Loway

The WombatDialer User Manual

Loway SA

Version 25.02, 2025/01/22

Table of Contents


wombat
DIALER

What is WombatDialer?

WombatDialer is a new-generation mass outbound calling platform for the Asterisk PBX.

It can be used to implement many different services. By offering a set of ready-to-use components and a monitoring GUI, you can create complex solutions in minutes.

WombatDialer can work on pre-defined call lists or can dynamically create them over an API (e.g. dial number X after 10:30 AM). It shares the load on one or more PBX servers and has a flexible rescheduling logic to handle missed calls. It is built to be used with your existing Asterisk PBX and does not require separate servers or a separate set of lines. It can call over VoIP or through the public telephone network.

WombatDialer is built to integrate with your business processes, can receive calls to be made over HTTP and/or notify an external system in real-time about calls made and results gathered.

WombatDialer works natively with the QueueMetrics Call-Center Monitoring Suite in order to produce state-of-the-art campaign analyses and insight.

Why was WombatDialer created?

If you are an Asterisk integrator, it may have happened to you: one of your clients requires simple outbound capabilities, e.g. calling back a customer who filled in a recall form on their website. Simple enough.

You start by creating an Asterisk callfile to generate each call - it works nicely and it is easy to set up, but if the call does not complete then it is lost. So you have to create a process that keeps track of the call and retries it in time. And that's not an easy feat to pull off when starting from a callfile.

The first thing your client notices is that calls end up using an unpredictable amount of lines - as they basically have an office PBX, people cannot dial in because at peak times your script is saturating all outgoing channels. Management is not happy and you have to keep track of trunk usage - not only your own, but your client's as well.

Then your client notices that those calls should not be made outside business hours - a customer might require a call at night, but there must be someone at their offices in order to call back. So you have to implement a calendar in your custom application.

Now that you have the calendar, your client notices that calls are generated at inconvenient times - sometimes all of their service reps are sitting idle, and other times they are all busy and calls keep piling up. So you have to edit your scripts to keep track of the current end-point statuses and decide when it is a good time to call.

Just at this point, they start to saturate their existing PBX, so they need to set up a cluster of boxes and they want your application to handle this. And while you are at it, what about usage statistics? and why not running different outbound services at once? and did they tell you they need to integrate their existing CRM? and could you add predictive capabilities to the set? and....

It looks like a nightmare. And it actually is - been there, done that. That's why WombatDialer was

created: all common outbound logic should be encapsulated through a declarative interface. No need to reinvent the wheel. You program the dialplan to be executed - either manually or through a GUI - and WombatDialer takes care of the rest. You create scripts if you need to send and receive data from WombatDialer, and you can control it all through a simple HTTP interface.

How much time is this going to save you?

Key features

- "Just Works" with your existing Asterisk PBX
- Easy, automated installation
- High scalability: from one to hundreds of outbound lines on multiple servers
- Run multiple prioritized campaigns in parallel
- Different dialing modes - automated, reverse and preview
- Pervasive security model with extensive auditing capabilities
- Programmable handling of calls that do not complete
- Easy to integrate through its HTTP API
- Strong real-time monitoring capabilities
- Runs locally - you do not have to depend on third-party services
- Provides a set of "building blocks" so you can create custom-tailored solutions

Typical usage scenarios

Telecasting

Send a pre-recorded message to a set of receivers. The message can be easily customized by having your PBX read custom variables, e.g. current account balances, planned service outages, end of current subscription periods. Works with your existing PBX.

- Automated warning systems (e.g school alerts)
- Event cancellations
- Number verification services

Telemarketing

Send a pre-recorded message to a list of contacts, and offer them an option to be put in contact with an operator if interested. When required, a maximum call duration can be enforced.

- Appointment reminders and cancellations for physicians, dentists, etc.
- Track subscription expirations and process renewals
- Product offerings
- Debt tracking and collection

Voice conferencing

Ever tried setting up a conference call with many attendants? WombatDialer can connect them all in parallel at the click of a button - no more wasted time and manually dialing busy numbers.

- Connect tens or hundreds of parties at once
- Different parties can have different access levels, e.g some may speak and some may only listen
- Virtual town hall

Automated phone interviews

Connect to a group of receivers and offer them a set of IVR options (reverse IVR). WombatDialer keeps track of selected options and forwards them to your tracking system.

- Automated service satisfaction interviews
- Quality Assessment of your services
- Instant automated polling stations

Getting started with WombatDialer

Using WombatDialer is easy once you get the hang of how it works. In order to work profitably with it, there are a few prerequisites that you should be aware of:

- WombatDialer drives one or more Asterisk PBX's and uses them to place calls. You should have the logins and passwords of the AMI interface for these PBXs.
- You must know the names of the trunks WombatDialer will use - they depend on the local setup. E.g. `SIP/myprovider/123456` or `DAHDI/g0/123456` might be valid formats to dial the number `123456`.
- When a call is started, WombatDialer directs it to some place in the Asterisk dialplan in order to be processed. So you should have a general idea of how to program your Asterisk PBX in order to execute the functionality you're after - e.g. playing a broadcast message, recording an audio message, setting up an inbound queue. You should know *where* in the dialplan these actions will happen, in terms of *extension* and *context* as used by Asterisk.

In the following example, we will show you one of the simplest things you can do with WombatDialer - that is, setting up a Direct Campaign and calling a list of numbers. All numbers will be called in sequence; if any of them cannot connect, then it will be retried.

To make life easier, we will run the following example by using the free 2-channel demo key that is embedded in WombatDialer. You may want to test more complex scenarios by getting a free demo key from our website in order to test-drive WombatDialer thoroughly.

We are going to be using FreePBX in this example and assuming that the PBX is already up and running; having queues and extensions already configured.

This is our battle plan:

- Install WombatDialer
- Set up WombatDialer
- Create a list of numbers to be called
- Create a new campaign
- Create reschedule rules for our campaign
- Run your first campaign
- Run a report on the campaign

Install WombatDialer

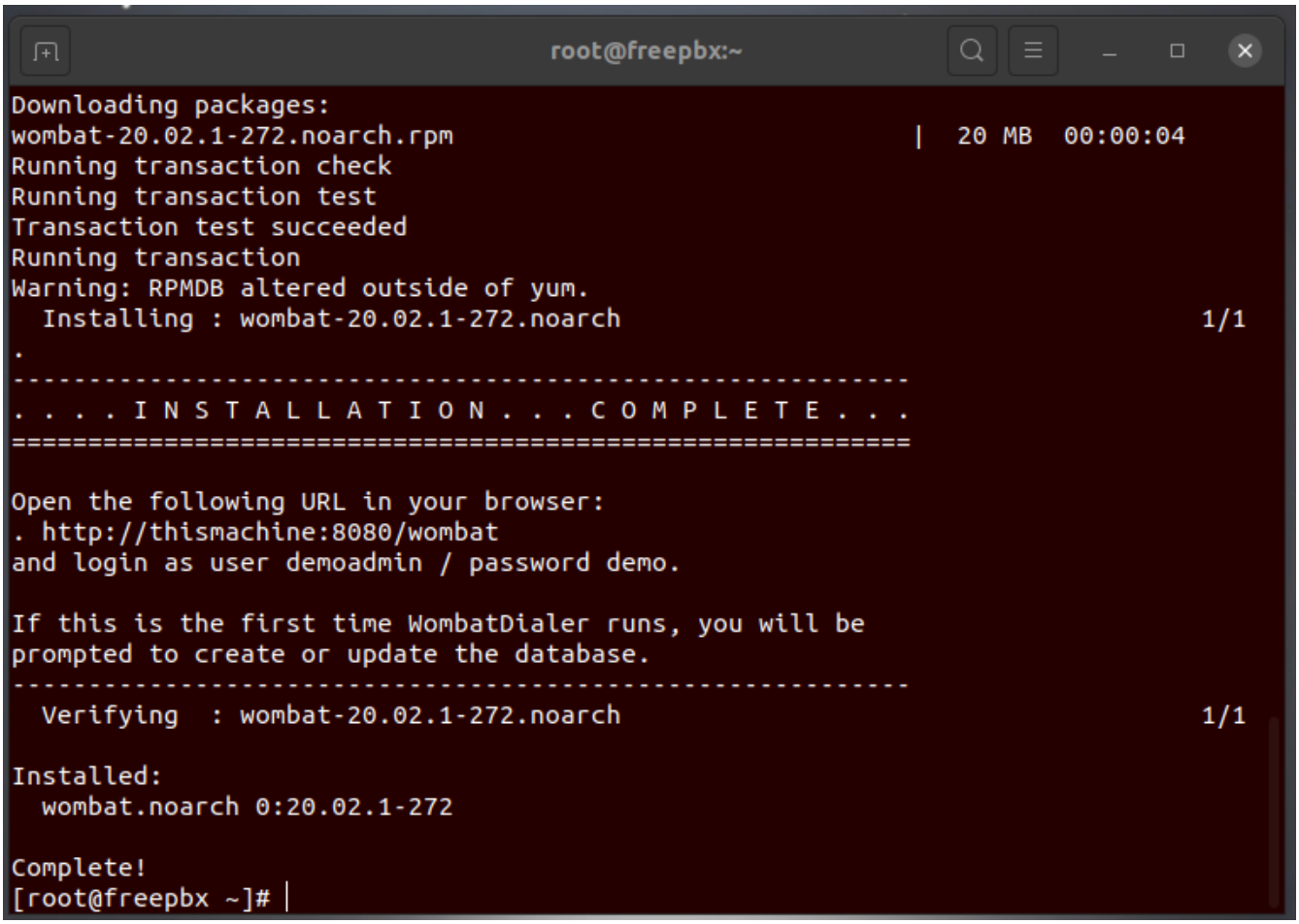
We will be installing the dialer on the same box as your PBX is. Please note that WombatDialer is NOT meant to be installed on an internet-accessible system.

