thread_id)
{
    assert(worker_config != 0);
    use();
    return(new RadiusModuleExampleWorker(this, worker_config,
        shared_config, thread_id));
}
```

About Worker Modules

The module worker is where most of the work of the module is accomplished. The virtual base class for module workers is RadiusModuleWorker. The salient parts are shown here from modbase.h. See modbase.h for detailed information.

Methods Defined in Derived Classes for Worker Modules

This section describes the APIs to module workers.

Constructor (required)

A constructor is required in order to initialize the module worker information. The base class constructor must be called to initialize the parent pointer and to copy the worker_config. Any configuration information that will be referenced must be copied. Specific configuration can also be extracted from the worker_config, the shared_config, or both. A CheckSend object is normally created here. Also, any standard options should be parsed here.

Example:

```c
RadiusModuleExampleWorker::RadiusModuleExampleWorker(RadiusModule *parent_,
    const ConfigEntry *worker_config,
    const ConfigEntry *shared_config,
    int thread_id)
    : RadiusModuleWorker(parent_, worker_config, thread_id)
{
    checksend = new RadiusCheckSend(&config);
    action = config.getValue(MOD_EXAMPLE_ACTION);
    if (action == "") {
        APP_LOG("[W]%s: No action specified in config. Using action=ignore. ",
            (const char *)&getName());
        action = MOD_EXAMPLE_ACTION_IGNORE;
    }
```
/* No shared_config for this module type */
}

Destructor
Destroys the object.
Example:

RadiusModuleExampleWorker::RadiusModuleExampleWorker()
{
    delete checksend;
}

acceptRequest()
Processes the given request. The method:

- May modify request->input if appropriate, by adding, modifying, or deleting attributes.
- May modify request->output if appropriate, by setting the type, or adding, modifying, or deleting attributes.
- Should call checksend->check() as the first thing, if appropriate, which should almost always be the case.
- Should call checksend->addSendAttr() as the last thing when sending a response.

Returns one of MRT_..., such as:

RadiusModuleWorker::ModuleReturnType
RadiusModuleExampleWorker::acceptRequest(RadiusModuleRequest *request)
{
    // Do standard check processing
    if (checksend->check(request->input, getSystemDict()) == 0) {
        return(MRT_CONTINUE);
    }
    DEBUG_LOG(("%s: check succeeded", (const char *)getName()));

    if (action == MOD_EXAMPLE_ACTION_DISCARD) {
        return(MRT_DISCARD);
    }

    if (action == MOD_EXAMPLE_ACTION_IGNORE) {
        return(MRT_CONTINUE);
    }

    if (action == MOD_EXAMPLE_ACTION_NAK) {
        switch (request->input->getType()) {
        case PW_ACCESS_REQUEST:
            request->output->setType(PW_ACCESS_REJECT);
            break;
        case PW_ACCOUNTING_REQUEST:
            request->output->setType(PW_ACCOUNTING_RESPONSE);
            break;
        default:
            return(MRT_DISCARD);
        }
    }
    }

    else if (action == MOD_EXAMPLE_ACTION_ACK) {
        switch (request->input->getType()) {
About the Module Class Model

Creating Custom Modules

The sections above describe the APIs to module masters and module workers; however, they don’t explain how module masters are instantiated. This is managed by a special module definition table. The excerpt below is from moddef.cpp:

```cpp
extern RadiusModule *RadiusModuleTransform_create(string theName, string theType,
                                                   ModuleDefFlags theFlags);

const ModuleConfigType module_config[] = {
    { "mod_null", MDF_NONE,  RadiusModuleNull_create },
    //{ "mod_unixpwd", MDF_NONE,  RadiusModuleUnixpwd_create },
    //{ "mod_ipass", MDF_NONE,  RadiusModuleIPass_create },
    { "mod_proxy", MDF_NONE,  RadiusModuleProxy_create },
    { "mod_text", MDF_NONE,  RadiusModuleText_create },
    { "mod_logging", MDF_NONE,  RadiusModuleLogging_create },
    { "mod_transform", MDF_NONE,  RadiusModuleTransform_create },
    { 0, MDF_NONE, 0 }
    { "mod_example", MDF_NONE,  RadiusModuleExample_create },
    { "mod_pin", MDF_NONE,  RadiusModulePin_create },
};
```

This table associates Module Type names with functions that know how to create a module master of the associated class. You can modify this table to add custom modules.

Example of a module master creation function:

```
Note: Configure the mod_pin module after configuring all the other modules because mod_pin prepares the RADIUS response and sends it to the client.
```

Instantiating Module Masters

The sections above describe the APIs to module masters and module workers; however, they don’t explain how module masters are instantiated. This is managed by a special module definition table. The excerpt below is from moddef.cpp:

```cpp
extern RadiusModule *RadiusModuleTransform_create(string theName, string theType,
                                                   ModuleDefFlags theFlags);

const ModuleConfigType module_config[] = {
    { "mod_null", MDF_NONE,  RadiusModuleNull_create },
    //{ "mod_unixpwd", MDF_NONE,  RadiusModuleUnixpwd_create },
    //{ "mod_ipass", MDF_NONE,  RadiusModuleIPass_create },
    { "mod_proxy", MDF_NONE,  RadiusModuleProxy_create },
    { "mod_text", MDF_NONE,  RadiusModuleText_create },
    { "mod_logging", MDF_NONE,  RadiusModuleLogging_create },
    { "mod_transform", MDF_NONE,  RadiusModuleTransform_create },
    { 0, MDF_NONE, 0 }
    { "mod_example", MDF_NONE,  RadiusModuleExample_create },
    { "mod_pin", MDF_NONE,  RadiusModulePin_create },
};
```

Note: Configure the mod_pin module after configuring all the other modules because mod_pin prepares the RADIUS response and sends it to the client.

This table associates Module Type names with functions that know how to create a module master of the associated class. You can modify this table to add custom modules.

Example of a module master creation function:
Adding a New Module to the RADIUS Configuration File

Edit the RADIUS configuration file (`BRM_home/apps/radius/config`) to add the new module to the module chain.

Sample Code for a Custom Module

This sample prints the name of the incoming RADIUS attributes and their values and appends the domain name to the user name attribute.

