

## Using Prepaid Billing

---

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

Cisco Access Registrar (CAR) supports two types of prepaid billing, IS835C and Cisco Real-time Billing (CRB), a Cisco proprietary solution. The IS835C version adheres to industry standards and is the preferred version.

Three components are required to support a prepaid billing service:

- AAA client,
- CAR server
- External prepaid billing server

The most important factor for an effective prepaid billing service is in developing a shared library to be configured under the prepaid RemoteServer object. The shared library should be developed to implement all specified API functions. You will have to provide a shared library that meets the needs of your environment. The shared library must implement the API functions to perform the various tasks required for your specific implementation of the prepaid billing service.



**Note**

---

Cisco works with you to develop the prepaid billing service and implement the API. For more information, contact your Cisco systems engineer.

---

The chapter contains the following sections:

- [Overview](#)
- [IS835C Prepaid Billing](#)
- [CRB Prepaid Billing](#)
- [Implementing the Prepaid Billing API](#)

## Overview

When a subscriber uses a prepaid billing service, each call requires a set of data about the subscriber. However, the AAA network has no previous knowledge of the subscriber's usage behavior. CAR uses an iterative authorization paradigm over multiple sessions to support the prepaid billing solution.

Each time an authorization request is made, the billing server apportions a fraction of the subscriber's balance into a quota. When a subscriber uses multiple sessions, each session must obtain its own quota. When a previously allocated quota is depleted, a session must be reauthorized to obtain a new quota.

**Note**

The granularity and the magnitude of the quota is in the design and implementation of the prepaid billing server and is beyond the scope of this document. In general, a smaller quota generates more network traffic, but allows more sessions per subscriber. When the quota is equal to a subscriber's total account balance, there is minimal network traffic, but only one session can be supported.

When a subscriber's current quota is depleted, the AAA client initiates a reauthorization request sending Access-Request packets. After the CAR server receives the request, it forwards the request to the billing server. The billing server then returns the next quota to use. The new quota might not be the same as the previous, and the billing server might adjust the quota dynamically.

## IS835C Prepaid Billing

CAR acts as a RADIUS protocol head for all the requirements specified in the *cdma2000 Wireless IP Network Standard: PrePaid Packet Data Service* specification:

[http://www.3gpp2.org/Public\\_html/specs/X.S0011-006-C-v1.0.pdf](http://www.3gpp2.org/Public_html/specs/X.S0011-006-C-v1.0.pdf)

As long as the prepaid client understands or accepts what the external billing server sends, the service should work. The CAR server neither imposes nor is affected by the values of attributes returned from the external billing server.

For additional information, see *cdma2000 Wireless IP Network Standard: Accounting Services and 3GPP2 RADIUS VSAs* at the following URL:

[http://www.3gpp2.org/Public\\_html/specs/X.S0011-005-C-v1.0.pdf](http://www.3gpp2.org/Public_html/specs/X.S0011-005-C-v1.0.pdf)

The IS835C specification requires that the CAR server be able to determine that a particular user is a prepaid billing user. A user is accepted as a valid prepaid user when the response dictionary of the incoming packet contains the CAR internal subattribute named *prepaid*.

The IS835C specification requires prepaid users to first be authenticated by the RADIUS server. This requires the configuration of a group service with an authentication service first, followed by the prepaid service that adds prepaid attributes as shown in [Setting Up an Authentication Group Service, page 15-5](#). The group service configuration enables the AA service to add the prepaid subattribute to the response dictionary upon successful authentication, before the prepaid service is invoked.

## Configuring IS835C Prepaid Billing

To configure an IS835C prepaid billing service, use the following sections to configure the required CAR objects:

- [Setting Up a Prepaid Billing RemoteServer](#)
- [Setting Up an IS835C Prepaid Service](#)
- [Setting Up Local Authentication](#)
- [Setting Up an Authentication Group Service](#)

### Setting Up a Prepaid Billing RemoteServer

**Step 1** Use `aregcmd` to add a RemoteServer under `/Radius/RemoteServers`.

```
cd /radius/remoteserver
add prepaid-is835c
```

**Step 2** Set remoteserver protocol to prepaid-is835c.

```
cd prepaid-is835c
set protocol prepaid-is835c

Set Protocol prepaid-is835c
```

The following is the default configuration of a prepaid-is835c RemoteServer.

```
[ //localhost/Radius/RemoteServers/prepaid-is835c ]
Name = prepaid-is835c
Description =
Protocol =
IPAddress =
Port = 0
Filename =
Connections = 8
```

Table 15-1 lists and describes the properties required for an IS835C RemoteServer object.

**Table 15-1** Prepaid-IS835C RemoteServer Properties

Property	Description
Filename	Name of the shared library provided by the billing server vendor, such as <b>libprepaid.so</b>
IPAddress	IP address of the billing server
Port	Port used on the billing server, such as port 66
Connections	Number of threads the prepaid service and billing server can each use (default is 8).

