Queue_which_calls_extensions

Introduction

The fifo application, where agents connect to fifo out and wait for customers, is probably the best way of handling queues in large organizations. In small organizations, though, where there are no full time agents waiting for customers, a queue which calls a number of extensions is more useful. This example shows how this can be accomplished.

Examples have been shown on the userlist which uses a combination of a php script and javascript coding. This example only uses javascript coding and has no requirement for a php demon running in the background. For the sake of terminology, we denote the extensions which are set up to ring, “agents”.

This queue system works much alike the queue in Asterisk: A number of agents are called in goes of XX seconds. If not answered in the first go, the system waits YY seconds before starting the second attempt. No active method is included to check if agents are engaged in order to call them as soon as they release. Instead, we just try in another go. The ring strategy can be modified by adjusting the dialstring created by the connectqueue.js script. The system does not include support for “you are number XX in the queue”, but I see no reason why the feature could not be implemented as well. The code examples below have not been test thoroughly, and it will be buggy! As you may see from checking the code below, the code and the measures used are not at all clear or clean, and a “real” dptools application which does the trick, would be much better than this.

The components

The example includes the following components:

1. A dialplan which receives calls from customers.
2. A disconnection javascript, disconnectqueue.js, which kills calls to all agents, when a customer abandons the queue before he has been serviced.
3. A javascript, playwelcome.js, which plays a welcome message to the customer.
4. A call small javascript, call_connectqueue.js, which (when customers enter the queue) calls another javascript, connectqueue.js, which connects to the fifo and calls the agents.
5. A javascript, connectqueue.js, which sets up a call from the fifo to the agents who are retrieved from a database.
6. A loopback dialplan, which passes calls from the gateway on to the right extension or the right external gateway.

Some mechanism is required in this setup to identify the call from the customer as it propagates through the scripts. In this example, the customer uuid is used and it is transferred to the rest of the application via javascript arguments and through SIP calls via the caller_id_number.

The dialplan which receives calls from the customer

<extension name="Receive queue, fifo customer" >

<condition field="destination_number" expression="^\*13\.*"/>

<action application="set" data="queue_caller_id_name=Queue $\{destination_number:4\} $\{effective_caller_id_name}"/>

<action application="set" data="queue_name=${used_domain}$\{destination_number:4}"/>

<action application="set" data="fifo_music=$\{hold_music}"/>

<action application="set" data="continue_on_fail=false"/>

<action application="answer"/>

<action application="javascript" data="playwelcome.js /var/spool/voiceprompts/$\{queue_name}.wav"/>

<action application="javascript" data="api_hangup_hook=jsrun disconnectqueue.js $\{argv\}"/>

<action application="javascript" data="call_connectqueue.js $\{argv\}"/>

<action application="fifo" data="$\{queue_name\} in"/>

</condition>

</extension>

The dialplan above takes calls from the customer. The dialstring is "*13<queue-number>".

A few comments: The queue_caller_id_name is used to carry some useful information about the call, which can be displayed to the agents when they take the call. Further down in the dialplan, the javascript call_connectqueue.js is called and this information is passed on together with the queue_namr, uuid and ANI as a part of a |-separated argument.

The javascript playwelcome.js plays a queue-dependent welcome message provided the presence of a .wav file.

If the caller hangs up before he has been answered, the javascript disconnectqueue.js is run. Without such a mechanism, the calls to the agents will just continue until answered.
Finally, the call is answered by the fifo application.

