

Contents

- [1 Conference Call](#)
- [2 Description](#)
- [3 Use Cases](#)
 - ◆ [3.1 Billing \(Pre-pay\)](#)
 - ◆ [3.2 Billing \(Post-pay\)](#)
 - ◆ [3.3 Pay-Per-Call Service Billing](#)
 - ◆ [3.4 Maximum Credit and/or Fraud Prevention](#)
- [4 Design Goals](#)
- [5 Status](#)
- [6 Installation and configuration](#)
 - ◆ [6.1 Database Tables](#)
- [7 Creating the database table for PostgreSQL](#)
- [8 Creating the database table for MySQL](#)
- [9 Billing a Call](#)
 - ◆ [9.1 Nibble Method \(Default\)](#)
 - ◆ [9.2 Alternative to Nibble Billings](#)
- [10 Examples](#)
 - ◆ [10.1 Different Rates per User](#)
 - ◆ [10.2 Single Rate for all Users](#)
 - ◆ [10.3 Different Rates per Area Code](#)
 - ◆ [10.4 Different Rates per Service Delivery](#)
 - ◆ [10.5 Hangup the Call When the Balance Is Depleted](#)
- [11 Application / CLI / API Commands](#)
 - ◆ [11.1 Check](#)
 - ◆ [11.2 Flush](#)
 - ◆ [11.3 Pause](#)
 - ◆ [11.4 Resume](#)
 - ◆ [11.5 Reset](#)
 - ◆ [11.6 Adding/Deducting funds](#)
 - ◆ [11.7 Enabling Session Heartbeat](#)
 - ◆ [11.8 Bill base on B leg Only](#)
- [12 Future Goals](#)
- [13 Other Notes](#)
- [14 FAQs](#)

Conference Call

Darren Schreiber talks about mod_nibblebill during one of our weekly conference calls. If you are using mod_nibblebill you should [listen](#) to the conference call. It will answer some of your questions.

Description

mod_nibblebill is a credit/debit module for FreeSWITCH. The module was initially written by Darren Schreiber to fill the gaps of a professional grade trunking system that lacked the ability to detect fraud real-time. Its purpose is to allow real-time debiting of credit or cash from a database while calls are in progress. I had the following goals:

- Debit credit/cash from accounts real-time
- Allow for billing at different rates during a single call
- Allow for warning callers when their balance is low (via audio, in-channel)
- Allow for disconnecting or re-routing calls when balance is depleted
- Allow billing functions listed above to operate with multiple concurrent calls

Use Cases

mod_nibblebill can be used in a variety of use cases, some of which are listed below.

Billing (Pre-pay)

You can allow people to put cash into an account and "nibble" away at it. In addition, when callers have almost depleted their account, a tone or other message can play (or another action can occur) warning the caller of this.

Upon full depletion of their account, the call can either be transferred to an extension that allows them to recharge their balance via touch-tones or otherwise, or the call can simply be disconnected.

Billing (Post-pay)

If your database column allows it, you can make the warning and out-of-cash thresholds a negative number. In this manner, callers can "dip into" negative numbers in the database, and then you can bill them after their usage. In this way, you are also able to protect yourself from abuse, since callers will still be terminated if they go below some (negative) threshold you set (i.e. spent too much money in a month).

This is a more typical approach to billing for landlines and it allows for an account to automatically be cut-off if excessive usage occurs without someone paying their bill.

Pay-Per-Call Service Billing

You could bill for providing a special service, either via fixed fee or via per-minute after a certain event (entering a credit card number and being approved, for example).

Maximum Credit and/or Fraud Prevention

You can setup a credit field that gets depleted by your users, similar to pre-pay above, but just not tell them about it. When they deplete all their credit for a day/week/month/etc. they can't make any more calls. You can use an external script to deposit more credit into their account at a pre-set interval.

This would allow something like "100 minutes a day free" or other such promotions to work.

Design Goals

- Concurrent design - allows for supervision of multiple in-progress channels that belong to the same account/account code
- Scalability - allow for different heartbeat intervals (or turning off supervision during calls altogether). This allows the administrator to tweak checks depending on system load
- Flexibility - allow warning levels and "out-of-funds" levels to be flexible on a global and per-user basis, and allow customization as to what happens when the caller is out of funds
- Customizable - all settings should be customizable, including when people are terminated or warned and what happens when they are terminated or warned

Status

I am still finishing mod_nibblebill, but it does function right now. You can use it today and it should be stable.

Many features (including a few listed in this wiki) are not done yet. If you have any questions, ask pyite in IRC or email d@d-man.org.