## Setting Up an IS835C Prepaid Service

CAR uses a service type **prepaid** to support the prepaid billing solution. The prepaid service mediates between the client NAS and the external prepaid billing server.

**Step 1** Use **aregcmd** to add a prepaid service under **/Radius/Services**:

```
cd /radius/services
add prepaid

Added prepaid
```

**Step 2** Set the service type to prepaid.

```
cd prepaid
```

**set type prepaid**

```
Set Type prepaid
```

A prepaid service has the following default properties:

```
[ //localhost/Radius/Services/prepaid ]
Name = prepaid
Description =
Type = prepaid
IncomingScript~ =
OutgoingScript~ =
OutagePolicy~ = RejectAll
OutageScript~ =
MultipleServersPolicy = Failover
RemoteServers/
```

**Step 3** Add a reference to the is835c-prepaid RemoteServer.

**cd RemoteServer****add 1 prepaid-is835c**

```
Added 1
```

---

## Setting Up Local Authentication

If you use the CAR server for authentication and authorization in your prepaid billing solution, you should configure an AA service. For example, you might configure a service similar to **local-users** (in the example configuration) for authentication and authorization of local users.

If some of the users are non-prepaid users or if the prepaid users need to have RADIUS authorization attributes returned, you should configure an AA service to perform that authentication and authorization.

---

**Step 1** Use **aregcmd** to set up a local authentication service.

**cd /radius/services****add Prepaid-LocalAuthentication**

```
Added prepaid-LocalAuthentication
```

**cd prepaid-LocalAuthentication**

```
[ //localhost/Radius/Services/prepaid-LocalAuthentication ]
Name = prepaid-LocalAuthentication
Description =
Type =
```

**Step 2** Set the service type to local.

**set type local**

```
Set Type local
```

**Step 3** Set the UserList property to the userlist that contains IS835C prepaid users.

```
set UserList userlist_name
```

```
Set UserList userlist_name
```

**Note**

You can use an LDAP or ODBC service in place of the local authentication service.

The authentication service must add the CAR internal attribute *prepaid* (subattribute 22) to the response upon successful authentication.

## Setting Up an Authentication Group Service

Your prepaid billing solution usually requires a group service to tie together an AA service with a prepaid service, a group service to tie together an accounting service with a prepaid service, or both.

If you are using an AA service with your prepaid billing solution, you must configure a group service, for example **prepaid-users**, that ties the requests to the AA service (**local-users** in our example) with the prepaid service.

If you are using CAR for an accounting service with your prepaid billing solution, you must configure a group service, for example **prepaid-file**, that ties accounting requests to both the regular accounting service (**local-file** in our example) and the prepaid service.

**Step 1** Use **aregcmd** to add a prepaid authentication group service under **/Radius/Services**.

```
cd /radius/services
```

```
add prepaid-groupAuthentication
```

```
Added prepaid-groupAuthentication
```

```
cd prepaid-groupAuthentication
```

```
[ //localhost/Radius/Services/prepaid-groupAuthentication ]
  Name = group-prepaidAuthentication
  Description =
  Type =
```

**Step 2** Set the service type to group.

```
set type group
```

```
Set Type group
```

The group service requires the ResultRule to be set to AND, the default setting for a group service.

```
ls
```

```
[ //localhost/Radius/Services/group-prepaidAuthentication ]
  Name = group-prepaidAuthentication
  Description =
  Type = group
  IncomingScript~ =
  OutgoingScript~ =
```

```
ResultRule = AND
GroupServices/
```

- Step 3** Change directory to GroupServices and add references to the prepaid service and the authentication service.

### cd GroupServices

```
[ //localhost/Radius/Services/group-prepaidAuthentication/GroupServices ]
```

### add 1 Prepaid-LocalAuthentication

```
Added 1
```

### add 2 prepaid

```
Added 2
```

## CRB Prepaid Billing

Cisco Real-Time Billing (CRB) is a Cisco proprietary method of providing prepaid billing service. CAR uses vendor-specific attributes (VSA) to extend the standard RADIUS protocol to carry information not usually present in the standard RADIUS packet. CAR uses a set of VSAs allocated to the Cisco VSA pool [26,9].

CAR required several different types of measurements to support a prepaid billing solution. These measurements require the use of metering variables to perform usage accounting. [Table 15-2](#) lists the different measurements and what the AAA client, CAR server, and billing server do with them.