Installation is very easy; on an FreePBX box it is enough to login as `root` and enter the following commands:

```
wget https://yum.loway.ch/loway.repo -O /etc/yum.repos.d/loway.repo
```

```
yum install wombat
```

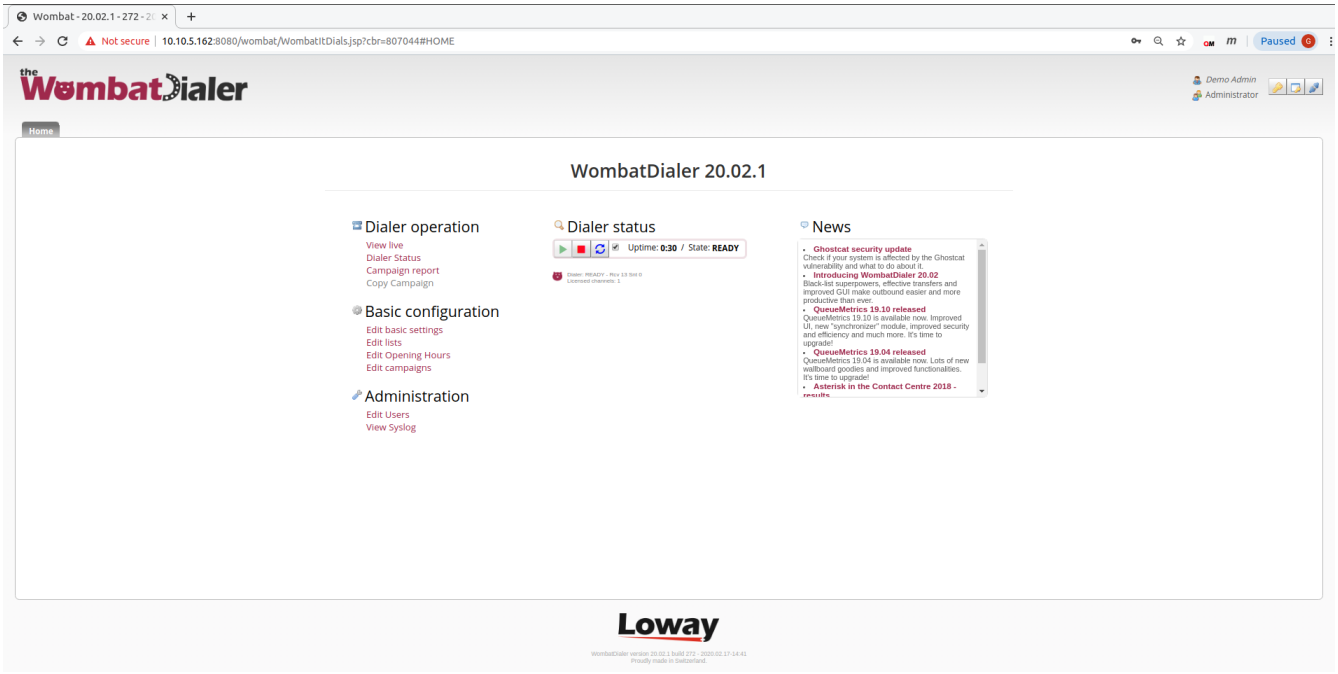
A number of packages will be selected and installed on your local system. This may take a while.



When done, go to `http://serveraddress:8080/wombat` and follow the instructions to create the database.

When done, login as `demoadmin` password `demo` by clicking on the Users icon on the top-right.

If all goes well, you should be presented a screenshot like the following one:



Set up WombatDialer

Inside WombatDialer; after logging in, click on "Edit basic settings" and add a new Asterisk Server like:

- Server description: MyPBX
- Server Type: Asterisk AMI
- Address: 127.0.0.1
- AMI port: 5038
- Login: cxpanel
- Password: cxmanager*con

Near the "Server Type" you will find a "plug" icon. Use this button to test the AMI connection with your PBX.

in this example, we use the default AMI user for FreePBX - it would be generally better to create a new user specifically for WD.

Now we create a new trunk. This will be used to dial numbers out. We might be using a SIP trunk, a local channel or whatever suits you. For the moment we use the default format used by FreePBX, so that numbers will be composed as if they were dialed on a local extension.

- Asterisk server: MyPBX
- Trunk name: MyTrunk
- Dial string: `Local/${num}@from-internal/n`
- Capacity: 10

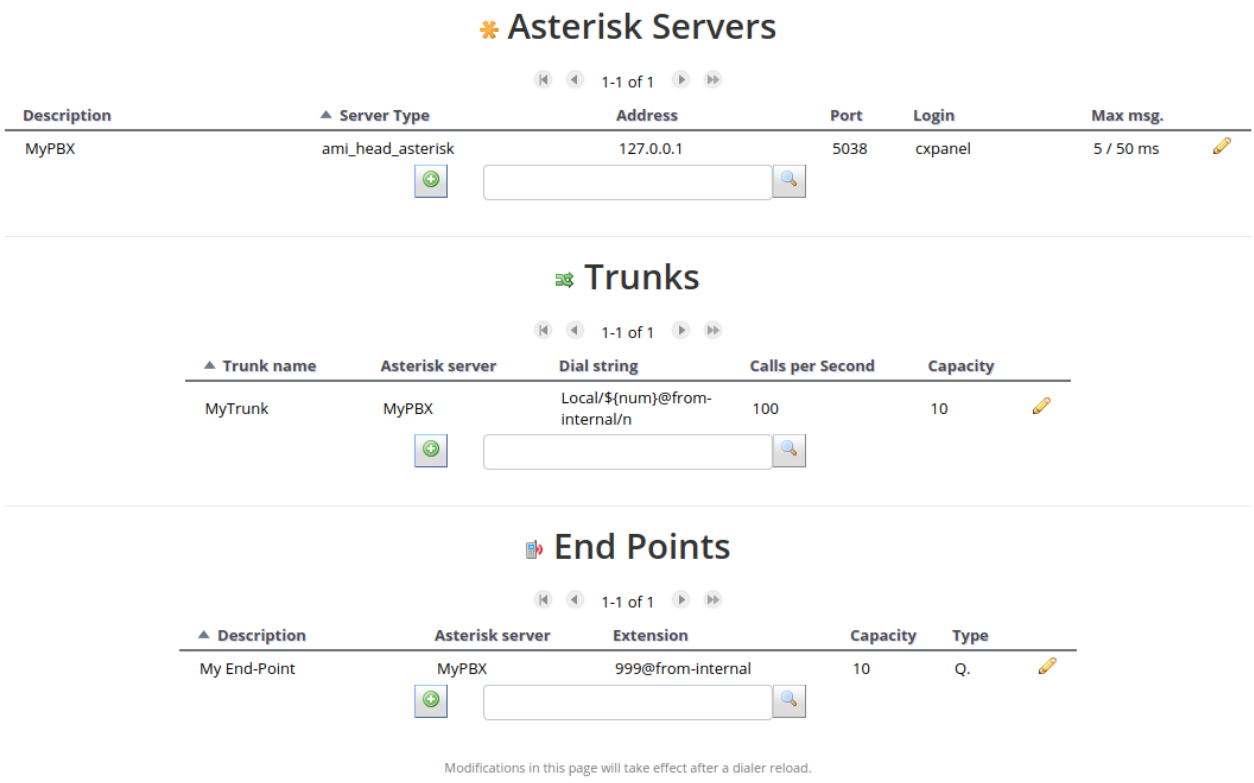
Near the "Dial string" you will find a "plug" icon. Use this button to test the Trunk.