The disconnection javascript

The disconnectqueue.js looks up in the SQLite database: core, table: channels. The FreeSWITCH uses this database when keeping track of the calls. The calls are selected by the state and the caller-id_number, which has been set in the connectqueue.js script. The uuid is thus retrieved and killed using the api uuid_kill. As a precaution, this script removes the customer from the waiting queue. This may well have happened before, but it catches early abandon situations.

```javascript
//This script is used to disconnect agents being rung from a queue when abandoned by the customer
//before answered.
var variable = new String(argv);
var components = variable.split("|");
var fifoname = components[1];
var disconnect_identifier = components[2];
var ani = components[3];
var used_domain = components[4];
var queue_lengthname = "queue_length_" + used_domain + fifoname;
var waitinglistname = "queue_waitinglist_" + used_domain + fifoname;

use("CoreDB");
var db = new CoreDB("core");
var sql="select uuid from channels where cid_num='" + disconnect_identifier + "' and state='CS_EXECUTE';";
db.prepare(sql);
while(db.next()) {
  rec = db.fetch();
  apiExecute("uuid_kill", rec["uuid"]);
}
db.close();

//Deplete uuid from list of customers waiting
var waitingliststring = getGlobalVariable(waitinglistname);
waitingliststring = waitingliststring.replace("|" + disconnect_identifier,");
waitingliststring = waitingliststring.replace(disconnect_identifier + "|", "");
waitingliststring = waitingliststring.replace(disconnect_identifier, "");
setGlobalVariable(waitinglistname,waitingliststring);

console_log("info","Call in queue ended: " + fifoname + "\n");
console_log("info","waitingliststring: " + getGlobalVariable(waitinglistname) + "\n");
```

The playwelcome javascript

playwelcome.js:

```javascript
//This script checks if a voiceprompt file exists
filename = argv;
var fd = new File(argv);
if (fd.isFile) {
  session.answer();
  session.streamFile(filename);
}
```

The call connectqueue script

This script calls another script via an api call. Thus the other script is executed outside the channel. Call_connectqueue.js:

```javascript
//This script used the fifo to simulate an Asterisk queue, TEST!
apiExecute("jsrun", "connectqueue.js "+argv);
```
The connectqueue javascript

The example below shows a javascript having room for much improvement such as handling of agent priorities, ring strategies, "you are number XX in the queue", etc. It simply generates a dialstring based on the queue number. The agents in this dialstring are retrieved from a database, which has the same format as in the real-time queue_member table in Asterisk, but any other mechanism for retrieving the agents could be used.

Connectqueue.js:

```javascript
// This script uses the fifo to simulate an Asterisk queue without priorities!

// We call the dialstring and connect to the fifo with stated fifoname
function connectqueue(dialstring,fifoname,timeout,disconnect_identifier,used_domain) {
    var newSession = new Session();
    var result = newSession.originate(session,dialstring,timeout);
    if(result) {
        // We remove the customer from waiting list
        var waitinglistname = "queuewaitinglist_%" + fifoname;
        var waitingliststring = getGlobalVariable(waitinglistname);
        waitingliststring = waitingliststring.replace("|"+disconnect_identifier,"");
        waitingliststring = waitingliststring.replace(disconnect_identifier+"|","");
        waitingliststring = waitingliststring.replace(disconnect_identifier,"");
        setGlobalVariable(waitinglistname,waitingliststring);
        newSession.execute( "fifo", fifoname + " out nowait" );
        exit();
    }
}

// Creates a dummy session to be used for the wait function, we try 4 times
// If the session has been created before, we use that
function makedummysession() {
    var cont = true;
    var i = 0;
    var waitSession = new Session();
    while (cont) {
        result = waitSession.originate(session,"sofia/gateway/500/wait",1);
        if(result) {
            console_log("info","+++++++++++++++Waiting session created++++++++++++
");
            cont = false;
        }
        i++;
        if (i>4) cont = false;
    }
    return waitSession;
}

// Wait function
function wait(waitSession,timeout) {
    waitSession.execute("sleep", timeout*1000);
    // Alternatively, comment out the line above and uncomment the line below:
    // msleep(1000*timeout);
    // Just for debug, the error type makes it easily read!
    console_log("err","+++++++++++++++Waiting session++++++++++++
");
}

// Find position in queue, if -1: uuid is out of the queue
function queueposition(waitinglistname,uuid) {
    var waitingliststring = getGlobalVariable(waitinglistname);
    var j = -1;
    if (waitingliststring) {
        var waitingliststringarray = waitingliststring.split("|");
        var i = -1;
        var imax = waitingliststringarray.length - 1;
        while (i<imax) {
            i++;
            if (waitingliststringarray[i] == disconnect_identifier) {
                imax = i;
                j = i;
            }
        }
    }
}
```
// We check if the caller - customer end - is still alive
function checkcall(uuid) {
  use("CoreDB");
  var db = new CoreDB("core");
  var sql = "select uuid from channels where uuid='" + uuid + "' and state='CS_EXECUTE';";
  db.prepare(sql);
  var alive = false;
  if (db.next()) alive = true;
  db.close();
  return alive;
}