**Table 15-2** *Measurements and Component Actions*

Measurement Type	Billing Server Action	AAA Server Action	AAA Client Action
Duration	Return duration quota	Convert duration quota to VSAs and pass along	Compare running duration quota with quota returned by CAR server
Total volume	Return volume quota	Convert volume quota to VSAs and pass along	Compare running volume quota with quota returned by CAR server
Uplink volume	Return volume quota	Convert volume quota to VSAs and pass along	Compare running volume quota with quota returned by CAR server
Downlink volume	Return volume quota	Convert volume quota to VSAs and pass along	Compare running volume quota with quota returned by CAR server

**Table 15-2** *Measurements and Component Actions (continued)*

Measurement Type	Billing Server Action	AAA Server Action	AAA Client Action
Total packets	Return packet quota	Convert packet quota to VSAs and pass along	Compare running packet quota with quota returned by CAR server
Uplink packets	Return packet quota	Convert packet quota to VSAs and pass along	Compare running packet quota with quota returned by CAR server
Downlink packets	Return packet quota	Convert packet quota to VSAs and pass along	Compare running packet quota with quota returned by CAR server
Logical OR of two measurements	Return quota of both measurements	Convert both to VSA and pass along	Monitor both quota and issue reauthorization packet when any one trips

CAR provides maximum flexibility to billing servers by allowing the metering variable to be modified as the service is used. This requires network nodes to measure all parameters all the time, but to report values only after receiving a reauthorization request.

**Note**

If you have been using an earlier implementation of CRB prepaid billing (CAR 3.5.2 or earlier), you must recompile the API implementation with the newer API due to the addition of the parameter `ebs_context` as the first parameter to all API methods. Contact your Cisco systems engineer for assistance with the new API.

## Configuring CRB Prepaid Billing

To configure an CRB prepaid billing service, use the following sections to configure the required CAR objects:

- [Setting Up a Prepaid Billing RemoteServer](#)
- [Setting Up a CRB Prepaid Service](#)
- [Setting Up a Local Accounting Service](#)
- [Setting Up a Local Authentication Service](#)
- [Setting Up a Prepaid Accounting Group Service](#)
- [Setting Up an Authentication Group Service](#)

If you are using CRB prepaid billing with Service Selection Gateway (SSG), you must also configure extension point scripts and prepaid clients. See [Configuring CRB Prepaid Billing for SSG, page 15-13](#).

### Setting Up a Prepaid Billing RemoteServer

**Step 1** Use `aregcmd` to add a RemoteServer under `/Radius/RemoteServers`.

```
cd /radius/remoteservers
```

**add prepaid-crb**

```
Added prepaid-crb
```

**Step 2** Set the RemoteServer protocol to prepaid-crb.

**cd prepaid-crb****set protocol prepaid-crb**

```
Set Protocol prepaid-crb
```

The following is the default configuration of a prepaid-crb RemoteServer.

```
[ //localhost/Radius/RemoteServers/prepaid-crb ]
Name = prepaid-crb
Description =
Protocol =
IPAddress =
Port = 0
Filename =
Connections = 8
```

Table 15-3 lists and describes the properties required for an CRB RemoteServer object.

**Table 15-3** Prepaid-CRB RemoteServer Properties

Property	Description
Filename	Name of the shared library provided by the billing server vendor, such as <b>libprepaid.so</b>
IPAddress	IP address of the billing server
Port	Port used on the billing server, such as port 66
Connections	Number of threads the prepaid service and billing server can each use (default is 8).

## Setting Up a CRB Prepaid Service

CAR uses a service type **prepaid** to support the prepaid billing solution. The prepaid service mediates between the client NAS and the external prepaid billing server.

The prepaid service must receive accounting requests to accurately charge the prepaid billing user. You can also set the prepaid service in a group service to log accounting requests locally or to proxy the accounting requests to another service or to both locations.

**Step 1** Use **aregcmd** to add a prepaid service under **/Radius/Services**:

```
cd /radius/services
```

**add prepaid**

```
Added prepaid
```

**Step 2** Set the service type to prepaid.

```
cd prepaid
```

```
set type prepaid
```

```
Set Type prepaid
```

A prepaid service has the following default properties:

```
[ //localhost/Radius/Services/prepaid ]
Name = prepaid
Description =
Type = prepaid
IncomingScript~ =
OutgoingScript~ =
OutagePolicy~ = RejectAll
OutageScript~ =
MultipleServersPolicy = Failover
RemoteServers/
```

**Step 3** Add a reference to the prepaid-crb RemoteServer.

```
cd RemoteServers
```

```
add 1 prepaid-crb
```

```
Added 1
```



**Note**

The following steps are required only when using Prepaid-CRB with SSG.

**Step 4** Set the IncomingScript to **IncomingScript PPI-Parse-Prepaid-Incoming**.

```
set IncomingScript PPI-Parse-Prepaid-Incoming
```

```
Set IncomingScript PPI-Parse-Prepaid-Incoming
```

**Step 5** Set the OutgoingScript to **OutgoingScript PPO-Parse-Prepaid-Outgoing**.

```
set OutgoingScript PPO-Parse-Prepaid-Outgoing
```

```
Set OutgoingScript PPO-Parse-Prepaid-Outgoing
```

## Setting Up a Local Accounting Service

If you want to use the CAR server to record the accounting records locally or to forward the accounting records to another RADIUS server, you must configure an accounting service. You might configure a service similar to **local-file** (in the example configuration) for accounting requests. Accounting requests can be logged locally (with an accounting service) or remotely (with a RADIUS service).

If you use the prepaid billing server to generate the accounting records, an accounting service is not necessary.