```c
const RadiusAttr *theAttr = NULL;
RadiusAttr *newAttr = NULL;
/* This code prints the names of the incoming RADIUS attributes and their values */
DEBUG_LOG(("printing all attributes in the packet"));
while (((theAttr = request->input->getEntry ( theAttr )) != 0 )) {
    string name = theAttr->printName(getSystemDict());
    string value = theAttr->printValue(getSystemDict());
    DEBUG_LOG(("attribute name = %s", (const char *)name));
    DEBUG_LOG(("attribute value = %s", (const char *)value));
    if (theAttr->getCode() <= PW_LAST_VALID_ATTR_CODE) {
        newAttr = new RadiusAttr (theAttr->getCode(),
            theAttr->getBuffer(),
            theAttr->getBufferLength());
        request->output->addAttr ((RadiusAttr *)newAttr);
    }
}
/* the following piece of code will append a domain name to the user-name attribute i.e if the username is joe, this code will change it to joe@myisp.com, the 'add_domain' keywords must be defined in your .h file and specified in the config file. */
DEBUG_LOG(("appending domain name to the user-name attribute"));
theAttr = request->input->getEntry ( PW_USER_NAME, NULL);
string value = theAttr->printValue(getSystemDict());
if (add_domain !=") {
    DEBUG_LOG(("adding domain %s to username %s", add_domain.PeekString(),
        value.PeekString()));
    value += add_domain;
    RadiusAttr *newAttr = new RadiusAttr (PW_USER_NAME, value);
    const RadiusAttr *modAttr = request->input->modifyAttr (theAttr, newAttr);
    RadiusAttr *outAttr = new RadiusAttr (modAttr->getCode(), modAttr->getBuffer(),
        modAttr->getBufferLength());
    request->output->addAttr ((RadiusAttr *)outAttr);
}
```

Adding a New Module to the RADIUS Configuration File

Edit the RADIUS configuration file (`BRM_home/apps/radius/config`) to add the new module to the module chain.
Example:

type=<mod_new>
status=enable
<check>
<module specific actions>
<send>
<etc.)

Starting and Stopping the RADIUS Daemon

When you finish modifying the RADIUS configuration file you must restart the RADIUS daemon.

Use this procedure:

1. Check to ensure that the RADIUS config file and dictionary file are in the directory
   BRM_home/apps/radius.

2. Run either the start or stop script, BRM_home/bin/start_radius or BRM_home/bin/stop_radius. These scripts can be run manually but you should make them part of the software initialization at startup time.

3. If you want pin_radiusd to start automatically when you restart the machine.

4. Run the BRM_home/bin/install_radius script after you install the software. The install_radius script puts the required entries in the /etc/rc2.d directory to start pin_radiusd automatically.
Managing Large Volume Uploads and Downloads

This chapter describes how to use and configure the `mod_unit` module to manage upload and download volumes greater than 2 GB.

**How mod_unit Module Works**

To comply with the NAS requirement to support upload and download volumes greater than 2 GB, Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager supports unsigned integer data type. However, because the Dialup event structure supports signed integer data type, when an opcode call is made from the RADIUS modules for values greater than 2 GB, proper scaling needs to be done by the RADIUS Manager before calling the opcode. For this reason, the `mod_unit` module is used to scale unsigned integer values to signed integer values so that the values can be stored using the Dialup event structure.

**Note:** You don’t need to use `mod_unit` if you don’t have upload and download volumes greater than 2 GB.

**Setting the Scaling Unit for mod_unit**

You set the scaling unit for `mod_unit` in the RADIUS configuration file to be byte (1), kilobyte (2), megabyte (3), or gigabyte (4). For instance, to rate incoming packets by kilobyte, you set the `scale-unit` configuration parameter to 2.

The `mod_unit` module reads the Acct-Input-Octets and Acct-Input-Gigawords and, depending on the scaling unit specified, calculates the new values for Acct-Input-Octets and Acct-Output-Octets as follows:

- **When scale-unit is set to 1:**
  - \[\text{Acct\_Input\_Octets} = \text{Acct\_Input\_Gigaword} \times 4 \times 1024 \times 1024 + \text{Acct\_Input\_Octets}\]
  - \[\text{Acct\_Output\_Octets} = \text{Acct\_Output\_Gigaword} \times 4 \times 1024 \times 1024 + \text{Acct\_Output\_Octets}\]
- **When scale-unit is set to 2:**
  - \[\text{Acct\_Input\_Octets} = \text{Acct\_Input\_Gigaword} \times 4 \times 1024 + \text{Acct\_Input\_Octets}/1024\]
  - \[\text{Acct\_Output\_Octets} = \text{Acct\_Output\_Gigaword} \times 4 \times 1024 + \text{Acct\_Output\_Octets}/1024\]
■ When `scale-unit` is set to 3:
  - \[\text{Acct\_Input\_Octets} = \text{Acct\_Input\_Gigaword} \times 4 \times 1024 + \frac{\text{Acct\_Input\_Octets}}{1024 \times 1024}\]
  - \[\text{Acct\_Output\_Octets} = \text{Acct\_Output\_Gigaword} \times 4 \times 1024 + \frac{\text{Acct\_Output\_Octets}}{1024 \times 1024}\]

■ When `scale-unit` is set to 4:
  - \[\text{Acct\_Input\_Octets} = \text{Acct\_Input\_Gigaword} \times 4 + \frac{\text{Acct\_Input\_Octets}}{1024 \times 1024 \times 1024}\]
  - \[\text{Acct\_Output\_Octets} = \text{Acct\_Output\_Gigaword} \times 4 + \frac{\text{Acct\_Output\_Octets}}{1024 \times 1024 \times 1024}\]

**Note:** The fields which are modified by `mod_unit` are related to accounting only. After the conversion, the input values are overwritten in Acct-Input-Octets and Acct-Output-Octets with the modified value.

### Changing Rate Plans to Use Scale Unit as the Unit for Rating

When an opcode call is made from the RADIUS modules for values greater than 2 GB, the `mod_unit` module converts the unit from bytes to kilobytes, megabytes, or gigabytes, depending on the scaling unit configuration that you set.

If you set the scaling unit to scale higher, than you must also change your rate plans to rate using the higher unit. For example, if your existing rate plans use bytes as the unit for rating, and `mod_unit` is used to scale the upload and download volumes from bytes to kilobytes, than you must change your rate plans to use kilobytes as the unit for rating.

However, if you have specific requirements for using bytes as the unit for rating, you can also extend the Dialup event to record the upload and download volume information in the decimal field format. Then define custom RUM settings to rate using the volume information stored in the decimal format. In this case, you do not use `mod_unit` for scaling of units.

**Note:** You need to change your rate plans only if the upload and download volumes are more than 2GB, and if `mod_unit` module is used to scale the units or the information is recorded in decimal format.

### Configuring mod_unit

You configure `mod_unit` in the $MODULES section of the RADIUS configuration file (`BRM_home/apps/radius/config`, where `BRM_home` is the directory in which BRM components are installed). Use the `scale-unit` configuration parameter to specify the scaling unit. For example:

```plaintext
exp {
  type=mod_unit
  scale-unit = 3
}
```
Note: You must restart the RADIUS daemon after modifying the configuration file.

For more information, see "Adding a New Module to the RADIUS Configuration File" and "Starting and Stopping the RADIUS Daemon".
Using a Virtual Private Dialup Network (VPDN)

This chapter explains how to use a virtual private dialup network (VPDN) with the Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager. Before using RADIUS Manager to set up a VPDN, you should be familiar with these topics:

- Configuring RADIUS Manager
- Understanding RADIUS Manager Modules
- Using the Authentication and Authorization Modules

About Using a Virtual Private Dialup Network

A virtual private dialup network (VPDN) enables a customer to transparently access a private network by using a public dialup Internet service provider (ISP). VPDN allows a company, that doesn’t want to own POPs (Points of Presence) and network infrastructure to provide dialup access to a company network.

Implementation requirements for VPDN solutions:

- Provide a POP (phone lines, modems, and terminal servers).
- Provide authentication of users.
- Track failed authentication requests for potential security breaches.

If the terminal servers are outside a firewall, you must also provide:

- A mechanism for information to flow through a firewall
- Secure data transmission using encryption

About Firewalls and Tunnels

Typically an organization uses a firewall and provides access from the Internet. A tunnel needs to be created between the user’s computer and the organization’s network. This tunnel transparently transmits information over the Internet and through the firewall. Since the information being transmitted through the tunnel is generally sensitive information, it is usually encrypted.

Tunneling Protocols

There are a number of standard and proprietary tunneling protocols available such as L2TP, L2F, ATM, and PPTP. Each protocol requires different setup information (attributes) to establish the tunnel.
While RADIUS Manager is basically independent of the protocol used between a NAS and the tunnel gateway, only L2F running on CISCO 3600 series hardware has been validated.

Overview of the VPDN Model

Example of how a VPDN user model can work:

1. For a non-VPDN service, the user dials the ISP and logs in as joe or joe@isp.com.
2. If Joe is authenticated successfully against the ISP database, he is allocated an IP address from the ISP’s pool of addresses and given access to the Internet. In this situation, Joe is not be able to access systems and services at company.com, because those systems are behind a firewall.
3. After the appropriate arrangements have been made, Joe is able to dial in to the ISP and get through the firewall.
4. The RADIUS server determines that joe@isp.com requires a VPDN tunnel to be established. The RADIUS server sends attributes back to the NAS which tell the NAS how to set up the tunnel (destination gateway, authentication information, and so on).
5. The NAS establishes the tunnel through to the gateway at company.com and provides Joe with transparent access to the systems and services at company.com.
6. The RADIUS server also accepts accounting information from the NAS. This information is used to bill Joe or company.com for the service.

This solution requires that the employee have only a standard dialup system since tunneling is provided transparently by the ISP between the NAS and the organization’s gateway.

About VPDN Implementation Models

Two VPDN models are supported:

- User model: Users pay for an manage their accounts
- Organization model: The organization controls and pays for accounts

Both models are implemented by configuring the mod_pin module instances in the RADIUS configuration file (BRM_home/apps/radius/config, where BRM_home is the directory in which BRM components are installed). The organization model also requires changes to the default BRM behavior.

Implementing the User Model

With the user model, an organization provides access to users through the organization’s network using a VPDN, but requires users to pay for and manage their own ISP accounts.

User Model Example

1. User Joe dials his local ISP and logs in as joe@isp.com.
2. The NAS sends an Access-Request request to the RADIUS server.
3. The RADIUS service makes two calls to BRM: one to authenticate joe@isp.com, and one to find session creation attributes.
4. The BRM database is searched for the /service/ip storable object containing the login joe@isp.com. This service object defines how to set up the tunnel for the user.

5. Information from the /service/ip storable object defining how to set up the tunnel is returned to the RADIUS server for the reply to the NAS.

6. The NAS uses the tunnel setup attributes to create a tunnel through the appropriate gateway.

7. When this session starts and ends, the NAS sends an Accounting-Request to the RADIUS server. This information is passed to BRM and an /event/session/dialup event is create.

8. The event is then rated and charged to Joe’s account.

Using Multiple Network Access Servers with the User Model

If multiple NAS types are used, they must all be set up to support your tunneling method. It is possible to work around this limitation in the PCM_OP_TERM_POL_AUTHORIZE opcode by determining the type of the NAS and manually setting the appropriate attribute-value pairs for the NAS type.

A limitation of using multiple network access servers is that the default PCM_OP_TERM_POL_AUTHORIZE policy opcode does not provide common configuration information across services.

For example, if joe@isp.com and susan@isp.com both have the same tunneling information, there is no default policy provision for storing that information in a common location and referring to it in the individual services.

Handling VPDN Requests with the User Model

To handle VPDN requests your RADIUS configuration file (BRM_home/apps/radius/config) must be set up to use one of these storable objects:

- /service/vpdn
- /service/ip

These storable objects provide the necessary VPDN tunneling information.

Example Using the /service/vpdn Storable Object

This example shows the syntax in the RADIUS configuration file (BRM_home/apps/radius/config) for using the /service/vpdn storable object with the VPDN user model.

```bash
# Cannot find the account, see if it is a user_vpdn
user_vpdn_auth {
  type = mod_pin
  check {
    Pseudo-Request-Type = Access-Request
    User-Name = "**@**"
  }
  send {
    Class = user_vpdn
  }
  service-type = /service/vpdn
  options = Ignore-No-Entry
}
```
Implementing the User Model

```plaintext
user_vpdn_acct {
  type = mod_pin
  check {
    Pseudo-Request-Type = Accounting-Request
    Class = user_vpdn
  }
  service-type = /service/vpdn
}
```

No `service-type` setting is required to use the `/service/ip` storable object with the user model VPDN. The default BRM authentication and accounting setup processes the requests.

**Example Using the `/service/ip` Storable Object**

This example shows the syntax in the RADIUS configuration file (`BRM_home/apps/radius/config`) for using the `/service/ip` storable object to implement the VPDN user model:

```plaintext
pin_auth {
  type = mod_pin
  check {
    Pseudo-Request-Type = Access-Request
  }
  send {
    Class = "pin"
  }
  options=Ignore-No-Entry
}
pin_acct {
  type = mod_pin
  check {
    Pseudo-Request-Type = Accounting-Request
    Class = "pin"
  }
  options=Ignore-No-Entry
}
```

Both examples require you to configure the PIN_FLD_ARGS array in the appropriate `/service/ip` or `/service/vpdn` storable objects.

**Returning VPDN Attributes to the NAS**

To establish a VPDN tunneling session for an access request, a variety of attributes must be returned to the NAS. The exact attributes needed for each VPDN configuration differs with each VPDN hardware vendor.

To return these attributes to the NAS, the `/service/vpdn` or `/service/ip` storable object must be configured to return the attributes. These attributes are stored in the PIN_FLD_ARGS array in the `/service/vpdn` and `/service/ip` storable objects. All of the required tunneling information is stored in this array. There are two ways to set the attributes in the PIN_FLD_ARGS array:

- Use `testnap` or a custom PCM client.
- Use `PCM_OP_TERM_POL_AUTHRORIZE` to set the attributes during account creation

Using `testnap` or a custom application to set VPDN attributes
The array can be accessed by calling the PCM_OP_WRITE_FLDs opcode from either testnap or a simple custom PCM client.

This example shows how to include the fictitious Custom-Attr-1 attribute in the PIN_FLD_ARGS array:

```plaintext
# Remember to use the PCM_OPFLG_ADD_ENTRY (0x0020) flag when calling the
# opcode if you are adding new array entries.
#
0 PIN_FLD_POID       POID [0] 0.0.0.8 /service/ip 8686
0 PIN_FLD_ARGS      ARRAY [0]
1 PIN_FLD_NAME       STR [0] "Custom-Attr-1"
1 PIN_FLD_VALUE      STR [0] "testing_nas"
```

This example shows the PCM_OP_TERM_POL_AUTHORIZE output flist when the PIN_FLD_ARGS are set:

```plaintext
D Fri October 22 15:19:52 1999 danube pin_radiusd:20320 pin_login.c:498
GOT FROM AUTHORIZE
# number of field entries allocated 13, used 13
0 PIN_FLD_POID       POID [0] 0.0.0.8 /service/ip 8686 5
0 PIN_FLD_ACCOUNT_OBJ POID [0] 0.0.0.8 /account 8942 0
0 PIN_FLD_PROTOCOL    ENUM [0] 1
0 PIN_FLD_COMPRESSION ENUM [0] 0
0 PIN_FLD_ARGS      ARRAY [0] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "User-Service"
  1 PIN_FLD_VALUE STR [0] "2"
0 PIN_FLD_ARGS      ARRAY [1] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Framed-Protocol"
  1 PIN_FLD_VALUE STR [0] "1"
0 PIN_FLD_ARGS      ARRAY [2] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Ascend-Route-IP"
  1 PIN_FLD_VALUE STR [0] "1"
0 PIN_FLD_ARGS      ARRAY [3] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Framed-Netmask"
  1 PIN_FLD_VALUE STR [0] "255.255.255.0"
0 PIN_FLD_ARGS      ARRAY [4] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Ascend-Idle-Limit"
  1 PIN_FLD_VALUE STR [0] "120"
0 PIN_FLD_ARGS      ARRAY [5] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Ascend-Link-Compression"
  1 PIN_FLD_VALUE STR [0] "1"
0 PIN_FLD_ARGS      ARRAY [6] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Ascend-Assign-IP-Pool"
  1 PIN_FLD_VALUE STR [0] "1"
0 PIN_FLD_ARGS      ARRAY [7] allocated 2, used 2
  1 PIN_FLD_NAME  STR [0] "Custom-Attr-1"
  1 PIN_FLD_VALUE STR [0] "testing_nas"
0 PIN_FLD_RESULT    ENUM [0] 1
D Fri May 22 15:19:52 1998 danube pin_radiusd:20320 pin_radiusd.c:4143
authPinPwd returning 0
D Fri May 22 15:19:52 1998 danube pin_radiusd:20320 pin_radiusd.c:4143
Sending Ack of id 0 to 127.0.0.1 (localhost)
```

This example shows the return packet:

```plaintext
Request-Type                     = Auth-Ack
User-Service                   = 2
Framed-Protocol                = 1
Ascend-Route-IP                = 1
Framed-Netmask                 = 255.255.255.0
```
Configuring VPDN Service for an Account

If you are implementing VPDN using the /service/vpdn storable object, you configure the following options for a customer account by using Customer Center:

- The protocol the customer will use (PPP or SLIP)
- The IP address of the customer’s host machine (xxx.xxx.xxx.xxx)

---

**Ascend-Idle-Limit** = 120
**Ascend-Link-Compression** = 1
**Ascend-Assign-IP-Pool** = 1
**Custom-Attr-1** = "testing_nas"

### Setting VPDN Attributes during Account Creation

You can use the PCM_OPTERM_POL_AUTHORIZE opcode to pass the required VPDN attributes to PCM_OP_CUST_POL_PRE_COMMIT and PCM_OP_CUST_PREP_CUSTOMER. The attributes become part of the inherited service information in the /service/ip or /service/vpdn storable object. See the corresponding opcode information for details.

Both models log sessions as /event/session/dialup events in the corresponding service object. The organization model VPDN is a superset of the user model VPDN. The instructions for setting up the organization model assume that the user model VPDN has been implemented.

Table 12–1 maps RADIUS attributes to fields in the /event/session/dialup service object:

**Table 12–1 Radius Attributes Mapping**

<table>
<thead>
<tr>
<th>RADIUS Attribute</th>
<th>BRM Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Name</td>
<td>PIN_FLDBLD_LOGIN</td>
</tr>
<tr>
<td>NAS-Identifier/NAS-IP-Address</td>
<td>PIN_FLDBLD_TERMSERV_ID</td>
</tr>
<tr>
<td>NAS-Port</td>
<td>PIN_FLDBLD_TERMSERV_PORT</td>
</tr>
<tr>
<td>Acct-Terminate-Cause</td>
<td>PIN_FLDBLD_TERMINATE_CAUSE</td>
</tr>
<tr>
<td>NAS-Port-Type</td>
<td>PIN_FLDBLD_TYPE</td>
</tr>
<tr>
<td>network</td>
<td>PIN_FLDBLD_NETWORK</td>
</tr>
<tr>
<td>timezone</td>
<td>PIN_FLDBLD_TIMEZONE</td>
</tr>
<tr>
<td>Acct-Session-Id</td>
<td>PIN_FLDBLD_TRANS_ID</td>
</tr>
<tr>
<td>Called-Station-Id</td>
<td>PIN_FLDBLD_DNIS</td>
</tr>
<tr>
<td>Calling-Station-Id</td>
<td>PIN_FLDBLD_ANI</td>
</tr>
<tr>
<td>Ascend-Data-Rate</td>
<td>PIN_FLDBLD_SPEED</td>
</tr>
<tr>
<td>Framed-IP-Address</td>
<td>PIN_FLDBLD_IPADDR</td>
</tr>
<tr>
<td>desc</td>
<td>PIN_FLDBLD_DESCR</td>
</tr>
<tr>
<td>Acct-Input-Octets</td>
<td>PIN_FLDBLD_BYTES_IN</td>
</tr>
<tr>
<td>Acct-Output-Octets</td>
<td>PIN_FLDBLD_BYTES_OUT</td>
</tr>
<tr>
<td>Acct-Input-Packets</td>
<td>PIN_FLDBLD_PACKETS_IN</td>
</tr>
<tr>
<td>Acct-Output-Packets</td>
<td>PIN_FLDBLD_PACKETS_OUT</td>
</tr>
</tbody>
</table>

---

12-6  BRM RADIUS Manager
Implementing the Organization Model

With this model, the organization controls which users have access to their network, and pays an ISP to provide the network infrastructure and employee access. The organization model must be used along with the user model. The user model must be implemented prior to setting up the organization model.

Organization Model Example

1. The user dials his local ISP and logs in as user@isp.com.
2. The NAS sends an Access-Request request to the RADIUS server, which sends an authentication request to BRM.
3. The authentication request fails because there is no service login of user@isp.com in the BRM database.
4. The RADIUS server picks up this authentication failure, and passes the request through to the module instance configured to handle VPDN requests.
5. The authentication request is sent to BRM again, this time with a login name containing company.com. The request in checked against the /service/vpdn service.
6. The authentication is successful, and PCM_OP_TERM_POL_AUTHORIZE is called to find session creation attributes.
7. As in the user model, attribute-value pairs are selected from the PIN_FLD_ARGS array in the /service/vpdn storable object, and merged into the attributes returned to the NAS.
8. The NAS uses the tunnel setup attributes and creates a tunnel through the appropriate gateway.
9. When a session starts and ends, the NAS also sends Accounting-Request requests to the RADIUS server, which passes these requests to BRM.
10. The requests cause an /event/session/dialup event to be created, which is then rated and charged to the account for company.com.

Handling VPDN Requests with the Organization Model

Most installations need to support both VPDN models. This section provides information and example solutions for supporting the two VPDN models. Both VPDN models can be implemented using the /service/ip or /service/vpdn storable object. However, using the /service/vpdn storable object is recommended because this allows the /service/ip storable object to support the /account dialup function.

The /service/vpdn storable object is designed to support your VPDN. Perform the following tasks to use the /service/vpdn storable object to support the organization model VPDN:

1. Verify that the /service/vpdn storable object is in your BRM schema.
   If you are running BRM, the /service/vpdn storable object is installed when the system is first initialized.

- Whether or not header compression should be used
- Any protocol extensions (name/value pairs that control the IP connection)
2. Create price plans that incorporate the /service/vpdn storable object for creation of VPDN organization account. See Pricing Center Help for more information about creating plans.

3. Configure the mod_pin module instance in the RADIUS configuration file (BRM_home/apps/radius/config) to use the /service/vpdn storable object.

4. Configure the PIN_FLD_ARGS array in the /service/vpdn storable object for each organization mode VPDN account.

5. Combine the user and organization VPDN models in the RADIUS configuration file.

This example shows the syntax in the RADIUS configuration file for setting up the user model VPDN:

```plaintext
# Cannot find the account, see if the domain is # a registered organization VPDN domain.
vpdn_auth {
    type = mod_pin
    check {
        Pseudo-Request-Type = Access-Request
    }
    send {
        Class = vpdn
    }
    service-type = /service/vpdn
    login = Replace//*@// # authenticate against the domain # name only
    options = Ignore-No-Entry
}
# Rate the session under the organization account.
# Save the actual user name in description for reference.
vpdn_acct {
    type = mod_pin
    check {
        Pseudo-Request-Type = Accounting-Request
        Class = vpdn
    }
    service-type = /service/vpdn
    login = Replace//*@// # record session against the # organization account
desc = "%u"
}
```

**Example Combining the User and Organization Models**

To use the user and organization VPDN models together, put both models in the RADIUS configuration file (BRM_home/apps/radius/config). This example shows the RADIUS configuration file syntax for combining the user and organization model VPDN:

```plaintext
# Cannot find the account, see if it is a user_vpdn
user_vpdn_auth {
    type = mod_pin
    check {
        Pseudo-Request-Type = Access-Request
        User-Name = **@**
    }
    send {
        Class = user_vpdn
    }
```
Implementing the Organization Model

Using a Virtual Private Dialup Network (VPDN)

```
}

service-type = /service/vpdn
options = Ignore-No-Entry
}

user_vpdn_acct {
    type = mod_pin
    check {
        Pseudo-Request-Type = Accounting-Request
        Class = user_vpdn
    }
    service-type = /service/vpdn
}

# ... some other modules ...
# Cannot find the account, see if the domain is
# a registered organization VPDN domain.

vpdn_auth {
    type = mod_pin
    check {
        Pseudo-Request-Type = Access-Request
    }
    send {
        Class = vpdn
    }
    service-type = /service/vpdn
    login = Replace/*@//
    # authenticate against the domain
    # name only
    options = Ignore-No-Entry
}

# Rate the session under the organization account.
# Save the actual user name in description for reference.

vpdn_acct {
    type = mod_pin
    check {
        Pseudo-Request-Type = Accounting-Request
        Class = vpdn
    }
    service-type = /service/vpdn
    login = Replace/*@//  # record session against the
    # organization account
    desc = "@u"
}
```
This chapter explains how to use Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager with iPass roaming services. Before implementing support for iPass roaming services, you should be familiar with:

- Configuring RADIUS Manager
- Understanding RADIUS Manager Modules
- Using the Authentication and Authorization Modules

### About iPass Roaming

RADIUS Manager supports both local and remote iPass authentication and accounting. A typical iPass roaming solution:

- Performs all local and remote iPass tasks.
- Uses online accounting.
- Uses local authentication by BRM.
- Uses local accounting by BRM.

### iPass Checklist

Before you configure RADIUS Manager to support iPass roaming, you must perform these tasks:

- Obtain the iPass NetServer software and an Internet service provider (ISP) account number from iPass.
- Install and set up the NetServer software per the instructions included with the software. The NetServer software package contains the RoamServer software (VNAS) and iPass-enabled RADIUS software.

### Identifying and Forwarding Requests

In most installations, requests for iPass roaming can’t be immediately identified. Because there are no attribute patterns in the request packet that allow RADIUS Manager to identify the request as an iPass request, this procedure is used:

1. The `pin_radiusd` server forwards the request to an iPass-enabled RADIUS server.
2. The iPass-enabled RADIUS server uses the `mod_proxy` module to forward the request to the iPass central server.
Configuring mod_proxy

For iPass roaming requests to be processed:

- The login name must include an @ symbol. The mod_proxy module must be configured to check for this symbol.

- Any authentication modules configured in the module chain that precede the BRM authentication module, mod_pin, must not authenticate the user. To satisfy this criteria, all authentication modules that precede mod_proxy in the chain must use the Ignore-No-Entry option. Setting the Ignore-No-Entry option allows the request to pass on to the next module instead of returning a NAK (not acknowledged).

- The mod_proxy module should be the last authentication module in the chain.

After the authentication request is passed to the iPass-enabled RADIUS server, the NAS (network access server) sends the corresponding accounting requests to signal the start and end of the session. Use the send option to append the Class = "ipass" attribute-value pair in the auth_accept reply to the NAS and check for the appropriate Class attribute value.

You must configure mod_proxy to forward the requests to the iPass-enabled RADIUS server.

This example shows how to set the destination of the iPass-enabled RADIUS server in the $CORE section of the RADIUS configuration file (BRM_home/apps/radius/config, where BRM_home is the directory in which BRM components are installed):

```
$CORE {
...
  mod_proxy {
    server {
      name = ipass_proxy_auth
      addr = 10.15.210.14
      port = 1812
      secret = testing123
      timeout = 30
    }
    server {
      name = ipass_proxy_acct
      addr = 10.15.210.14
      port = 1813
      secret = testing123
      timeout = 30
    }
  }
...
}
```

This example shows how to configure the $MODULES section of the RADIUS configuration file (BRM_home/apps/radius/config) to forward requests to iPass:

```
ipass_auth {
  type = mod_proxy
  check {
    Pseudo-Request-Type = Access-Request
  }
  send {
    Class="ipass"
  }
}
Recording Locally Serviced iPass Sessions

All locally serviced iPass sessions are recorded as `/event/service/dialup` event objects and stored in the `ipass` account class. Locally serviced iPass sessions are identified by the class attribute-value pair, `Class = "ipass"`. Put the `mod_pin` module immediately after the `mod_proxy` module.

```
Note: The iPass account must already exist and contain at least one /service/ip account object prior to implementing the recording functions.
```

This example shows the syntax in the RADIUS configuration file for iPass sessions:

```
serviced_ipass_session {
  type = mod_pin
  check {
    Pseudo-Request-Type = Accounting-Request
    Acct-Status-Type = Stop
    Class="ipass"
  }
  login="ipass"
  desc="%u"
  options=Ignore-No-Entry
}
```

Handling Remote Authentication Requests

Remote authentication requests forwarded from iPass through the Virtual Network Access Server (VNAS) daemon are handled by the `mod_pin` module. This module handles both remote authentication and accounting requests. You must configure the `check` parameter to handle these requests.

This example shows how to configure the `check` parameter in the `mod_pin` module in the `$MODULES` section of the RADIUS configuration file (`BRM_home/apps/radius/config`):

```
ipass_remote {
  type = mod_pin
  check {
    NAS-Identifier = "I-PASS VNAS"
  }
}
```
Handling Online and Batch Requests

The iPass model offers two methods of handling accounting information: online and batch. These methods require different configurations.

**Caution:** Online and batch accounting should never be used concurrently with iPass. Changing from one method to the other may cause double billing.

iPass sessions are often rated differently from local dial-up sessions. If this is the case, modify the PCM_OP_TERM_POL_ACCOUNTING policy opcode to rate the sessions differently.

When the policy is activated by an iPass remote Accounting-Off request, the PIN_FLD_TERMSERV_ID field contains I-PASS VNAS. The policy opcode can use this field in the iPass sessions to rate sessions differently.

**Online Accounting**

Most iPass member ISPs use online accounting, which works well with the BRM billing model. When an iPass session is finished, the accounting records are forwarded in real time from the remote ISP to the home ISP of the roaming user. When you use online accounting, sessions are recorded as /event/session/dialup objects.

**Batch Accounting**

Call detail report (CDR) files support batch accounting. The pin_ipass_loader utility loads batch accounting records from CDR files. When you use batch accounting, sessions are recorded as /event/session objects. See "Using the pin_ipass_loader Utility for Batch Accounting".