Now we create an end-point, that is a destination for calls that are successfully answered. In our example the EndPoint will be a queue where extension 201 is waiting to be connected with the customer.

- On server: MyPBX
- EP Type: Queue
- Description: My End-Point
- Max Channels: 10
- Located at [extension]: 999
- Located at [context]: from-internal
- Queue name: 999

Near the "Located at [extension]" you will find a "plug" icon. Use this button to test the context that you are using.

By the end of the configuration, your page should look like the following one:

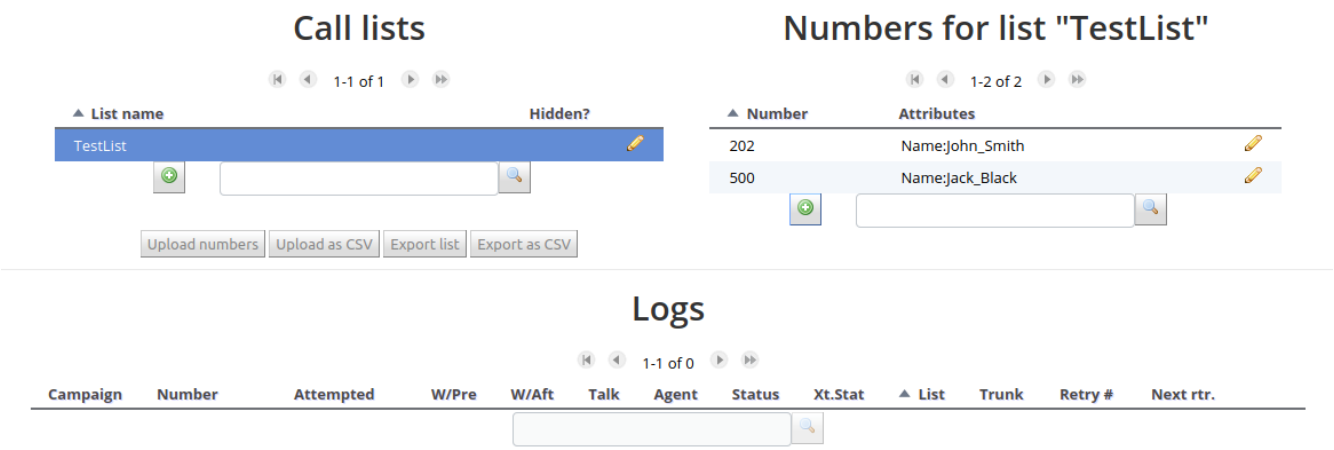


Create a list of numbers to be called

Now go back to the main page, click on "Edit lists" and create a new list called "TestList". The list has no settings, just a name.

Select the list just created on the left and then press "+" on the "Numbers for list" control. From here

we add numbers - you may enter any numbers you want, as well as Attributes.



You may want to add local extensions and remote channels - WD will work just fine. In our example, we added a local extension and a non-existent local extension, in order to display what WD does when a number cannot complete.

For real-life usage, you may prepare a CSV file with your favorite spreadsheet and upload thousands of numbers at once.

Create a new campaign

Let's go back to the home page and click on "Edit Campaigns"

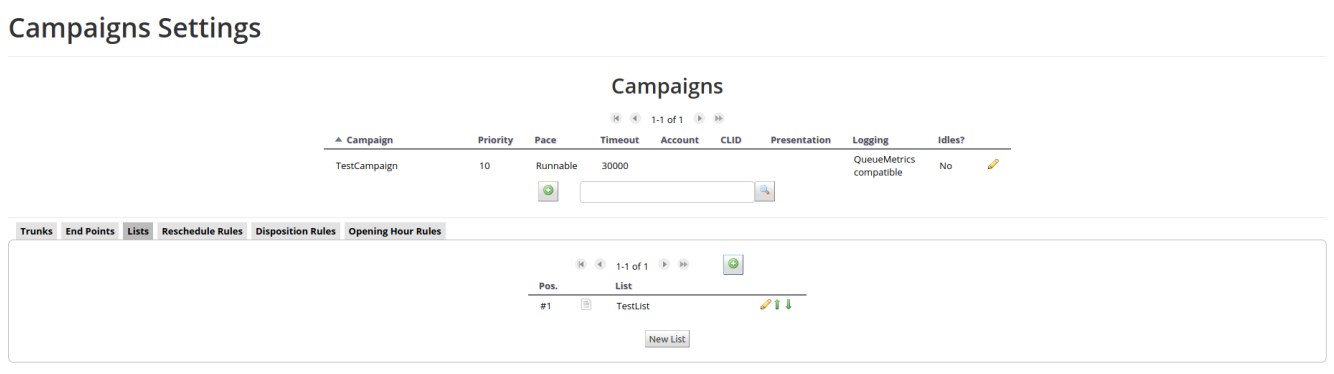
Add a new campaign by setting the following parameters:

- name: TestCampaign
- status: RUNNABLE

Then save it and select it; add the trunk, the end-point and the list you just created.

It is important to notice that, once you add a trunk or an end-point to your campaign, to change it with another one you need to add the new one and then remove the old one using the edit panel (the pencil icon). If you're adding lists or reschedule rules instead, you can change them directly by selecting a new rule or list in the edit panel.

At the end of the configuration the campaign will look like:



Campaigns have many options you can set to control their behavior - for example, you can set a caller-id, or the time period they are supposed to be active it, or how their results should be logged and notified. A detailed description is available on the Campaigns chapter.

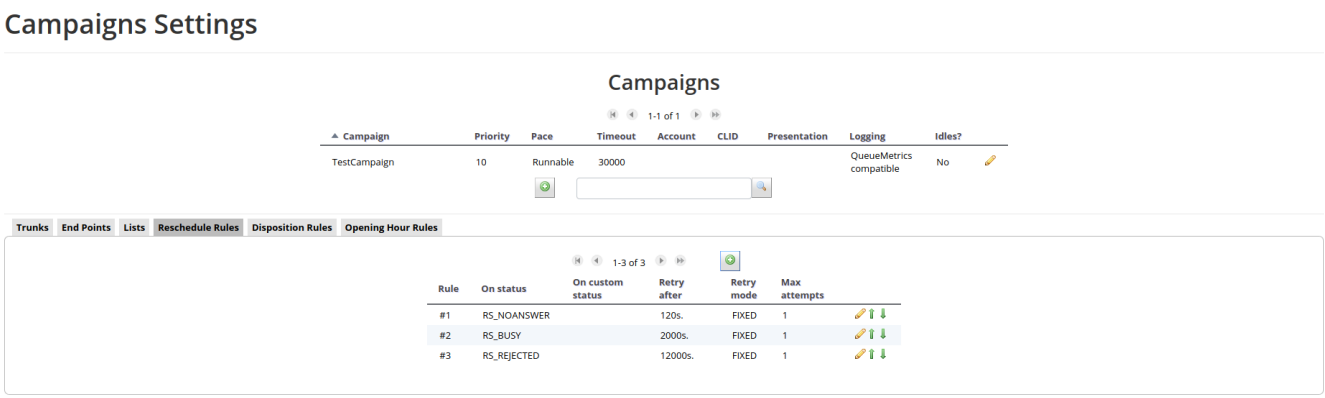
Create reschedule rules

While the campaign is selected, take a second to add reschedule rules. Basically they tell WD what to do in case a call does not go through. You will typically want to handle at least the following cases:

- RS_REJECTED: the network could not place the call
- RS_BUSY: the number dialed is currently busy
- RS_NOANSWER: the number dialed does not answer

For each case, add a new reschedule rule. Just enter a time in seconds after which the call is to be redialed, and a maximum number of retries.