**Step 1** Use **aregcmd** to add a local accounting service under **/Radius/Services**.

```
cd /radius/services
```

**add prepaid-LocalFileAccounting**

```
add prepaid-LocalFileAccounting
```

**Step 2** Set the service type to file.

**cd prepaid-LocalFileAccounting****set type file**

```
Set Type file
```

The file type service has the following properties:

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

**Step 3** Set the FilenamePrefix to Prepaid-Accounting.

**set FilenamePrefix Prepaid-Accounting**

```
Set FilenamePrefix Prepaid-Accounting
```

**Step 4** Set the MaxFileAge to one hour.

**set MaxFileAge "1 Hour"**

```
Set MaxFileAge "1 Hour"
```

The MaxFileSize should remain at the default value of 10 megabytes.

**Step 5** Set UseLocalTimeZone to TRUE.

**set UseLocalTimeZone TRUE**

```
Set UseLocalTimeZone TRUE
```

## Setting Up a Local Authentication Service

If you use the CAR server for authentication and authorization in your prepaid billing solution, you should configure an AA service. For example, you might configure a service similar to **local-users** (in the example configuration) for authentication and authorization of local users.

If some of the users are non-prepaid users or if the prepaid users need to have RADIUS authorization attributes returned, you should configure an AA service to perform that authentication and authorization.

If all of the users in a realm are prepaid users and the prepaid billing client does not require normal RADIUS authorization attributes, an AA service is not necessary.

**Step 1** Use `aregcmd` to set up a local authentication service.

```
cd /radius/services
```

```
add Prepaid-LocalAuthentication
```

```
Added prepaid-LocalAuthentication
```

```
cd prepaid-LocalAuthentication
```

```
[ //localhost/Radius/Services/prepaid-LocalAuthentication ]
  Name = prepaid-LocalAuthentication
  Description =
  Type =
```

**Step 2** Set the service type to local.

```
set type local
```

```
Set Type local
```

**Step 3** Set the `UserList` property to the userlist that contains IS835C prepaid users.

```
set UserList userlist_name
```

```
Set UserList userlist_name
```



**Note**

You can use an LDAP or ODBC service in place of the local authentication service.

## Setting Up a Prepaid Accounting Group Service

A prepaid billing solution usually requires a group service to tie together an AA service with a prepaid service, a group service to tie together an accounting service with a prepaid service, or both.

If you are using an AA service with your prepaid billing solution, you must configure a group service, for example **prepaid-users**, that ties the requests to the AA service (**local-users** in our example) with the prepaid service.

If you are using CAR for an accounting service with your prepaid billing solution, you must configure a group service, for example **prepaid-file**, that ties accounting requests to both the regular accounting service (**local-file** in our example) and the prepaid service.

**Step 1** Use `aregcmd` to create an accounting group service under `/Radius/Services`.

```
cd /radius/services
```

```
add Prepaid-Accounting
```

```
Added prepaid-accounting
```

**Step 2** Set the service type to group.

**cd prepaid-accounting**

```
[ //localhost/Radius/Services/prepaid-accounting ]
  Name = prepaid-accounting
  Description =
  Type =
```

**set type group**

```
Set Type group
```

The group service has the following properties:

```
[ //localhost/Radius/Services/prepaid-accounting ]
  Name = prepaid-accounting
  Description =
  Type = group
  IncomingScript~ =
  OutgoingScript~ =
  ResultRule = AND
  GroupServices/
```

**Step 3** Reference the Prepaid and Prepaid-LocalAccounting services under GroupServices.

**cd GroupServices**

```
[ //localhost/Radius/Services/prepaid-accounting/GroupServices ]
```

**add 1 prepaid**

```
Added 1
```

**add 2 prepaid-LocalFileAccounting**

```
Added 2
```

## Setting Up an Authentication Group Service

A prepaid billing solution usually requires a group service to tie together an AA service with a prepaid service, a group service to tie together an accounting service with a prepaid service, or both.

If you are using an AA service with your prepaid billing solution, you must configure a group service, for example **prepaid-users**, that ties the requests to the AA service with the prepaid service.

If you are using CAR for an accounting service with your prepaid billing solution, you must configure a group service, for example **prepaid-file**, that ties accounting requests to both the regular accounting service and the prepaid service.

**Step 1** Use **aregcmd** to add a prepaid authentication group service under **/Radius/Services**.

```
cd /radius/services
```

**add prepaid-groupAuthentication**

```
Added group-prepaidAuthentication
```

**cd group-prepaidAuthentication**

```
[ //localhost/Radius/Services/group-prepaidAuthentication ]
  Name = group-prepaidAuthentication
  Description =
  Type =
```

**Step 2** Set the service type to group.

**set type group**

```
Set Type group
```

The group service requires the ResultRule to be set to AND, the default setting for a group service.