**Note:** The PIN_FLD_DESCR field for the /event/session is a concatenation of the Description, Billing Code, GMT Time, Session charge per hour, and Dollar charge from iPass to home ISP fields from the CDR.

**Logging Accounting Packets for Batch Accounting**

To support batch accounting, all accounting packets forwarded from VNAs must be discarded or logged to a log file. To discard them, add this module configuration to the RADIUS configuration file (BRM_home/apps/radius/config):

```plaintext
discard_vnas_acct {
  type=mod_null
  check {
    Pseudo-Request-Type = Accounting-Request
    NAS-Identifier = "I-PASS VNAS"
  }
}
```
Sessions are loaded into `/event/session` objects from the iPass CDR file.

### Logging Accounting Packets for Online Accounting

To log the accounting packets to a log file, add this module configuration to the RADIUS configuration file (`BRM_home/apps/radius/config`):

```
log_vnas_acct {
    type=mod_logging
    dest=ipass_acct.log
    check {
        Pseudo-Request-Type = Accounting-Request
        NAS-Identifier = "I-PASS VNAS"
    }
}

discard_vnas_acct {
    type=mod_null
    check {
        Pseudo-Request-Type = Accounting-Request
        NAS-Identifier = "I-PASS VNAS"
    }
    action = ack
}
```

**Note:** The `mod_null` module is required to ACK the requests properly because `mod_logging` only records the session and then passes on the request.

### iPass Call Detail Record File Definition

A CDR file is an ASCII file containing one record per connection. Records are comma-delimited, and character strings are surrounded by double quotation marks. Numeric fields can include a decimal point.

**Table 13–1** shows the CDR values with reference to BRM.

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
<th>Stored in BRM As</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ID</td>
<td>The unique ID for a transaction.</td>
<td>PIN_FLD_EVENT_NO</td>
</tr>
<tr>
<td>Billing Code</td>
<td>Code used by iPass, but ignored by pin_ipass_loader.</td>
<td>PIN_FLD_DESCR</td>
</tr>
<tr>
<td>Login</td>
<td>User ID for this session.</td>
<td>PIN_FLD_SERVICE_OBJ</td>
</tr>
<tr>
<td>Domain</td>
<td>Added to Login when -u is specified.</td>
<td>PIN_FLD_SERVICE_OBJ</td>
</tr>
<tr>
<td>Description</td>
<td>Normally the remote location. Can also be used for “Monthly Fee”.</td>
<td>PIN_FLD_DESCR</td>
</tr>
<tr>
<td>GMT Time</td>
<td>Ignored by pin_ipass_loader.</td>
<td>PIN_FLD_DESCR</td>
</tr>
<tr>
<td>Local Time</td>
<td>Indicates the session end time.</td>
<td>PIN_FLD_END_T</td>
</tr>
<tr>
<td>Session Length</td>
<td>Can be used in rating calculations. Subtracted from Local Time to determine the session start time.</td>
<td>PIN_FLD_START_T</td>
</tr>
</tbody>
</table>
This example shows how to configure the RADIUS configuration file (BRM_home/apps/radius/config) to support a typical iPass roaming solution:

```plaintext
$CORE {
    ...
    mod_proxy {
        server {
            name = ipass_proxy_auth
            addr = 10.15.210.14    #proxy ip address
            port = 1812
            secret = testing123
            timeout = 30
        }
        server {
            name = ipass_proxy_acct
            addr = 10.15.210.14    #proxy ip address
            port = 1813
            secret = testing123
            timeout = 30
        }
        ...
    }
    ...
    ...
}
$MODULE {
    ipass_remote {
        type = mod_pin
        check {
            NAS-Identifier = "I-PASS VNAS"
        }
    }
    pin_local_auth {
```

**Table 13–1 (Cont.) CDR Values**

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
<th>Stored in BRM As</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session charge per hour</td>
<td>Session rate used by iPass, but ignored by pin_ipass_loader.</td>
<td>PIN_FLD_DESCR</td>
</tr>
<tr>
<td>Dollar charge from iPass to home ISP</td>
<td>Can be used in rating calculation and to calculate a balance impact.</td>
<td>PIN_FLD_DESCR</td>
</tr>
</tbody>
</table>

**Note:**

- Currency in a CDR file must be US dollars (USD).
- The EVENT_ID field in the /event object checks for uniqueness against the trans-id field in the CDR file. A session with the same transaction ID is loaded only once into the database.
- To add to the amount charged by iPass, rate by cost of session.
- To pass the monthly fee to the subscriber, use the monthly fee option with pin_ipass_loader.

**Example RADIUS Configuration File for iPass**

This example shows how to configure the RADIUS configuration file (BRM_home/apps/radius/config) to support a typical iPass roaming solution:

```plaintext
$CORE {
    ...
    mod_proxy {
        server {
            name = ipass_proxy_auth
            addr = 10.15.210.14    #proxy ip address
            port = 1812
            secret = testing123
            timeout = 30
        }
        server {
            name = ipass_proxy_acct
            addr = 10.15.210.14    #proxy ip address
            port = 1813
            secret = testing123
            timeout = 30
        }
        ...
    }
    ...
    ...
}
$MODULE {
    ipass_remote {
        type = mod_pin
        check {
            NAS-Identifier = "I-PASS VNAS"
        }
    }
    pin_local_auth {
```
Example RADIUS Configuration File for iPass

Setting Up iPass Roaming

```

```
Using the \texttt{pin\_ipass\_loader} Utility for Batch Accounting

The \texttt{pin\_ipass\_loader} utility loads the iPass CDRs into the BRM database. This utility reads and processes the specified CDR files line by line. For each line, the utility checks that the transaction ID does not already exist in the database and then applies the charges to the database.

The utility logs any errors to the \texttt{default.pinlog} file and moves to the next record. When processing is complete, the utility prints statistics for the batch and then indicates whether any errors occurred.

See "\texttt{pin\_ipass\_loader}" for parameter definitions and iPass Call Detail Record File Definition.

Running \texttt{pin\_ipass\_loader}

To run \texttt{pin\_ipass\_loader}, use this command:

\begin{verbatim}
pin_ipass_loader -f input_file
\end{verbatim}

The following example shows a CDR file with a single call.


\begin{verbatim}
"db4:2448","0","userb","anynet.com","Korea","02-Nov-2001 02:50:48","02-Nov-2001 02:50:48","600","1.00", "5.00"
\end{verbatim}

See "\texttt{pin\_ipass\_loader}".

The \texttt{pin\_ipass\_loader} utility provides two possible ways to rate a session: based on session length or dollar charge from iPass. Either or both parameters can be used.

The optional parameters -s, -c, -u and -m determine the specific behavior of \texttt{pin\_ipass\_loader} as follows:

\begin{itemize}
  \item To rate based on session length, specify the rate name after the -s option. The default is \texttt{ip/dialup/async/hourly}. BRM silently ignores this part of the charge if it cannot be rated.
  \item To rate based on session cost, specify the rate name after the -c option. The default is no rate name. BRM cannot rate a charge without a rate name. If you don’t specify a rate for the -c option, BRM silently ignores all dollar charges in the CDR file.
  \item To rate based on a monthly charge, specify the rate name after the -m option. The default is to print an error. iPass indicates "monthly charges" as charges with a session length of zero. BRM will rate the session. BRM silently ignores this part of the charge if it cannot be rated. To ignore monthly charges, specify a dummy string along with the -m option.
\end{itemize}

Error Handling

\texttt{pin\_ipass\_loader} processes one line at a time. If an error occurs, details are recorded to the problem file and the next record is processed. When processing completes, a message prints if any errors occurred. A line error occurs in following circumstances:

\begin{itemize}
  \item The line fails to parse, possibly due to an incorrect number of fields, bad syntax, or invalid values. Only limited data guarding is performed because the CDR file is computer generated.
  \item The transaction ID is already in the database. This prevents loading the same record multiple times.
\end{itemize}
The user is not found in the database. Checks that the unqualified/qualified options are appropriately specified.