// We create the dialstring, which includes all the phones to call
// This info is fetched from a database
function getDialstring(caller_id_name, ani, fifoname, disconnect_identifier, used_domain) {
  use("ODBC");
  var DSN = "maxpowersoft_odbc";
  var DB_USER = "XXXXX";
  var DB_PASS = "YYYYY";
  var db = new ODBC(DSN, DB_USER, DB_PASS);
  db.connect();
  var dialstring = "";
  // The asterisk type queue member table is used here as an illustration only
  var sql = "select interface from queue_member_table WHERE queue_name='" + fifoname + "' ORDER BY interface;";
  // var sql = "select astvalue from astdb WHERE astfamily='K00001210' AND astkey='MYPHONE_201;";
  // console_log("Info", sql + "\n");
  db.exec(sql);
  // Variable communication is via caller_id_name, as this call is leaving (and returning) FS
  // Remark: the dialstring below uses "," as separators => simultaneous call to all agents
  // Another strategy for the queue could be defined by playing with the dialstring constructed below.
  while (db.nextRow()) {
    record = db.getData();
    // We parse the Asterisk style interface and get the real number
    queuemember = record['interface'].substr(7).split("@")[0];
    // Right below, we check if the agent is an external number
    if (queuemember.length > 7) {
      dialstring = dialstring + ",[origination_caller_id_name=" + ani + ",origination_caller_id_number=" + disconnect_identifier + "]sofia/gateway/500/*delay*gateway45161002/" + queuemember;
    } else dialstring = dialstring + ",[origination_caller_id_name=" + caller_id_name + ",disconnect_identifier = "]sofia/gateway/500/*delay" + user + queuemember;
  }
  db.close();
  console_log("notice", dialstring.substr(1) + "\n");
  return dialstring.substr(1);
}

// Initialization of variables
var variable = new String(argv);
var components = variable.split("|");
// When arguments with spaces are passed to the variable argv, the spaces are replaced by commas.
// Right below, we reverse this.
var caller_id_name = components[0].replace("", " ", "g");
var fifoname = components[1];
var disconnect_identifier = components[2];
var ani = components[3];
var used_domain = components[4];
// We don't count, instead we use the max waiting time in the queue
var trials = 0;
var maxtrials = 100;
var waitingtime = 0;
var maxwaitingtime = 1800;
var timeout = 10;
var wait_between_trials = 5;
var used_timeout = timeout;
var waitinglistname = "queuewaitinglist_" + fifoname;
var dummysessionmade = false;

//add customer (stored as uuid) to end of waitingliststring
var waitingliststring = getGlobalVariable(waitinglistname);
waitingliststringarray = waitingliststring.split("|");
var restart = false;
if (waitingliststring.length>0) waitingliststring = waitingliststring+"|"+disconnect_identifier;
else {
    waitingliststring = disconnect_identifier;
    restart = true;
}
setGlobalVariable(waitinglistname,waitingliststring);
dialstring = getDialstring(caller_id_name,ani,fifoname,disconnect_identifier,used_domain);
used_dialstring = dialstring.replace("*delay*",0,"g");
i = queueposition(waitinglistname,disconnect_identifier);

//For each loop, one could chose a different dialstring, thus implementing
//some kind of priority among the agents
while ((waitingtime<maxwaitingtime) && (i>-1)) {
    if (i == 0) {
        connectqueue(used_dialstring,fifoname,used_timeout,disconnect_identifier,used_domain);
        used_dialstring = dialstring.replace("*delay*",wait_between_trials*1000,"g");
        used_timeout = timeout + wait_between_trials;
        waitingtime = waitingtime + used_timeout;
    } else {
        if (!dummysessionmade) {
            waitSession = makedummysession();
            dummysessionmade = true;
        }
        wait(waitSession,wait_between_trials);
    }
    i = queueposition(waitinglistname,disconnect_identifier);
}