**ls**

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

**Step 3** Change directory to GroupServices and add references to the prepaid service and the authentication service.

**cd GroupServices**

```
[ //localhost/Radius/Services/group-prepaidAuthentication/GroupServices ]
```

**add 1 Prepaid-LocalAuthentication**

```
Added 1
```

**add 2 prepaid**

```
Added 2
```

## Configuring CRB Prepaid Billing for SSG

In addition to the configuration described in [CRB Prepaid Billing, page 15-6](#), when using CRB-Prepaid billing with SSG, you must also perform the following:

- [Setting Up an Outgoing Script](#)
- [Setting Up an Incoming Script](#)

- [Setting Up a Prepaid Outgoing Script](#)
- [Add Prepaid Clients](#)

## Setting Up an Outgoing Script

---

**Step 1** Use `aregcmd` to add the **PCO-Parse-Client-Outgoing** outgoing script under **/Radius/Scripts**:

```
cd /radius/scripts
```

```
add PCO-Parse-Client-Outgoing
```

```
Added PCO-Parse-Client-Outgoing
```

```
cd PCO-Parse-Client-Outgoing
```

```
[ //localhost/Radius/Scripts/PCO-Parse-Client-Outgoing ]  
  Name = PCO-Parse-Client-Outgoing  
  Description =  
  Language =
```

**Step 2** Set the language to tcl.

```
set language tcl
```

```
Set Language tcl
```

**Step 3** Set the filename to **PCO-parse.client-outgoing.tcl**.

```
set filename PCO-parse.client-outgoing.tcl
```

```
Set Filename PCO-parse.client-outgoing.tcl
```

**Step 4** Set the EntryPoint to **PCO-parse-client-outgoing**.

```
set EntryPoint PCO-parse-client-outgoing
```

```
Set EntryPoint PCO-parse-client-outgoing
```

---

## Setting Up an Incoming Script

---

**Step 1** Use `aregcmd` to add the **PPI-Parse-Prepaid-Incoming** script under **/Radius/Scripts**.

```
cd /radius/scripts
```

```
add PPI-Parse-Prepaid-Incoming
```

**Step 2** Set the language to tcl.

```
cd PPI-Parse-Prepaid-Incoming
```

```
set language tcl
```

```
Set Language tcl
```

- Step 3** Set the filename to **PPI-Parse-Prepaid-Incoming.tcl**.

```
set filename PPI-Parse-Prepaid-Incoming.tcl
```

```
Set Filename PPI-Parse-Prepaid-Incoming.tcl
```

- Step 4** Set the EntryPoint to **PPO-Parse-Prepaid-Outgoing**.

```
set EntryPoint PPO-Parse-Prepaid-Outgoing
```

```
Set EntryPoint PPO-Parse-Prepaid-Outgoing
```

---

## Setting Up a Prepaid Outgoing Script

---

- Step 1** Use **aregcmd** to add the **PPO-Parse-Prepaid-Outgoing** outgoing script under **/Radius/Scripts**:

```
cd /radius/scripts
```

- Step 2** Add the **PPO-Parse-Prepaid-Outgoing** outgoing script under **/Radius/Scripts**.

```
cd /radius/scripts
```

```
add PPO-Parse-Prepaid-Outgoing
```

```
Added PPO-Parse-Prepaid-Outgoing
```

- Step 3** Set the language to tcl.

```
cd PPO-Parse-Prepaid-Outgoing
```

```
set language tcl
```

```
Set Language tcl
```

- Step 4** Set the filename to **PPO-Parse-Prepaid-Outgoing.tcl**.

```
set filename PPO-Parse-Prepaid-Outgoing.tcl
```

```
Set Filename PPO-Parse-Prepaid-Outgoing.tcl
```

- Step 5** Set the EntryPoint to **PPO-Parse-Prepaid-Outgoing**.

```
set EntryPoint PPO-Parse-Prepaid-Outgoing
```

```
Set EntryPoint PPO-Parse-Prepaid-Outgoing
```

---

## Add Prepaid Clients

---

- Step 1** Use **aregcmd** to add the prepaid clients under **/Radius/Clients**.

```
cd /radius/clients
```

```
add SSG
```

A RADIUS client has the following properties:

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

**Step 2** Set the `IPAddress` property to the client IP address.

```
set IPAddress aaa.bbb.ccc.ddd
```

```
Set IPAddress aaa.bbb.ccc.ddd
```

**Step 3** Set the `SharedSecret`.

```
set sharedsecret cisco
```

```
Set SharedSecret cisco
```

**Step 4** Set the `OutgoingScript` to **PCO-Parse-Client-Outgoing**.

```
set out PCO-Parse-Client-Outgoing
```

```
Set OutgoingScript PCO-Parse-Client-Outgoing
```

## Generic Call Flow

This section describes the generic call flow for the CAR CRB prepaid billing. The call flow is controlled by the AAA client. The CAR server converts VSAs into calls to the billing server.



### Note

For information about call flows and attributes for IS835C, see [IS835C Prepaid Billing, page 15-2](#).