Installation and configuration

mod_nibblebill is now part of the main FreeSWITCH source tree. It requires ODBC support to function properly.

A note about ODBC...

FreeSWITCH now includes ODBC support by default if it finds your UnixODBC libraries during compilation. For more info about UnixODBC look here: [mod_spidermonkey_odbc](#)

To install mod_nibblebill:

- Uncomment applications/mod_nibblebill in modules.conf (in the source tree)
- Recompile FreeSWITCH with ODBC support. For more information on compiling FreeSWITCH with core ODBC support, see [Using ODBC in the core](#)
- Modify the nibblebill settings in /usr/local/freeswitch/conf/autoload_configs/nibblebill.conf.xml . The settings should be relatively obvious. **You must edit this file with your database information**

A sample dsn for a postgresSQL database would be as follows: `<param name="db_dsn" value="pgsql://hostaddr=serverip dbname=dbname user=uname password="/>`

Mod_nibblebill

- Enable mod_nibblebill in your FreeSWITCH installation by adding it to /usr/local/freeswitch/conf/autoload_configs/modules.conf.xml like so:

```
<load module="mod_nibblebill"/>
```

- Start/restart FreeSWITCH

Note that you can also load/unload mod_nibblebill from the FreeSWITCH CLI. Just type "load mod_nibblebill" or "unload mod_nibblebill"

Database Tables

Per your configuration file (see nibblebill.conf.xml), make sure you have an ODBC database driver and database that is accessible and contains the correct database, table and column names.

A sample table is below

```
mysql> use tcapi;
mysql> select * from accounts;
+-----+-----+-----+
| id    | name      | cash  |
+-----+-----+-----+
| 1     | Darren    | 41.4161 |
| 2     | Joe       | 50      |
| 9     | tester9   | 50      |
| 10    | tester10  | 44.8213 |
| 837269 | My Company | 50      |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

In the above example, a table named "accounts" exists in the database "tcapi". That table contains id and cash columns for use by the billing script. id represents the account code, and cash represents the amount of money the user is allowed to spend. The corresponding settings in your nibblebill.conf.xml file for the above setup would be:

```
<param name="odbc_dsn" value="pgsql://db_ip/db_name"/>
<param name="db_table" value="accounts"/>
<param name="db_column_cash" value="cash"/>
<param name="db_column_account" value="id"/>
```

Creating the database table for PostgreSQL

```
create table accounts (
  id bigserial not null,
  name varchar( 256 ),
  cash double precision not null
);
```

Creating the database table for MySQL

```
CREATE TABLE accounts
(
  id int NOT NULL PRIMARY KEY,
  name VARCHAR(255),
  cash double precision NOT NULL
);
```

Billing a Call

Nibble Method (Default)

The default method of billing is based on the concept of a FreeSWITCH heartbeat. For every X seconds, we deduct Y amount from an account.

To bill a call, you must set a minimum of two variables on an in-progress channel. The variables are `nibble_rate` and `nibble_account`. As a neat feature, `mod_nibblebill` doesn't really care where you setup the billing variables from, as long as they exist before a hangup occurs. That means you can set them in the dialplan, in the directory, in a Lua script - wherever.

In it's simplest form, you can add this to a user's directory entry:

```
<variable name="nibble_rate" value="0.03"/>
<variable name="nibble_account" value="18238"/>
```

Now that user will be billed \$0.03/minute for every call made or received. The billing will go against account 18238.

By default, a heartbeat is set at 60 seconds. This means that every 60 seconds, 0.03 is deducted from their account. Note that all mathematical calculations are done using FreeSWITCH's internal microseconds counters. This means a few things:

1. If a heartbeat does not fire exactly on-time you will get a fraction of a cent billed. You should make sure your underlying database can support that.
2. Counters count the time in between ticks exactly. There is no "lost" billing.
3. Billing of minimums does not exist (yet)

You can modify the heartbeat interval globally with this parameter:

```
<param name="global_heartbeat" value="300">
```

That would make the heartbeat fire every 300 seconds, or every 5 minutes. Heartbeats can go as low as every second (though this is really not wise, as you're making a database call every second, per channel).

You can set wanted billing increment in seconds using `"nibble_increment"` variable, the logic is following

```
if time < increment {
  billing = increment
} else {
```

Mod_nibblebill

```
    billing = ceil(time/increment) * increment
}
```

to set billing cost rounded to 30 seconds set variable like:

```
<variable name="nibble_increment" value="30" />
```

Alternative to Nibble Billings

It is possible to use this module without heartbeats enabled. That means you just bill a call at the end of the call. You set the same variables as listed above, but you also set one additional variable in your mod_nibblebill.conf.xml file:

```
<param name="global_heartbeat" value="off">
```

By doing this, billing will only occur at the end of a call (on hangup). The time calculation will be from when the call was answered until the end of the call. If a call is never answered, billing is skipped.

The formula used to bill calls when this parameter is set is:

$[\text{time call ended}] - [\text{time call answered}] \times [\text{rate per minute}] = \text{\$total bill rate}$

NOTE: This method does not allow for any supervision of a call in progress, meaning fraud can occur and people can go over their allotted limits.

Examples

Different Rates per User

It is possible to have different rates per minute, per user. This can work IN ADDITION to the dialplan examples listed below, as long as you don't clobber the variables. Here's an example.

Let's say you have two users - one is billed at \$0.05/minute and one at \$0.10/minute. Neither is billed when calling an 800 number. You would setup their directory entries like this:

```
<user id="dschreiber">
  <params>
    <param name="password" value="1234"/>
  </params>
  <variables>
    <variable name="nibble_rate" value="0.05"/>
    <variable name="nibble_account" value="8182"/>
    <variable name="default_areacode" value="415"/>
    <variable name="toll_allow" value="domestic,international,local"/>
    <variable name="user_context" value="default"/>
  </variables>
</user>

<user id="expensive_guy">
  <params>
```

Mod_nibblebill

```
<param name="password" value="1234"/>
</params>
<variables>
  <variable name="nibble_rate" value="0.10"/>
  <variable name="nibble_account" value="2932"/>
  <variable name="default_areacode" value="212"/>
  <variable name="toll_allow" value="domestic,international,local"/>
  <variable name="user_context" value="default"/>
</variables>
</user>
```

Then in your dialplan, override the bill rates for toll-free calls only:

```
<extension name="tollfree800">
  <condition field="destination_number" expression="^1{0,1}(800\d{7})$">
    <action application="set" data="nibble_rate=0"/>
    <action application="bridge" data="sofia/gateway/bandwidth.com/$1"/>
  </condition>
</extension>
```

All non-800 number calls will be billed at the rates set on the user's account, while toll-free calls will be billed 0 (equivalent to no billing).

Single Rate for all Users

On your user accounts:

```
<include>
  <user id="diegoviola">
    <params>
      <param name="password" value="1234"/>
    </params>
    <variables>
      <variable name="toll_allow" value="domestic,international,local"/>
      <variable name="user_context" value="default"/>
      <variable name="nibble_account" value="1"/>
    </variables>
  </user>
</include>

<include>
  <user id="dschreiber">
    <params>
      <param name="password" value="1234"/>
    </params>
    <variables>
      <variable name="toll_allow" value="domestic,international,local"/>
      <variable name="user_context" value="default"/>
      <variable name="nibble_account" value="2"/>
    </variables>
  </user>
</include>
```

Mod_nibblebill

On the extension you want to bill:

```
<extension name="outbound">
  <condition field="destination_number" expression="^9(\d{10,})$">
    <action application="set" data="nibble_rate=0.05"/>
    <action application="set" data="${nibble_account}"/>
    <action application="bridge" data="sofia/gateway/telias/$1"/>
  </condition>
</extension>
```

Different Rates per Area Code

This example bills all calls at \$0.05/minute, except calls to area code 919 which are \$0.07/minute and calls to 800 numbers, which are free. Calls are billed to whatever account code is set for the user in their directory profile.

Note: In this example I set the rate from the dialplan. Be careful! This overrides any variable set on the user/directory level!

```
<extension name="tollfree800">
  <condition field="destination_number" expression="^1{0,1}(800\d{7})$">
    <action application="set" data="nibble_account=${accountcode}"/>
    <action application="set" data="nibble_rate=0"/>
    <action application="bridge" data="sofia/gateway/bandwidth.com/$1"/>
  </condition>
</extension>

<extension name="special919rate">
  <condition field="destination_number" expression="^1{0,1}(919\d{7})$">
    <action application="set" data="nibble_account=${accountcode}"/>
    <action application="set" data="nibble_rate=0.07"/>
    <action application="bridge" data="sofia/gateway/bandwidth.com/$1"/>
  </condition>
</extension>