You would expect to have something similar to the following example:

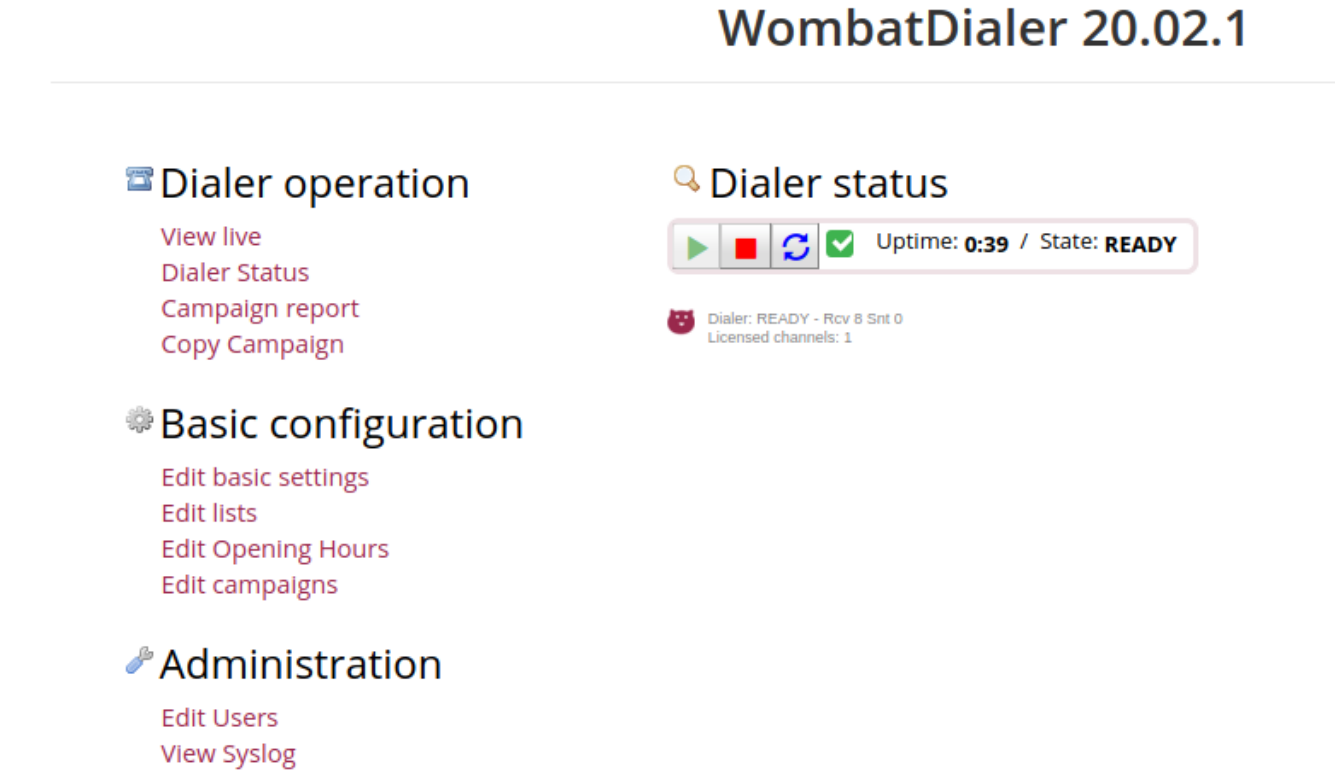


Note that, for each rule, you may have a different retry delay and a maximum number of retry attempts that is different based on the status encountered.

A detailed explanation of completion codes and how reschedule rules interact can be found at Reschedule Rules.

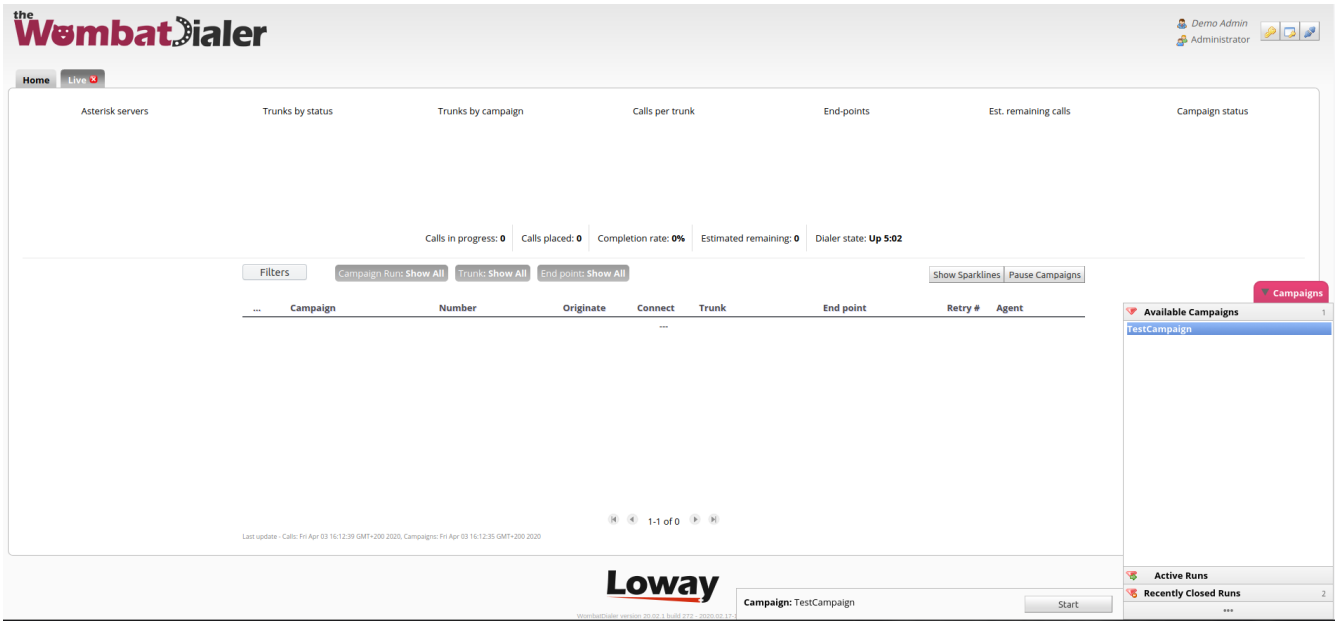
Run your first campaign

Now go back to the home page and start the dialer by clicking on the "Play" icon under "Dialer status". After a few seconds, click on the "Refresh" icon to confirm WD is up and running:

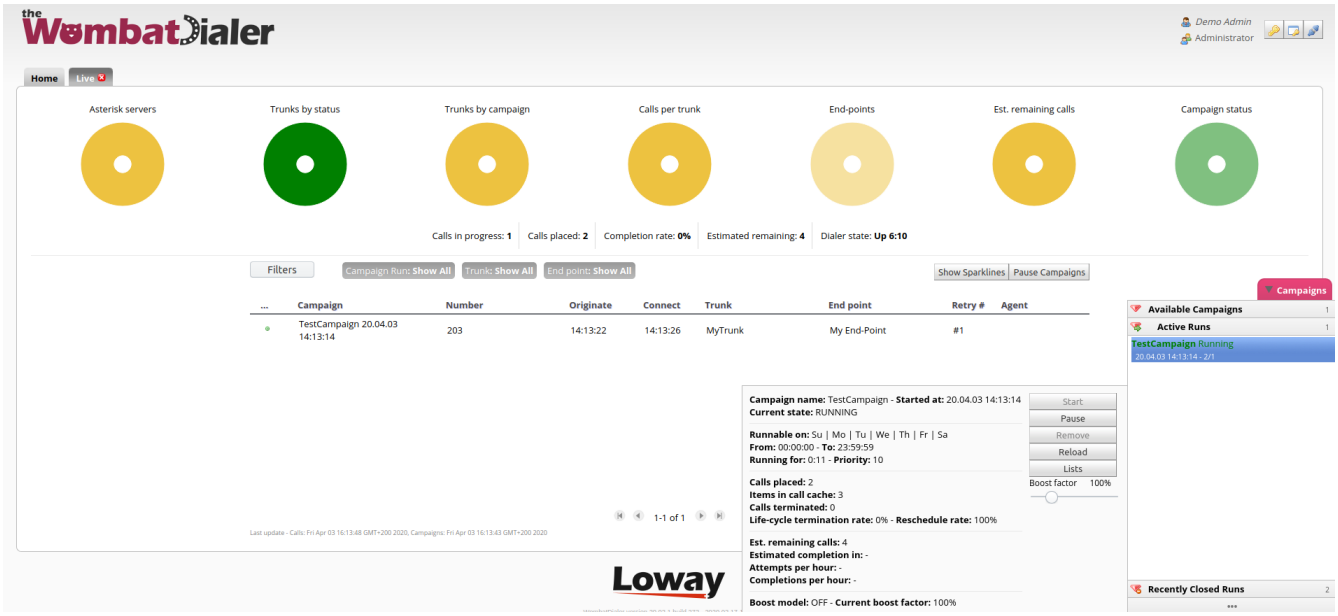


If you want to see what the dialer is doing without manually reloading, you may want to tick the box close to the Reload icon to enable automatic refresh.

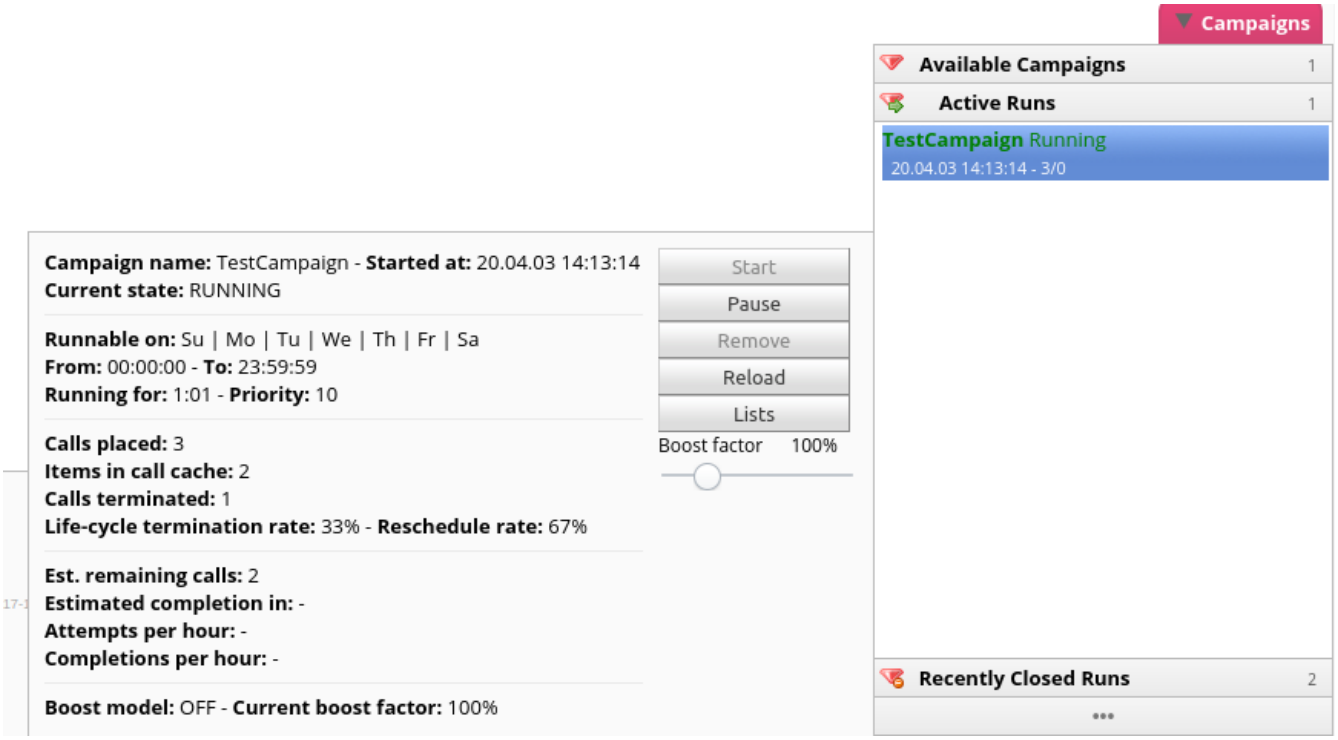
Click on the "View Live" page. From the "Available Campaigns" box, select your new campaign and run it by clicking on the "Start" button by the bottom of the page:



While the campaign is running, you will see WD dialing calls. Please note that while WD is running, your PBX can be used normally. The same campaign can be run only once - you cannot start it anymore when it’s running, but you will be able to run it again when it finishes.



If you select the running campaign from "Running Campaigns", you will see its completion statistics and expected termination time (as soon as it has completed enough calls to provide an estimate).

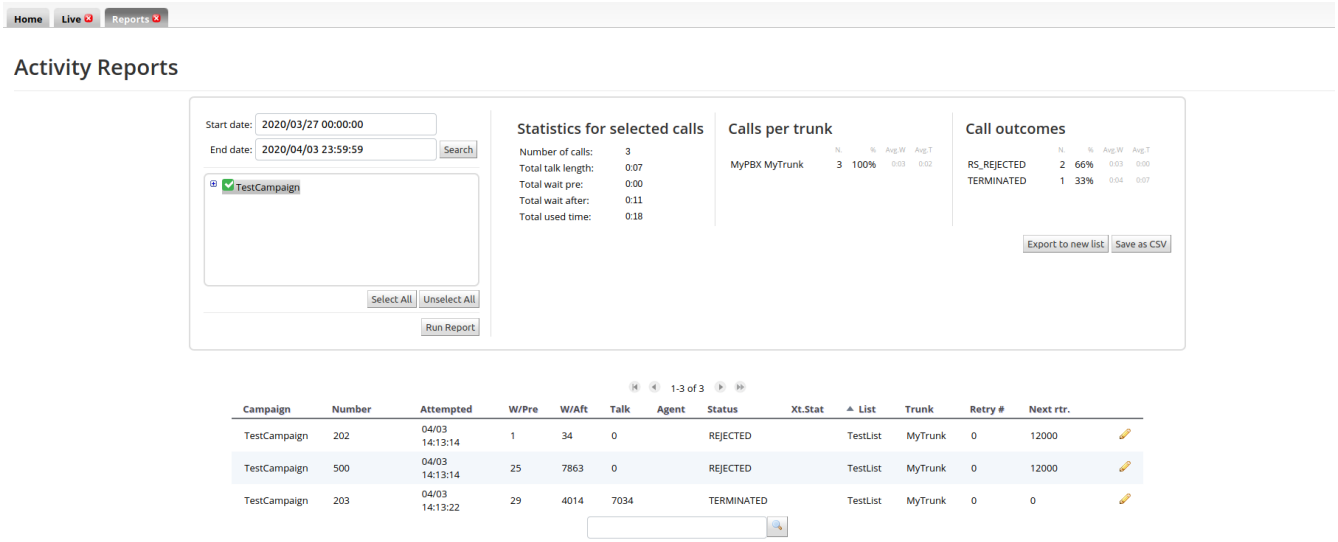


You can also click on a live call to see its details.