The packet flows presented in [Figure 15-1](#) are specific to the CAR CRB prepaid billing only. The headlines in the packet flows are general and do not represent all data transferred. The letters **c**, **s**, and **b** in [Figure 15-1](#) designate the packet's source of **client**, **server**, or **billing server**, respectively.

**Figure 15-1**      **Generic Call Flow Diagram**



## Access-Request (Authentication)

**Flow 1c** shows the client sending the Access-Request to AAA server, part of a normal authentication request. The exact nature of the message contents is dictated by the access technology, be it be CDMA1X-RTT, GPRS, or another. The Access-Request might involve other messages such as PAP/CHAP or another form of authentication.

The **Flow 1c** Access-Request might contain a prepaid specific VSA, CRB\_AUTH\_REASON. [Table 15-4](#) lists the attributes included in the authentication Access-Request. This tells the CAR server to authenticate the subscriber with the Prepaid server as well. If the value is CRB\_AR\_INIT\_AUTHENTICATE, the initial quota must be obtained for a single service prepaid solution. If this VSA is not present, the CAR server will not authenticate with the Prepaid billing server.

**Table 15-4** *Attributes Sent During Subscriber Authentication*

Attribute Number	Attribute Name	Description	Notes
1	User-Name	APPL: Mobile Node Username	Required
2	NAS IP Address	Accounting Node IP Address	APPL: Required, POA
31	Calling-station-ID	APPL:MSISDN or IMSI	APPL: Conditional
26, 9	CRB_AUTH_REASON CRB_AR_INIT_AUTHENTICATE	See VSA section	Required
26, 9	CRB_USER_ID	APPL:PDSN address or SSG address	APPL: Required, Address of the PDSN
26, 9	CRB_SERVICE_ID	APPL: Service ID such as Simple IP service, Mobile IP service, or VPN service	
26, 9	CRB_SESSION_ID	This VSA contains the session key ID information	Required; the session ID must be globally unique across all clients and across reboots of the client

In **Flow 1s**, the CAR server sends a call to the billing server to authenticate the prepaid user and possibly determine more information about the subscriber's account. The CAR server can be configured to generate this packet flow, using a subscriber profile parameter, if the request is from a prepaid subscriber.

## Access-Accept (Authentication)

**Flow 2b** shows the billing server returning the authentication result. The billing server returns a failure if the prepaid subscriber has an inadequate balance.

**Flow 2s** shows the CAR server sending the Access-Accept to the AAA client. This message flow contains at least one prepaid billing-specific VSA (listed in [Table 15-5](#)) and might contain other access technology-specific attributes.

**Table 15-5** *Attributes Sent to AAA client in Access-Accept (Authentication)*

Attribute Number	Attribute Name	Description	Notes
26, 9	CRB__USER_TYPE CRB_AR_INIT_AUTHENTICATE	See <a href="#">Vendor-Specific Attributes, page 15-23</a>	Optional

## Access-Request (Authorization)

In **Flow 3c**, the AAA client sends another Access-Request, this time to authorize the subscriber.

[Table 15-6](#) lists the attributes required by the CAR server to authorize the subscriber. The session key ID used must be specified using a prepaid VSA pointing to the RADIUS attribute (standard or VSA).

**Table 15-6** *Attributes Sent During Subscriber Authorization*

Attribute Number	Attribute Name	Description	Notes
1	User-Name	APPL: Mobile Node Username	Required
2	NAS IP Address	Accounting Node IP Address	APPL: Required, POA
31	Calling-station-ID	APPL:MSISDN or IMSI	APPL: Conditional
26, 9	CRB_AUTH_REASON CRB_AR_INIT_AUTHORIZE	See <a href="#">Vendor-Specific Attributes, page 15-23</a>	Required
26, 9	CRB_USER_ID	APPL:PDSN address or SSG address	APPL: Required, Address of the PDSN
26, 9	CRB_SERVICE_ID	APPL: Service ID such as Simple IP service, Mobile IP service, or VPN service	
26, 9	CRB_SESSION_ID	This VSA contains the session key ID information	Required; the session ID must be globally unique across all clients and across reboots of the client

In **Flow 3s**, the CAR server sends the Prepaid billing server to obtain a quota. The quota might contain several values depending on the number of measurement parameters chosen.

## Access-Accept (Authorization)

**Flow 4b** shows the billing server returning the quota array for the subscriber.

In **Flow 4s**, the CAR server converts the quota array received into VSAs and sends an Access-Accept with the assembled VSAs to the AAA client. [Table 15-7](#) lists the prepaid-specific VSAs that might be included in the Access-Accept response message sent to the AAA client. For more detailed information about the VSAs, see [Vendor-Specific Attributes, page 15-23](#).

**Table 15-7** *Attributes Sent to AAA client in Access-Accept (Authorization)*

Attribute Number	Attribute Name
26, 9	CRB_DURATION
26, 9	CRB_TOTAL_VOLUME
26, 9	CRB_UPLINK_VOLUME
26, 9	CRB_DOWNLINK_VOLUME
26, 9	CRB_TOTAL_PACKETS
26, 9	CRB_UPLINK_PACKETS
26, 9	CRB_DOWNLINK_PACKETS

Flows **3c** through **4s** are repeated for every service started or restarted by the AAA client.