<extension name="domestic">
  <condition field="destination_number" expression="^(1{0,1}\d{10})$">
    <action application="set" data="nibble_account=${accountcode}"/>
    <action application="set" data="nibble_rate=0.05"/>
    <action application="bridge" data="sofia/gateway/bandwidth.com/$1"/>
  </condition>
</extension>
```

Different Rates per Service Delivery

This is kind of neat, and not yet baked. But it can be used today. There will be more advances with this.

This idea encompasses the concept of changing the nibble_rate while the call is in progress. **WARNING!!!** The current "catch" is that you really want to flush the current call's billings to the database before the call's rate changes. This is to write out any billed seconds since the last query to DB with the old rate.

Mod_nibblebill

In the meantime, here's the idea... A caller could call-in and for the first part of their call they might be getting billed at \$1.00/minute, maybe to talk to Tier 1 support. If they need Tier 2 support, the rate goes to \$5.00/minute. The rate changes when the call is transferred, simply by changing the variable. You can even set the amount to 0 while the caller is on hold or in a FIFO queue.

```
<extension name="tier1">
  <condition field="destination_number" expression="^2000$">
    <!-- Save anything billed at a previous rate -->
    <action application="nibblebill" data="flush"/>
    <!-- Change the rate -->
    <action application="set" data="nibble_rate=1.00"/>
    <!-- Transfer to Tier1 rep -->
    <action application="transfer" data="1000 XML default"/>
  </condition>
</extension>
```

```
<extension name="tier2">
  <condition field="destination_number" expression="^2000$">
    <!-- Save anything billed at a previous rate -->
    <action application="nibblebill" data="flush"/>
    <!-- Change the rate -->
    <action application="set" data="nibble_rate=5.00"/>
    <!-- Transfer to Tier2 rep -->
    <action application="transfer" data="1001 XML default"/>
  </condition>
</extension>
```

Another possible use of this is to bill a caller while they're talking to support, but to stop billing after the call when you give them a survey. Same concept as above, but done this way:

```
<extension name="survey-after-call">
  <condition field="destination_number" expression="^2000$">

    <!-- Handle support request here at $1.00/minute via extension 1001 -->
    <action application="set" data="nibble_rate=1.00"/>
    <action application="set" data="hangup_after_bridge=false"/>
    <action application="bridge" data="sofia/internal/1001@${domain}"/>
    <action application="nibblebill" data="flush"/>

    <!-- Set rate to 0, then transfer caller to the survey IVR -->
    <action application="set" data="nibble_rate=0.00"/>
    <action application="bridge" data="sofia/internal/1002@${domain}"/>
  </condition>
</extension>
```

Hangup the Call When the Balance Is Depleted

When the balance of an account drops below the setting you have specified in the configuration for "noba_amt", the call gets transferred to an extension of your choice. This allows you to play a message such as "Your call has been terminated due to insufficient funds." Since we're really just transferring the call to an extension and suspending billing, you could get fancy and potentially make the user key in their credit card number to re-up their funds.

In your conf/autoload_configs/nibblebill.conf.xml add something like this:

```
<param name="noba_amt" value="0"/>
```

Mod_nibblebill

```
<param name="nobal_action" value="hangup XML default"/>
```

In this example, note the nobal_action of "hangup XML default". This tells mod_nibblebill to transfer the call to the extension named "hangup" in the default context of your XML dialplan when the balance reaches the nobal_amt threshold. You can then add this to your dialplan:

```
<extension name="hangup">
  <condition field="destination_number" expression="^(hangup)$">
    <action application="playback" data="no_more_funds.wav"/>
    <action application="hangup"/>
  </condition>
</extension>
```

In this example, when a caller's balance reaches zero their call will be transferred to the hangup extension. That extension will play a message stating that they are out of funds (assuming you record a sound file name no_more_funds.wav) and the call will disconnect.

Note carefully that the B leg currently also gets transferred to the same extension. If you don't like this behaviour, tell me, and I'll make a new flag :).

Application / CLI / API Commands

The following commands can be used from the dialplan, CLI or API. The syntax is basically the same for each, with the somewhat obvious difference being that applications are in the format:

```
<action application="nibblebill" data="action [params]">
```

Where as CLI and API commands are just:

```
nibblebill <channel-uuid> <action> [params]
```

Check

Inserting this in your application or using it on the CLI with a UUID returns the balance that has been billed so far. This does not include any increments not yet written to the database.

