Contents

- 1 Introduction
- 2 Setup notes
- 3 API/Event Socket
- 4 Dialplan
  - 4.1 Dialplan Examples
- 5 Field Definitions
  - 5.1 destnum
  - 5.2 dialstring
  - 5.3 group
  - 5.4 limit
  - 5.5 acctcode
- 6 ODBC/Database Setup
  - 6.1 FreeSWITCH
    - 6.1.1 easyroute.conf.xml
  - 6.2 Custom Query
  - 6.3 MySQL Sample Setup
  - 6.4 PostgreSQL Sample Setup
  - 6.5 SQLite Sample Setup

Introduction

Easyroute is a simple DID routing engine that uses a database lookup to determine how to route an incoming call. It uses a simple principle of numbers and gateways, with a many-to-one relationship. The lookup is performed by submitting the DID number to the database and getting a set of data fields in return. The database is supplied by the user. Easyroute can be called from the dialplan or from the API/Event Socket interface.

Setup notes

- Enable mod_easyroute in trunk/modules.conf by uncommenting the line: #applications/mod_easyroute before building
- Enable ODBC in the core when running configure: ./configure --enable-core-odbc-support
- Enable the module in the config, edit modules.conf.xml, add the line: <load module="mod_easyroute"/>

API/Event Socket

The syntax for the easyroute API is:

```
easyroute <phonenumber> [field]
```

The field can be one of the five fields returned from the database. If the field is not specified then the following channel variables are set:

- easy_destnum
- easy_dialstring
- easy_group
- easy_limit
- easy_acctcode

From there your application can use the variables for routing, processing, etc.

Dialplan

The syntax for easyroute in the dialplan:

```
<action application="easyroute" data="[phone number]"/>
```

or,

```
<action application="set" data="chan_var={easyroute([phone number] [data field])}"/>
```

Dialplan Examples

```
<!-- This example will test the limit value for the DID number being checked -->
<!-- If over limit, send 503 -->
<extension name="easyroute">
        <condition field="destination_number" expression="^1?(\d{10})$" break="on-true">
                <action application="easyroute" data="$1"/>
                <action application="limit" data="db easyroute ${easy_group} ${easy_limit} LE-$1"
                <action application="bridge" data="${easy_dialstring}"/>
        </condition>
</extension>

<extension name="easyroute_limit_exceeded">
        <condition field="destination_number" expression="^LE-(\d{10})$" break="on-true">
                <action application="respond" data="503 Buzz Off You Ran Out of Channels"/>
        </condition>
</extension>

<!-- Simple example - just grab the dialstring and send the call -->
<extension name="easyroute">
        <condition field="destination_number" expression="^1?(\d{10})$" break="on-true">
                <action application="set" data="dialstring={easyroute($1 dialstring)"/>
                <action application="bridge" data="${easy_dialstring}"/>
        </condition>
```

```
</extension>
```

Field Definitions

destnum

Destination number, i.e. DID number

dialstring

Dialstring to use for outbound leg

group

Group name

limit

Limit value

acctcode

Account code

ODBC/Database Setup

FreeSWITCH

FreeSWITCH will need to be compiled with ODBC support. Please see the instructions for how to do this.

easyroute.conf.xml

The easyroute.conf.xml file will need to be configured with the settings for your database:

```
<param name="db-username" value="root"/>
<param name="db-password" value="password"/>
<param name="db-dsn" value="easyroute"/>
```

```
    <!-- Default Technology and profile -->
    <param name="default-techprofile" value="sofia/default"/>

    <!-- IP or Hostname of Default Route -->
    <param name="default-gateway" value="192.168.66.6"/>
```

NOTE: The default technology profile and default gateway will be returned if the db lookup fails.

Custom Query

You can provide your own custom query instead of using the table layout suggested below. However, the query **must** have these fields **in this order**:

```
gateway varchar(128) - contains destination gateway host:port pair (ex: 192.168.1.1:5060 )
group varchar(128) - contains optional group name
call_limit varchar(16) - contains optional call limit
tech_prefix varchar(128) - tech prefix used to build dial string (ex: sofia/default )
acctcode varchar(128) - used to set channel variable acctcode for logging into the CDRs
destination_number varchar(16) - Number returned in the query for building the dial string. (ex:
```

MySQL Sample Setup

```
CREATE TABLE `numbers` (
  `number_id` int(10) unsigned NOT NULL auto_increment,
  `gateway_id` int(10) unsigned NOT NULL,
  `number` varchar(16) NOT NULL,
  `acctcode` varchar(16) NOT NULL,
  `translated` varchar(16) NOT NULL,
  PRIMARY KEY  (`number_id`),
  UNIQUE KEY `number` (`number`),
  KEY `gateway_id` (`gateway_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='Numbers Table'

CREATE TABLE `gateways` (
  `gateway_id` int(10) unsigned NOT NULL auto_increment,
  `gateway_ip` varchar(16) NOT NULL,
  `group` varchar(15) NOT NULL,
  `limit` int(10) unsigned NOT NULL,
  `techprofile` varchar(128) NOT NULL,
  PRIMARY KEY  (`gateway_id`),
  KEY `gateway_ip` (`gateway_ip`,`group`)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='Gateways Table'
```

PostgreSQL Sample Setup

```
-- DROP TABLE gateways;

CREATE TABLE gateways
(
  gateway_id serial NOT NULL PRIMARY KEY,
  gateway_ip varchar(16) NOT NULL,
  "group" varchar(15) NOT NULL,
  "limit" integer NOT NULL,
```

```
  techprofile varchar(128) NOT NULL
);

CREATE INDEX gateways_ip_group on gateways(gateway_ip, "group");

-- DROP TABLE numbers;

CREATE TABLE numbers
(
  number_id serial NOT NULL PRIMARY KEY,
  gateway_id integer REFERENCES gateways(gateway_id),
  number varchar(16) UNIQUE NOT NULL,
  acctcode varchar(16) NOT NULL,
  translated VARCHAR(16) NOT NULL
);
```

SQLite Sample Setup

- If you have done this please add your notes here