However, if the return parameters indicate that the authorization is rejected, an Access-Accept message is generated and sent to the client as shown in [Table 15-8](#). When this type of error condition occurs, no other VSA is included in the Access-Accept message.

**Table 15-8** Attribute Sent to Report Error Condition to AAA client

Attribute Number	Attribute Name	Description	Notes
26, 9	CRB_TERMINATE_CAUSE	Identifies why a subscriber failed authentication: 1. Exceeded the balance 2. Exceeded the overdraft 3. Bad credit 4. Services suspended 5. Invalid User	Conditional; rejection might be returned with Access-Accept and zero (0) quota

## Accounting-Start

In **Flow 5c**, the AAA client sends the Accounting-Start. In **Flow 6s**, the CAR server replies with the Accounting-Response.

## Data Flow

At this point, the data transfer begins. The AAA client monitors the subscriber's allocated quotas for metering parameters. A subscriber's Reauthorization request is generated when a quota for at least one of the metering parameters, is depleted.

## Access-Request (Quota Depleted)

**Flow 7c** shows the client sending an Access-Request to the CAR server because at least one quota has been depleted. The Access-Request includes different measurements of how much of the quotas were used in VSA format. This enables the billing server to account for the usage and manage the subscriber's balance before assigning a new quota. [Table 15-9](#) lists the attributes returned to the CAR server:

**Table 15-9** Attributes Sent by NAS When Quota Depleted

Attribute Number	Attribute Name	Description	Notes
1	User-Name	APPL: Mobile Node Username	Conditional
2	NAS IP Address	Accounting Node IP Address	APPL: Required, POA address, or Home Node address
31	Calling-station-ID	APPL:MSISDN or IMSI	APPL: Conditional
26, 9	CRB_AUTH_REASON	See VSA	Required
26, 9	CRB_USER_ID	APPL: PDSN address or SSG address	APPL: Required, address of SGSN

**Table 15-9** *Attributes Sent by NAS When Quota Depleted (continued)*

Attribute Number	Attribute Name	Description	Notes
26, 9	CRB_DURATION	See <a href="#">Vendor-Specific Attributes</a> , page 15-23	Required
26, 9	CRB_TOTAL_VOLUME		Conditional
26, 9	CRB_UPLINK_VOLUME		
26, 9	CRB_DOWNLINK_VOLUME		
26, 9	CRB_TOTAL_PACKETS		
26, 9	CRB_UPLINK_PACKETS		
26, 9	CRB_DOWNLINK_PACKETS		

## Accept-Accept (Quota Depleted)

**Flow 7s** shows the CAR server returning the used quota array to the billing server. The call includes `aaa_ebs_reauthoriz()`. The billing server sends an updated quota array for the next period to the CAR server.

In **Flow 8s**, the CAR server converts the quota array into VSAs and sends them to the AAA client.

**Table 15-10** *Attributes Sent to AAA Client in Access-Accept (Reauthorization)*

Attribute Number	Attribute Name
26, 9	CRB_USER_TYPE
26, 9	CRB_DURATION
26, 9	CRB_TOTAL_VOLUME
26, 9	CRB_UPLINK_VOLUME
26, 9	CRB_DOWNLINK_VOLUME
26, 9	CRB_TOTAL_PACKETS
26, 9	CRB_UPLINK_PACKETS
26, 9	CRB_DOWNLINK_PACKETS

## Accounting Stop (Session End)

In **Flow 9c**, the client sends an Accounting-Stop to the CAR server to end the session. The Accounting-Stop message includes an updated quota array with the usage adjustments since the previous authorization in the VSA form.

[Table 15-11](#) lists the attributes included in the Accounting-Stop message set to the CAR server and forwarded to the billing server.

## Accounting Response (Final Status)

In **Flow 9s**, the CAR server sends the used quota array to the billing server in an Accounting-Stop message. Any values returned by the billing server in **Flow 10b** are discarded.

**Flow 10s** shows the CAR server sending final Accounting-Response message to the AAA client.

**Table 15-11** Attributes Sent in Accounting-Stop Message

Attribute Number	Attribute Name	Description	Notes
1	User-Name	APPL: Mobile Node Username	Conditional
2	NAS IP Address	Accounting Node IP Address	APPL: Required, POA
31	Calling-station-ID	APPL:MSISDN or IMSI	APPL: Conditional
40, 2	Acct_status_type	Indicates the accounting “Stop” for the service	Required; this value (2) indicates an Accounting-Stop request message
42	Acct-Input-Octets	The number of octets sent by the subscriber; uplink	Required
43	Acc_Output-Octets	The number of octets received by the subscriber; downlink	
46	Acct-Session-Time	Duration of the session	
47	Acct-Input-Packets	Number of packets sent by the subscriber	
48	Acct-Output-Packets	Number of packets received by the subscriber	
49	Acct-Terminate-Cause	This parameter, used for tracking, should remain the same for all accounting requests for a given service.	
26, 9	CRB_DURATION	See <a href="#">Vendor-Specific Attributes, page 15-23</a>	Conditional
26, 9	CRB_TOTAL_VOLUME		
26, 9	CRB_UPLINK_VOLUME		
26, 9	CRB_DOWNLINK_VOLUME		
26, 9	CRB_TOTAL_PACKETS		
26, 9	CRB_UPLINK_PACKETS		
26, 9	CRB_DOWNLINK_PACKETS		
26, 9	CRB_SESSION_ID	Specifies the RADIUS attribute carrying the session ID information	Optional