```
<action application="nibblebill" data="check"/>
```

Flush

Inserting this in your dialplan:

```
<action application="nibblebill" data="flush"/>
```

...will immediately write to the database any pending billings. Billing will continue, but everything that needed to be billed up to this point in time will be calculated and recorded.

This has no effect when billing is paused.

Pause

Inserting this in your dialplan:

```
<action application="nibblebill" data="pause"/>
```

...will set a flag to pause billing. If the call is terminated while billing is paused, no billing from the time the call was paused onward will be calculated, but billing prior to the pause will still get recorded. You can also manually resume billing later on during the call with the resume command (see below).

Note that if you call the pause command when a call is already paused, the call is ignored.

Resume

Inserting this in your dialplan:

```
<action application="nibblebill" data="resume"/>
```

...will resume billing during a call that was previously paused. The time in between pause and resume is not billed. Note that you can pause and resume a call multiple times and the amount of time in between each pause period will be tracked.

Reset

Inserting this in your dialplan:

```
<action application="nibblebill" data="reset"/>
```

...resets the billing timer to the current time. Note that all you are doing here is resetting all the internal counters that track the call's progress to the current time, so any time that would have been billed prior to now (but has not yet been committed to disk) will be "lost" and considered "free."

Any amounts already deducted in the database for a particular account are considered committed - i.e. a done deal. This command has no impact on commits already made to disk.

Adding/Deducting funds

Inserting this in your dialplan:

```
<action application="nibblebill" data="adjust 5.00"/>
```

...adds or deducts a certain amount of funds from an account (in this case, we're adding \$5.00). Note that this occurs immediately and currently circumvents any protections that exist for when the database is down. It is your responsibility to deal with having a functioning database when you use this command.

Use negative numbers to deduct from an account.

Enabling Session Heartbeat

Enabling the session heartbeat is done during a call as follows:

```
<action application="nibblebill" data="heartbeat 60"/>
```

This sets the heartbeat for the current call (only) to 60 seconds. You can set this differently per-call.

Bill base on B leg Only

If you want to bill only B-Leg enable_heartbeat_events variable must be enable:

```
<action application="bridge" data="{enable_heartbeat_events=5,nibble_rate=1,nibble_account=083883
```

Future Goals

The following things are still in the works:

1. Add ability to warn callers when their balance is low (via audio, in-channel).
2. Add ability to transfer/terminate a caller if the call goes beyond a certain dollar amount.
3. Add ability to have verbose logging, which writes to DB every time a deduction is made from an account.
4. Rounding up, so we don't do fractions of a cent, on call completion (not during the call though).
5. Consolidate various functions - there is a lot of repeat code in there that can be shrunk down.
6. Make error handling for database, which inherently adds support for no database, such that when the database is down (or not installed) just log to a text file.
7. Add buffering abilities, to reduce the number of database calls.

Other Notes

At the end of a call, the module sets a variable named nibble_total_billed. You can use mod_cdr to record this variable to your CDR log. This is helpful for comparing the amount a customer was billed with the actual call activity later on, in the event of a dispute.

When operating in bypass_media mode, you can't get the scheduler to fire more often than once every 60 seconds. Not sure why - working on finding out. This might cause calls to continue past their threshold.

FAQs

- Q: Can you bill based on the B-Leg and not the A-Leg?
 - ◆ Yes, you can. You need to set the billing variables on your outbound calling leg and NOT on your A-Leg. You can turn on the heartbeat on your outbound leg but you don't have to - it will bill at the end of the call automatically if the variables are set.

Mod_nibblebill

- Q: Can you bill based on a multi-call B-Leg, where you have, say a conference call with multiple people on it?
 - ◆ Yes, see answer above re: B-Leg and not A-Leg.

- Q: When do you start billing?
 - ◆ Billing starts from the time the A leg is considered answered. This can cause some issues when you don't properly handle early media and such and a call is considered "answered" by FreeSWITCH even when the other end hasn't picked up yet. The current workaround for this is to reset billing when the call is answered and not have a heartbeat set until after the call is answered.

- Q: Can you bill on both the A-Leg and the B-leg with different rates to different accounts?

Yes you can. I have an example dialplan entry here:

```
<extension name="Internal-XXX_Mobile">
  <condition field="destination_number" expression="^(1\d+)$">
    <action application="set" data="hangup_after_bridge=true"/>
    <action application="set" data="nibble_account=9999"/>
    <action application="set" data="nibble_rate=0.05"/>
    <action application="export" data="nolocal:nibble_account=1111"/>
    <action application="export" data="nolocal:nibble_rate=0.03"/>
    <action application="bridge" data="sofia/external/$1@10.0.0.10"/>
    <action application="hangup"/>
  </condition>
</extension>
```

Notice A-Leg is billed and rated with the "set" function and the B-Leg is billed and rated with the "export" function. FYI: This works with CDR-CSV account codes as well, if you enable billing "ab" call legs.