When the campaign completes, its run is placed in the "Recently closed" section and you are able to start it again.

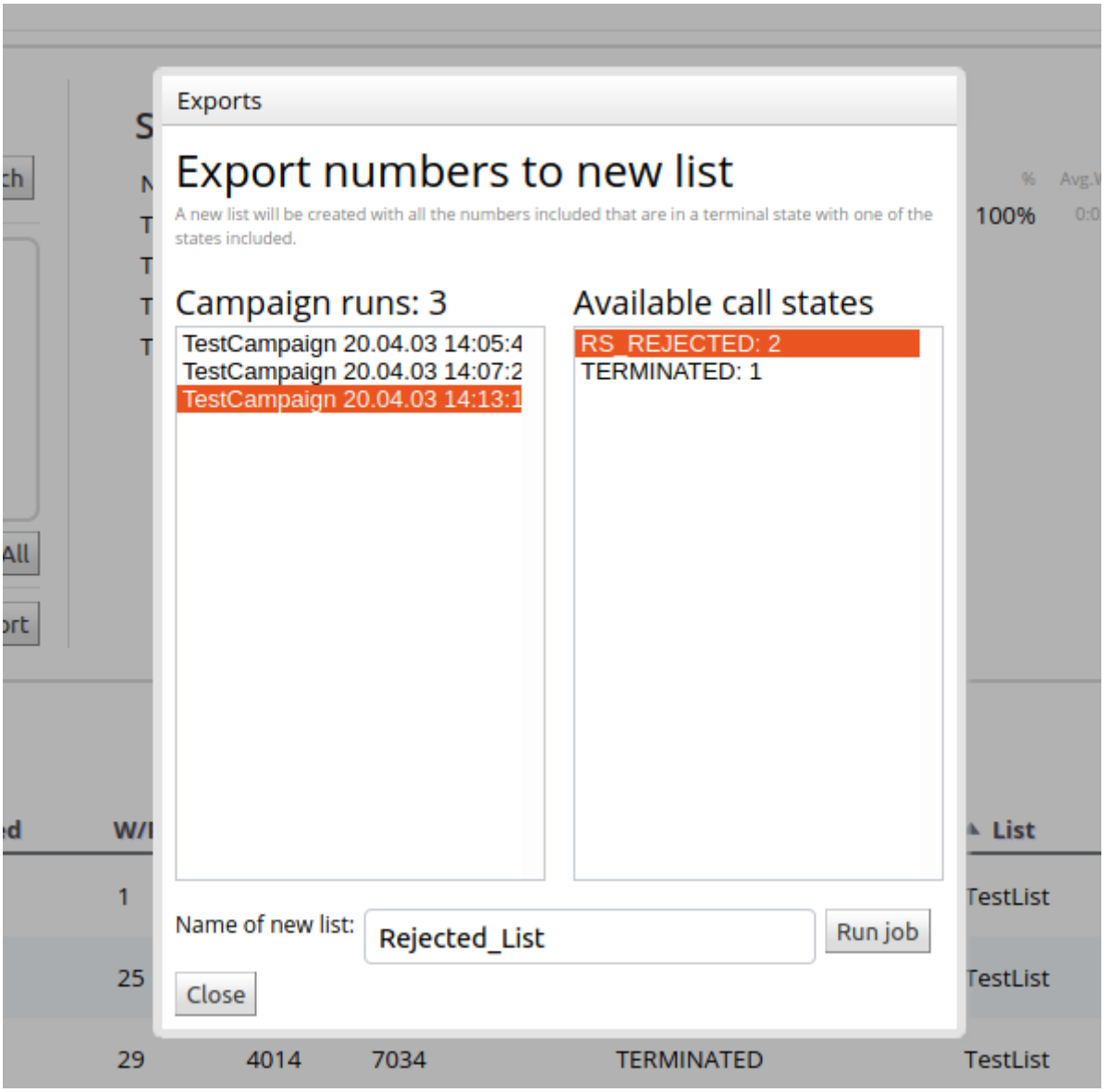
Run a report on your campaign

From the home page, click on "Campaign reports". Click on "TestCampaign" and make sure your run is selected. Then press ">>>".



You will see statistics about the runs included, trunk usage and status codes; and details for every attempt made.

If you want, from here you can create a new list that only contains calls for which the final status was not successful. Click on "Export to new list", select the runs you are interested in and the final statuses you want to include, and enter a name for your new list.



This way you can save calls to be retried at a later date.

Where to go from here

This manual provides a detailed description of the concepts involved in WD and how it is run and administered, plus a complete API reference.

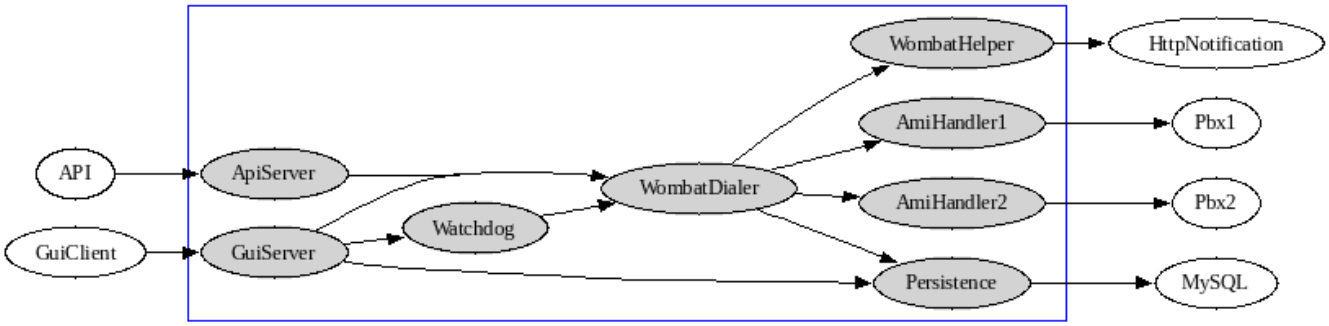
Of particular interest can be the section named "WombatDialer Cookbook" that includes a number of recipes that cover different real-world scenarios and show common usage patterns.

WombatDialer Concepts

In order to work profitably with WombatDialer, it is necessary to understand the core concepts that come into play.

The architecture of WombatDialer

WombatDialer is a complex system that is built out of different subsystems. Understanding which ones they are and what they are for will make understanding the whole product easier.



As a first thing, WombatDialer can be accessed through its **GUI Client** (in order to configure it, view what it is doing and run reports) or through **APIs** (see the HTTP API section). APIs are meant for external programs to control the behavior of the dialer, e.g. by controlling runs, adding new numbers to dial and creating new campaigns.

The dialer itself is controlled by the **GUI Server** - this way you can start and stop the dialer process from the GUI. When the dialer runs, an associated **watchdog** process runs - if the dialer process is to terminate for an unexpected error, the watchdog is supposed to log the error on the system log and restart it. As the dialer is able to sync to the state of an external PBX, during the restart phase calls might not be placed for a few seconds but existing calls will be preserved and tracked correctly.

When the dialer is running, it creates separate **AMI handlers** for each Asterisk PBX. Each of them runs as a separate thread and connects to a PBX through its Asterisk Manager Interface (AMI). If a PBX crashes, the rest of the system keeps on working; and if one PBX is delayed or loses connection, this does not impact other PBXs. In case of errors, each AMI handler will automatically retry until a connection is established.