A monthly charge (when session_length_rate is set to 0) is found and the -m option was not specified.

The PCM_OP_ACT_LOAD_SESSION opcode returns an error.

---

**Note:** `pin_ipass_loader` silently fails if BRM cannot rate the session. This can happen if an unknown rate name is used with the `-s`, `-c`, or `-m` option, or if no `-c` option is specified and the user wants to capture the US dollar (USD) charge from iPass.
Testing a RADIUS Configuration

This chapter describes how to use the `rad_tester` utility to test a RADIUS configuration and how to use the `pin_term_acct` utility to test sending account request packets to the RADIUS server and monitor open sessions.

Before reading this chapter, you should be familiar with:

- Configuring RADIUS Manager
- Understanding RADIUS Manager Modules
- Using the Authentication and Authorization Modules

About the `rad_tester` Utility

The `rad_tester` utility replicates the functionality of a terminal server by sending RADIUS packets to the RADIUS server. This is useful for testing and developing code. After a request is accepted, you can manually send another request packet. You can view the packets sent by `rad_tester` in the debug log file `/var/portal/7.4/radius/radius.log`.

---

**Note:** This utility is *not* designed for performance testing or load testing where concurrency is important.

---

See "`rad_tester`" for information on utility parameters. See "Running `rad_tester`" for information on using this utility.

Configuring `rad_tester`

Before you use `rad_tester`, configure Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager. You must also configure the accounting and authentication ports, the client from where you will run the `rad_tester` utility, and the dictionary file. You can then create the authentication, start accounting, and stop accounting RADIUS input packet files. Then you run `rad_tester` to simulate RADIUS activity.

See "Creating Input Packets".

Configuring the Accounting and Authentication Ports

The accounting and authentication ports are defined in the `$CORE` section of the RADIUS configuration file (`BRM_home/apps/radius/config`, where `BRM_home` is the directory in which BRM components are installed). This was done during RADIUS server configuration. Ensure that the ports are properly configured.
About the rad_tester Utility

See "Setting the IP Port Numbers".

For example:

```
listen {
    port = 1812
}
listen {
    port = 1813
}
```

Defining the Client in the RADIUS Configuration File

Ensure that the client computer is included in the client list. The client list is defined in the $CORE section of the RADIUS configuration file (BRM_home/apps/radius/config).

For example:

```
client {
    addr = 156.151.55.9
    secret = testing123
}

client {
    addr = 156.151.37.81
    secret = testing123
}
```

Creating Input Packets

For each simulated user, you must create three input RADIUS packets: auth, start, and stop. You can include all three packets in a single file, or you can use three separate files (to test opcode, use a single file; to simulate timing, use three files). Place these files in a directory on the client.

Each packet is defined by a series of attribute-value pairs delimited by one or more blank lines. Use this format for attribute-value pairs:

```
Attribute_name = Attribute value
```

Note: In rad_tester, blank lines indicate the end of a record. Use blank lines only when formatting three packets in one file.

Simulating Multiple Users

Simulate different users by making copies of the three input files (as many as you like) and changing the user names and passwords, as well as other attributes. Each user can have three input packets (auth, start, stop). The specified user names must be valid users.

Running rad_tester

Before starting rad_tester, ensure that the RADIUS server is running. To run rad_tester, use the following command:

```
rad_tester -h host_name -p 1812 -a 1813 -s testing123 -O 30 -f input_packet
```

See "rad_tester" for parameter descriptions.
The following examples show how to simulate authentication and how to start and stop accounting.

**Sample Authentication Request**

An **auth** input packet simulates an authentication request from a Network Access Server (NAS). The specified user name must be a valid user.

For example:

```
Request-Type = Auth-Req
User-Name = username
User-Password = userpassword
NAS-Identifier = 1.1.1.1
NAS-Port = 1
```

**Sample Start Accounting Request**

A **start** input packet simulates a start accounting request from the NAS. The **Acct_Session_Id** value must be changed to a different value each time this request is sent to RADIUS Manager. Each session ID number must be unique because RADIUS Manager discards duplicate requests.

For example:

```
Request-Type = Acct-Req
User-Name = username
NAS-Identifier = 1.1.1.1
NAS-Port = 1
Acct-Status-Type = Start
Acct-Session-Id = 10
```

**Sample Stop Accounting Request**

A **stop** input packet simulates a stop accounting request from the NAS. The **Acct_Session_Id** value must be identical to the one specified in the corresponding **start** request packet.

For example:

```
Request-Type = Acct-Req
User-Name = username
NAS-Identifier = 1.1.1.1
NAS-Port = 1
Acct-Status-Type = Stop
Acct-Session-Id = 10
Acct-Delay-Time = 1
Acct-Session-Time = 3600
```

---

**Note:** You can add any parameters as long as they conform to the data dictionary and *RFC 2865: Remote Authentication Dial In User Service (RADIUS)*.

---

**Sample Interim Accounting Request**

An **interim** input packet simulates an update accounting request from the NAS. The **Acct_Session_Id** value must be identical to the one specified in the corresponding **start** request packet.

For example:

```
Request-Type = Acct-Req
```
User-Name = username
NAS-Identifier = 1.1.1.1
Framed-IP-Address = 1.1.1.1
NAS-Port = 1
Acct-Status-Type = Interim-Update
Acct-Session-Id = 10
Acct-Delay-Time = 1
Acct-Session-Time = 3600

Sending Input Packets to the RADIUS Server
You manually send the sample input packets to the RADIUS server. Use the following commands to simulate authentication and to start and stop accounting for users:

```plaintext
rad_tester -h host_name -p 1812 -a 1813 -s testing123 -O 30 -f auth_pkt.sample
rad_tester -h host_name -p 1812 -a 1813 -s testing123 -O 30 -f start_pkt.sample
rad_tester -h host_name -p 1812 -a 1813 -s testing123 -O 30 -f stop_pkt.sample
```

About the Account Request Utility
You use the `pin_term_acct` utility to perform the following tasks:

- Simulate receiving Accounting-On or Accounting-Off requests from a terminal server by manually calling the PCM_OP_IP_DIALUP_ACCOUNTING_ON and PCM_OP_IP_DIALUP_ACCOUNTING_OFF opcodes from RADIUS Manager.
  This is useful if the terminal server crashes before sending an Accounting-Off packet. The time parameter (-t) for this utility allows sessions on the affected terminal server to be closed and billed at or before the time of the terminal server crash. This ensures that subscribers are not overbilled for connection time.

- Create a list of all opened sessions based on TERMSERV_ID.
  This list helps you monitor sessions open on one or all of your terminal servers and assess the impact of rebooting the terminal server.

See "pin_term_acct" for parameter descriptions. See "Specifying How Accounting-On Events Are Handled after a NAS Failure".