//Some information for debugging...
console_log("err","+++++++++++++++ ANI: "+ani+" ++++++++++++\n");
console_log("err","+++++++++++++++ Waitingliststring: "+getGlobalVariable(waitinglistname)+" +++++++++++++\n");
console_log("err","+++++++++++++++ Position in queue: " + i + " +++++++++++++\n");
}

//After the trials we hangup the customer
apiExecute("uuid_kill", disconnect_identifier);

Please note that the gateway numbers above are specific to the dialplan, replace when appropriate.

The following seems to be obsolete, based on one person's specific setup and only tested against an old version of FreeSwitch..... will be deleted later in 2012 if nobody comments otherwise........

The dialstring generated by the function getDialstring is a bit tricky:

First of all, we want the possibility of calling a many agents simultaneously. In the versions tested (last tested on is 1.0 trunk (10220)), a dialstring using the
syntax: user/user1, user/user2 only calls user1. I have tried to find out if this is a general problem or related to my specific setup, and according the
userslist, it seems to be specific to my setup. Anyhow, it has also been reported by others (B Karthik). To circumvent this problem, the dialstring calls the
agent through a gateway which is looped back to the FreeSWITCH. By this trick, we achieve almost the same as using the Local channel in Asterisk (I
apologize if you have no Asterisk background). The required channel variables are propagated through the caller_id_number and caller_id_name. This is
not very elegant, but it works.

The format used for the gateway dialstring depends on the agent: if the agent is an extension, one format is used, if it is an external number, another
format is used. This also depends on the specific dialplan used in the solution.

The script uses a special construction for creating a "sleep". This is used for customers, who are not in front of the queue. We let them wait some seconds
before we check their position. A dummy session is created, which in combination with the session.execute("sleep","xxxx") yields the desired result. This
may not be a smart way, and a functioning wait command in javascript, which does not rely on an existing session, would be nice!

The msleep command, which was introduced by Anthony, has been tested, but made the application unpredictable, and the sessions hang. I have tested it
in various versions, the last version tested is 10638, where it sometimes messes up the audio.
The fifo information, such as the uuid in the queue, could be retrieved using a command like this: `fifoinfo = apiExecute("fifo", "list " + fifoname);`, but the problem is that the script runs before the fifo has been populated, and such information would not exist when needed the first time. Therefore, we have opted for another solution: We store the required information in a global variable (called: `queuewaitinglist_ + <fifoname>;`), and we update the variable when needed.

**Loopback dialstring**

The calls initiated by the `connectqueue.js` script are looped back to the switch, where they are treated by the dialplan:

```xml
<extension name="loopback to local and external">
  <condition field="destination_number" expression="^\d*user(.*)$" break="on-true">
    <action application="sleep" data="$1"/>
    <action application="set" data="effective_caller_id_number=${caller_id_name:10}" />
    <action application="set" data="effective_caller_id_name=${caller_id_name}" />
    <action application="bridge" data="user/$2@${used_domain}"/>
  </condition>
  <condition field="destination_number" expression="^\d*gateway(.*)$">
    <action application="sleep" data="$1"/>
    <action application="set" data="ignore_early_media=true"/>
    <action application="set" data="effective_caller_id_number=${caller_id_name}" />
    <action application="set" data="effective_caller_id_name=${caller_id_name}" />
    <action application="bridge" data="sofia/gateway/$2"/>
  </condition>
</extension>
```

The dialplan above includes a sleep application used to insert delays between the ring rounds. Unfortunately the javascript does not include a sleep function, so pauses between the dial rounds must be implemented in this fashion. The delay is parsed from the destination_number.

For wait function (before agents are attempted called), a special dialplan used for a semi-permanent session, is made:

```xml
<extension name="loopback to local and external, used to create a wait">
  <condition field="destination_number" expression="wait">
    <action application="answer" />
    <action application="sleep" data="3600000"/>
    <action application="hangup" />
  </condition>
</extension>
```