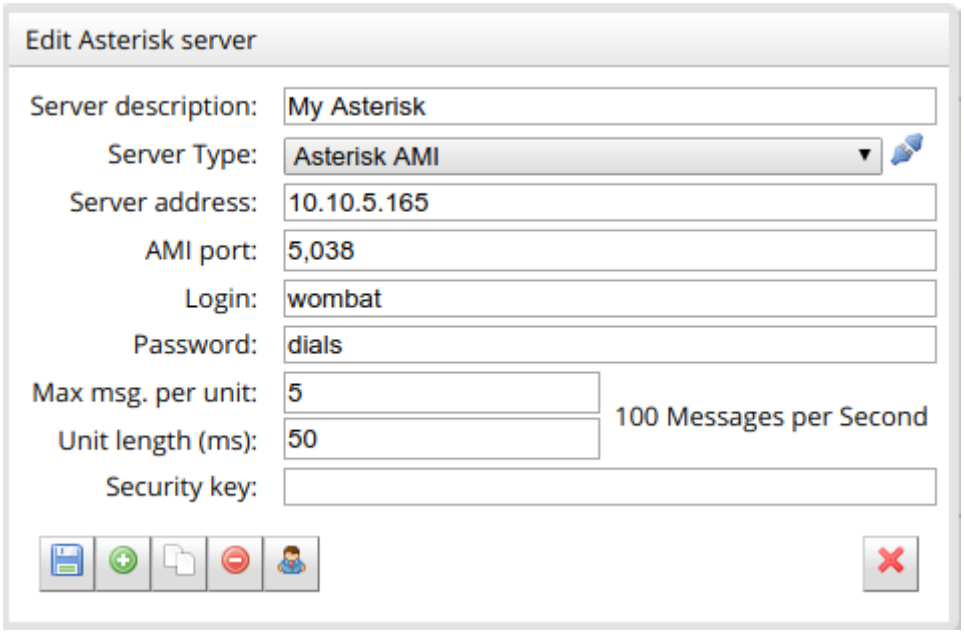
The dialer also spawns a **helper thread** that is meant to run long operations, e.g. **HTTP notifications** (see the HTTP Notifications section), without delaying the main dialer.

Both the dialer and the GUI use a **persistence layer** that reads and writes to the MySQL database used for long-term persistence and log tracking.

Asterisk servers

WombatDialer can control activities on multiple Asterisk servers at once. It does so by keeping a control channel open with each server through its Asterisk Manager Interface (AMI for short). Each

server is managed independently and in case of failure does not stop the rest of WD from running. Servers can be on the same network as WD is, or can be in remote locations.



When editing an Asterisk server, the following information is required:

- **Server description**: the name that this server will appear under in WD
- **Server type**: for an Asterisk system, select "Asterisk AMI". If running a test, you can select "Fake AMI" - see Fake AMI servers.
- Close to the Server type, there is the **Test Connection** icon button. It will try connecting to the currently server with the credentials you entered.
- **Server address**: the PBX server's name or IP address
- **AMI port**: the port that Asterisk's AMI interface uses. Default 5038.
- **Login** and **Password**: the login and password for the AMI user
- **Unit length** and **Max msg per unit**: these settings work as congestion control on the AMI port. WD displays the total maximum number of messages per second.
- **Security key** is the key that will protect this resource. Leave blank if not needed.

How congestion control works

In order to avoid flooding the PBX with too many messages at once, WD uses the concept of **Time Unit**; this is a period of time in which no more than a fixed number of messages can be sent.

For example, using the default values, if WD has to start 100 calls at once, it will send no more than 5 requests to Asterisk every 50 ms, and will queue the rest for the next time unit. This still amounts to a respectable rate of about 100 messages per second! (though multiple messages are needed to track a call's life-cycle).

The difference is hardly ever noticeable from a human point of view, but sometimes the PBX might crash if it receives too many requests at once. If your PBX runs on low-end hardware, you may want to reduce the number of messages per unit; on the other side, if your PBX is on a high-end

server, you may want to increase it. In general the defaults work fine in the majority of cases.

Do I have to configure Asterisk to work with WombatDialer?

You need to tell Asterisk that WD is allowed to connect and send commands. Our suggestion is to have a dedicated AMI user on each machine - so it is easier to keep track of which applications are connected to the PBX at a given time. To use Asterisk, make sure your server type is set to "Asterisk AMI".

You could e.g. have a stanza in your */etc/asterisk/manager.conf* file like the following one:

```
[wombat]
secret = dials
read = all
write = all
```

Even in high load conditions, it is safe to run WD with events "all", as it will automatically manage the set of events that Asterisk is to send it in order to avoid overloading the channel.

What is the Asterisk status?

When WD is running, it will display the current Asterisk status - if a successful connection is possible, then the Asterisk instance will be UP, otherwise it will be DOWN and WD will simply retry after a few seconds.

The current Asterisk status is propagated to the entities that belong to that instance - that is, its trunks, end-points and queues. If there are calls being processed while an Asterisk system becomes unreachable, WD will try and reconcile them to the current status as soon as the system comes back up. This should work even if WD is restarted in the meantime.

What is a "Fake AMI" server?

A "Fake AMI" server is a test server that can be used for load testing/integration testing, without the need of a physical PBX to connect to. It will simulate people answering the phone (or not answering it) - so it's a handy way to check if your environment is working or simulate an externally-driven campaign without actually making those calls. Or if you want to check how a WD instance behaves when connected to 5 separate Asterisk servers, without actually setting them up.

It will:

- try and answer 50% of the calls
- pick up calls between 1 and 3 seconds
- have calls open between 3 and 6 seconds

This ensures that the campaign will generate way more calls per second than your average "real" campaign.

When using Fake AMI, the settings for Server, Port and Login/Password are ignored (though they

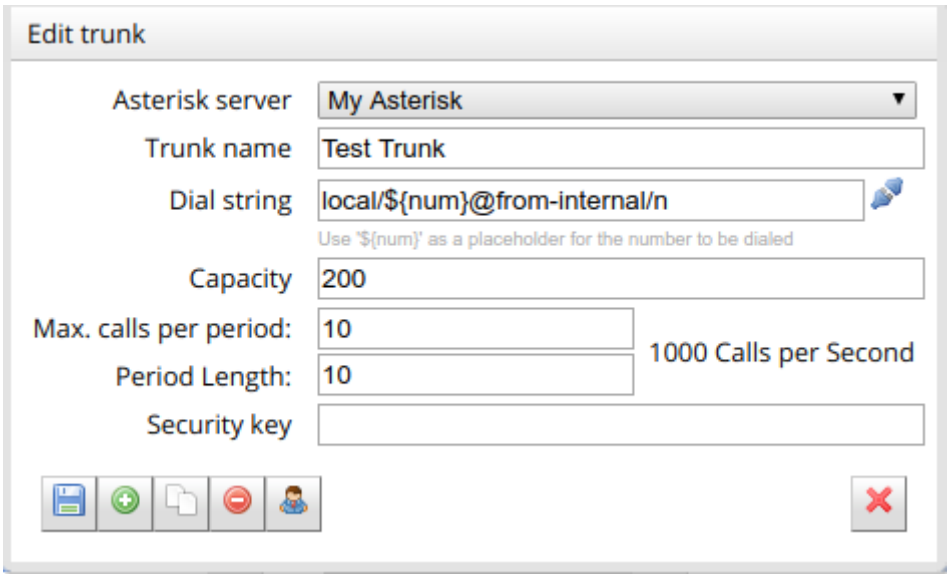
must still be entered).

Connection tests for Fake AMI always pass.

At the moment, the Fake AMI server does not support queue end-points; so it will not try simulating queues and will not send events for them.

Trunks

A trunk is a set of lines that are addressed as a single logical entity. They can be a set of physical lines (like a DAHDI interface in Asterisk) or a set of logical lines (like a connection to your SIP provider, or to another PBX). It may be one single line as well if nothing else is available!



- **Asterisk server**: is the name of the PBX on which this trunk is located
- **Name**: is a logical name for the trunk to appear in WD
- **Dial string**: is the actual Asterisk channel name that WD will invoke (see below).
- Close to the Dial string, there is the **Test Connection** icon button (see below).
- **Capacity**: is the number of parallel calls that WD can dial. Make sure you do not exceed the trunk's physical capacity!
- **Max calls per period** (number) and **Period length** (milliseconds): the maximum CPS rate limiter (see below).
- **Security key** is the key that will protect this resource. Leave blank if not needed.

You do not need to define all the trunks that are on your PBX, or to define them to their full capacity. For example, if you have a 15-channel E1 to your telco, you might define the trunk in WD as being a 10-channel one, so that you can use the rest with your PBX without any special rule.