Sending an Accounting-Off Request
This example shows how to send an Accounting-Off (-m off) request in charge mode (-c C) to a terminal server (-n) at 2:00 (-t 2:00:00). Verbose mode (-v) is on so that configuration information displays.

```plaintext
% pin_term_acct -m off -t 2:00:00 -v -n 1.1.1.3 -c C
```

Configuration Information:
```
-------------------------------
verbose    = on
nas ID     = 1.1.1.1
sent time  = Fri Jan 23 15:48:09 1998
run mode   = off
charge mode= C
-------------------------------
```

Listing All Open Sessions
This example displays the output for `pin_term_acct` in -m list mode.

```plaintext
% pin_term_acct -m list
```
In this example, the first column lists the terminal server ID to which the session is logged on (1.1.1.3); the second column lists the user ID for the session (billing0x); the third column lists the IP address of the session; the remaining column lists the session start time.
This chapter provides reference information for Oracle Communications Billing and Revenue Management (BRM) RADIUS Manager utilities.
pin_ipass_loader

Use the pin_ipass_loader utility to load the iPass call detail records (CDRs) into the BRM database.

For more information, see "Using the pin_ipass_loader Utility for Batch Accounting" and "Setting Up iPass Roaming".

---

**Important:** To connect to the BRM database, the pin_ipass_loader utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in BRM System Administrator’s Guide.

---

**Location**

BRM_home/bin

**Syntax**

```bash
pin_ipass_loader -f input_file -s session_length_rate [-c cost_rate] -u qualified | unqualified
-p problem_file [-m] monthly_charge_rate [-t] [-h] [-v]
```

**Parameters**

- `-f input_file`
  Specifies the name of the CDR file. The default is stdin. See "iPass Call Detail Record File Definition".

- `-s session_length_rate`
  Specifies the rate based on session length. The default is ip/dialup/async/hourly.

- `-c cost_rate`
  Rate based on session cost rating.

- `-u qualified | unqualified`
  Includes or excludes the domain in user name searches. The default is unqualified.

- `-p problem_file [-m] monthly_charge_rate [-t] [-h] [-v]`
  Specifies the name of problem file. The default is pin_ipass_loader.problem.yyyymmddhhmm. The file is placed by default in BRM_home/.

- `-m monthly_charge_rate`
  Rate based on monthly iPass charges. Optional.

- `-t`
  Specifies test mode, which parses the file and runs consistency checks without modifying the database.

- `-v`
  Displays information about successful or failed processing as the utility runs.

- `-h`
  Displays the syntax and parameters for this utility.
Results

If the utility doesn’t notify you that it was successful, look in the default.pinlog file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.
Use the `pin_radiusd_sig` utility to initiate dynamic reconfiguration of the RADIUS server, extract information about the RADIUS server, and stop the RADIUS server.

For more information, see "Reconfiguring Your RADIUS Server without Stopping Operation".

---

**Important:** To connect to the Oracle Communications Billing and Revenue Management (BRM) database, the `pin_radiusd_sig` utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator’s Guide*.

---

### Location

`BRM_home/bin`

### Syntax

```
pin_radiusd_sig HUP|USR1|INT host_name port secret
```

### Parameters

**HUP**

Initiates dynamic reconfiguration. This parameter causes the server to reload the `pin_radiusd_config` file containing the changes that you want to deploy.

**USR1**

Logs status information about the RADIUS server to the `radius.pinlog` log file, `/var/portal/7.4/radius/radius.pinlog`.

**INT**

Shuts down the RADIUS server.

**host_name**

Specifies the name of the RADIUS server host computer.

**port**

Specifies the port number that the server listens to for requests.

**secret**

Specifies the secret shared between the client NAS (network access server) and RADIUS Manager.

### Results

If the utility doesn’t notify you that it was successful, look in the `default.pinlog` file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.
pin_term_acct

Use the pin_term_acct utility to simulate receiving Accounting-On or Accounting-Off requests from a terminal server by manually calling the PCM_OP_IP.Dialup_ACCOUNTING_ON and PCM_OP.IP.Dialup_ACCOUNTING_OFF opcodes from RADIUS Manager. You can also use pin_term_acct to monitor all opened sessions based on TERMSERV_ID.

For more information, see "About the Account Request Utility".

---

**Note:** To connect to the Oracle Communications Billing and Revenue Management (BRM) database, the pin_term_acct utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in BRM System Administrator’s Guide.

---

**Location**

*BRM_home/bin*

The configuration file for this utility is located in *BRM_home/apps/pinapps/pin_term_acct/pin.conf*.

**Syntax**

```
pin_term_acct -m list|on|off -n termserv_id -v verbose -d days
-t hh:mm:ss -c charge mode [-h] help
```

**Parameters**

- **-m list|on|off**
  
  Sets the mode of operation.
  
  - list mode (default) displays a list of all open sessions.
  
  - on sends an Accounting-On request.
  
  - off sends an Accounting-Off request.

  You use these parameters to simulate end-of-cycle usage. See "Listing All Open Sessions" for the output format.

- **-n termserv_id**
  
  Specifies the terminal server ID. This parameter is optional in list mode. If not specified in list mode, the default is all terminal servers.

- **-v**
  
  Displays information about successful or failed processing as the utility runs.

- **-d days**
  
  Specifies the date that an Accounting-On/Accounting-Off event occurs. The default is today’s date.

- **-t hh:mm:ss**
  
  Specifies the time that an Accounting-On/Accounting-Off event occurs. The default is the current time. Hours are in 24-hour format.
-c NCIC
Specifies the charge mode.
- NC (the default) does not charge sessions that were closed in off mode.
- C charges sessions that were closed in off mode.
This parameter has no effect in list and on modes.

-h
Displays the syntax and parameters for this utility.

Results

If the utility doesn’t notify you that it was successful, look in the default.pinlog file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.
rad_tester

Use the \textit{rad_tester} utility to send RADIUS packets to the RADIUS server to replicate a terminal server.

For more information, see "Testing a RADIUS Configuration".

\textbf{Important:} To connect to the Oracle Communications Billing and Revenue Management (BRM) database, the \textit{rad_tester} utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in \textit{BRM System Administrator's Guide}.

\section*{Location}

\textit{BRM_home/bin}

\section*{Syntax}

\texttt{rad_tester -h host_name -p port_number -a port_number -s secret -f input_file [-r input_file [-O] sss-d dictionary_file_name [-D] [-v]}

\section*{Parameters}

\textbf{Note:} These parameters are case sensitive.

- \texttt{-h host_name}
  Host name of the RADIUS server. The default is the local host name.

- \texttt{-p port_number}
  Authorization port.

- \texttt{-a port_number}
  Accounting port.

- \texttt{-s secret}
  Shared RADIUS secret. The default is \texttt{testing123}.

- \texttt{-f input_file}
  Input file name. The default is \texttt{stdin}.

- \texttt{-r input_file}
  Number of times to re-send a request when the request times-out. The default is 3.

- \texttt{-O sss}
  Timeout value in seconds to wait for an ACK to a send request. The default is 10.

- \texttt{-d dictionary_file_name}
  Specifies the dictionary file. See "Selecting the Data Dictionary".

- \texttt{-D}
  Debug mode.
-v
Displays information about successful or failed processing as the utility runs.

**Supported Attributes and Request Types**
All attributes included in the data dictionary are supported.

**Results**
If the utility doesn’t notify you that it was successful, look in the default.pinlog file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.