## Vendor-Specific Attributes

Vendor-specific attributes are included in specific RADIUS packets to communicate prepaid user balance information from the CAR server to the AAA client, and actual usage, either interim or total, between the NAS and the CAR Server.

Table 15-12 lists the VSAs that will be defined in the API. Table 15-12 also lists the string to be used with Cisco-AVPair below the VSA.


**Note**

VSAs that start with CRB are used for Cisco Radius Billing prepaid service.

**Table 15-12 Vendor-Specific Attributes for the Cisco Prepaid Billing Solution**

VSA Name	Type	Source (Call Flow)	Description
CRB_AUTH_REASON crb-auth-reason	Int8	1c, 7c, 7'c	Passed with re-authorization: 1. Initial Authentication 2. Initial Authorization 3. Re-authorization 4. Return Quota 5. Query to EBS
CRB_USER_ID crb-user-id	String	1c, 7c, 7'c	APPL: In PDSN this can be Address of the PDSN.
CRB_SERVICE_ID crb-service-id	String	1c, 7c	Identifies the subscriber's service
CRB_USER_TYPE crb-entity-type	Int8	4s	Type of user: 1. Prepaid user 2. Post-paid with no credit limit 3. Post-paid with credit limit 4. Invalid user  The source for this VSA value could be from the Subscriber profile or from the billing server

**Table 15-12 Vendor-Specific Attributes for the Cisco Prepaid Billing Solution (continued)**

<b>VSA Name</b>	<b>Type</b>	<b>Source (Call Flow)</b>	<b>Description</b>
CRB_DURATION crb-duration	Int32	4s, 8s	Downlink quota received by the AAA client
CRB_TOTAL_VOLUME crb-total-volume			Total Volume quota received by the AAA client
CRB_UPLINK_VOLUME crb-uplink-volume			Uplink volume quota received by the AAA client
CRB_DOWNLINK_VOLUME crb-downlink-volume			Uplink Volume quota received by the AAA client
CRB_TOTAL_PACKETS crb-total-packets			Downlink Packet quota received by the AAA client
CRB_UPLINK_PACKETS crb-uplink-packets			Uplink Packet quota received by the AAA client
CRB_DOWNLINK_PACKETS crb-downlink-packets			Uplink Volume quota received by the AAA client
CRB_SESSION_ID crb-session-id			String

**Table 15-12 Vendor-Specific Attributes for the Cisco Prepaid Billing Solution (continued)**

VSA Name	Type	Source (Call Flow)	Description
CRB_TERMINATE_CAUSE crb-terminate-cause	Int8	4se	Identifies why a subscriber failed authentication: <ol style="list-style-type: none"> <li>1. Exceeded the balance</li> <li>2. Exceeded the overdraft</li> <li>3. Bad credit</li> <li>4. Services suspended</li> <li>5. Invalid User</li> <li>6. Invalid Password</li> <li>7. System Error</li> <li>8. Disabled</li> <li>9. Expired</li> <li>10. Valid in Future</li> <li>11. Used up</li> <li>12. No Parallel sessions</li> <li>13. Session Already closed</li> <li>14. Invalid session</li> </ol>
CRB_PRIVATE crb-private	String	n/a	Reserved for future use

## Implementing the Prepaid Billing API

A shared library must implement the API functions to perform the various tasks given in the description of each of the function. This needs to be compiled as a shared library and then specified as part of the remote server configuration at the Filename property. See [Setting Up a Prepaid Billing RemoteServer, page 15-2](#) or [Setting Up a Prepaid Billing RemoteServer, page 15-7](#).

At startup, CAR loads the library dynamically and registers the API functions, then calls out the library initialization API once at startup. The call to initialize functions initializes various data structures and connections with the billing server, as required.



### Note

Cisco works with you to develop the prepaid billing service and implement the API. For more information, contact your Cisco systems engineer.

At various times, according to the call flow described in the Prepaid Call Flow Specification (CRB or IS835C), CAR calls out appropriate API functions present in the shared library. The values for the arguments passed to these API calls are purely derived from the incoming RADIUS packet and CAR does not maintain any dynamic information related to the call flow. It is up to the API function to make use of the information passed to it as C structures to contact the Billing server, get appropriate data, and return the same to CAR using the designated arguments.



### Note

See the API specifications for more details pertaining to the arguments and return values of the API.