It is also perfectly legal to define a physical trunk multiple times, splitting its capacity: again in the example you have a 15-channel trunk, you might define it as two trunks in WD, one having 10 channels and another having 5. Then you can assign each (or even both) to campaigns to control the maximum number of parallel outgoing calls.

A note on the dial string: it must be a valid Asterisk channel name, and the string *${num}* is replaced with the actual number being dialed. So the following ones are valid examples:

- *DAHDI/g0/0${num}* - dial the number through group 0 of your PSTN interface, prepending the digit zero
- *SIP/myprovider/${num}* - dial the number through the SIP server *myprovider* defined in sip.conf
- *Local/${num}@from-internal/n* - dial the number as if was input on an local extension when using a popular Asterisk GUI

When using channels of type *Local*, Asterisk will sometimes change them during the call, so that Wombat may lose them. Though WD is able to keep track of calls through multiple masquerades and channel renames, if all else fails, you may want to add a trailing */n* to your Local channels to make sure that their channel names are not renamed.

Testing a trunk

The **Test connection** button on the trunk allows you to check your trunk. In order to do this, WD first makes sure that there is a working Asterisk connection, and then tries dialing a channel on the trunk using a number that you supply. If all goes well, the number is called and a whistling "milliwatt" sound is played.

The test shows a "success" icon if the call is answered within 30 seconds.

For example, if your trunk is set with a channel format of "SIP/provider/${num}\" and you enter the number "5551234", WD will try dialing the channel "SIP/provider/5551234". Make sure your number is in a format that your provider accepts.

Limiting maximum CPS (calls per second)

A trunk allows fine control on its maximum numbers of calls per second. Many providers limit the maximum number of originates per second, usually to something like 25 or 30 (but it depends on your provider and the kind of contract you have with them). Providers often blacklist accounts that dial too fast, so if you exceed their allowed CPS rate all calls will end up as REJECTED or BUSY as soon as they are dialed.

As a trunk typically matches one specific outbound route, they usually map to a specific customer account for the provider that in turn would be used as their unit of measuring your originate rate.

As this check is enforced per trunk, if you have multiple campaigns sharing a trunk, no campaign can dial faster than the combined CPS; while if you have a campaign with two separate trunks, the maximum CPS will be the sum of each trunk's own CPS.

In order to implement CPS limiting, what happens is that when Wombat inquires for the number of empty channels on a Trunk, the Trunk will keep track of the number of calls just placed within the

last time period, so if the trunk has (say) 50 channels free, but in the last second 10 calls have been requested and the maximum CPS is 30, it will report back that it has only 30-10 = 20 channels free.

Rate limiting is expressed in terms of a maximum number of calls per a specific period in milliseconds. When a new trunk is created, its default is 100 calls per 1000 ms, that is a value high enough to mean "no CPS limits" for most scenarios.

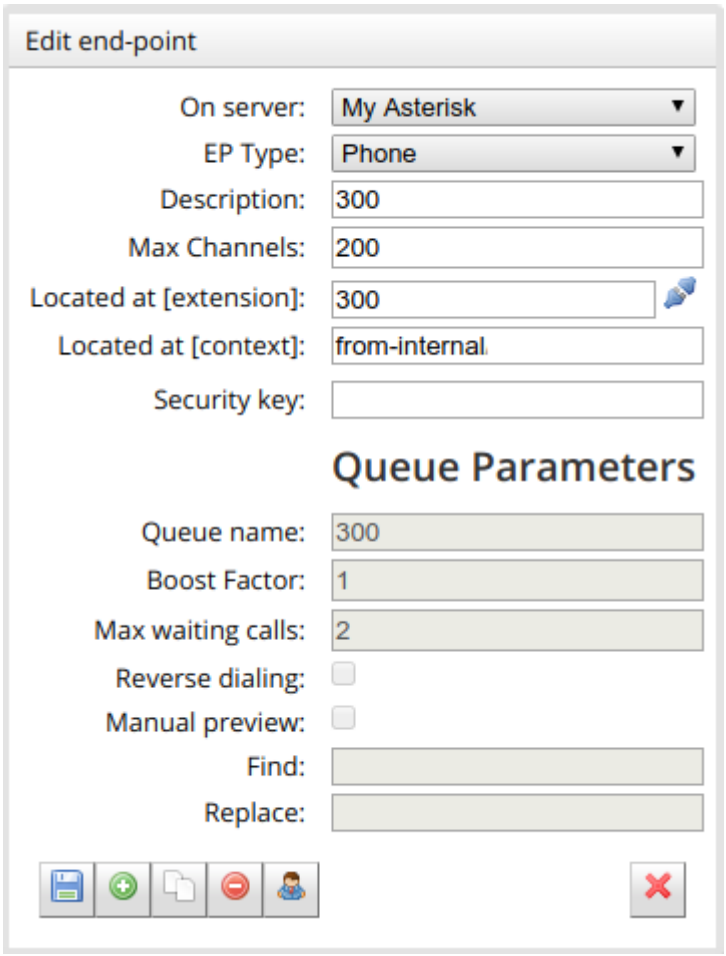
Using a lower CPS limit does not usually affect so much the speed of your campaigns in real-life, as WD will only effectively "spread out" the calls to be dialed over a few seconds. If you are concerned about this, you can see its current impact by monitoring how often the status **LIMITED BY TRUNK RATE** appears on the running campaign in the Dialer State page.

Calls that end up on a blacklist are evaluated before they are scheduled on a trunk, so rate limiting is not influenced by the rate of black-listed calls.

Rate limiting is mostly useful if you have a number of channels that is bigger than your maximum CPS.

End-points

An end-point is where a call goes after being answered on the trunk.



- **Server**: is the Asterisk server the EP is on
- **Description**: a free name that the EP will appear as

- **EP Type**: it can be QUEUE (if it is an Asterisk queue - see below), API or PHONE in all other cases.
- **Max channels**: the maximum number of parallel calls to be handled by this EP.
- Located at **Extension** and **Context**: the location in the Asterisk dialplan where the EP can be reached.
- Close to the Extension, there is the **Test Connection** icon button (see below).
- **Security key** is the key that will protect this resource. Leave blank if not needed.

WD will try and connect answered calls from trunks to end-points on the same Asterisk server. If a campaign has multiple end-points, it will try to connect any EP that has free capacity.

An end point of type PHONE could be:

- a dial-plan script that plays a recorded message
- a physical phone given to an agent (set max channels to 1 to receive one call after the other)
- an IVR script
- a conference call
- anything that can be programmed in Asterisk!

WD assumes that Trunks and EPs are always usable to their declared capacity, so it will try to fill them in as soon as possible (within allowed CPS limits). If they are to be shared, make sure you have enough capacity for WD plus other resources that may be using them. If you exceed their physical capabilities, you may experience REJECTED calls or a degradation of speech quality for VoIP. Beware!

Queue end-points

As a special case, it is possible to have EPs of type QUEUE. They are used to distribute calls on a set of agents that are members of a queue. WD will try and observe the queue, in order to determine:

- how many agents are logged on to the queue
- how many of them are currently available, that is, neither in conversation nor paused
- how many calls are currently queued without being answered

The difference between the number of available agents and the number of calls queued is taken as the current capacity of the queue. This value is computed in real-time, so the EP will immediately respond to changes in state to its agent set and to calls queued.

Please note that you can use a single queue both for inbound and outbound activity: if the number of free agents exceeds the number of queued calls, WD will try and fill-in the rest. This makes it easy to implement small blended (inbound/outbound) call-centers.

If the number of free agents is below the number of calls waiting, WD will not place any call and will wait until there are free slots on the queue. This way:

- calls are in general distributed to agents as soon as they are queued

- if the number of available agents is not enough to serve calls at once (e.g because some logged off, or pause in the meantime) then calls are queued and picked up by available agents when they become free.

Queue EPs have a few additional parameters that control their behavior:

- **Queue Name**: The exact name of the Asterisk queue being used
- **Boost factor**: as most of the calls in a campaign are going to be unanswered, it is often effective to have WD place a number of calls that is a multiple of available agents. For example, if you have a boost factor of 1.5 and 4 available agents, it will try and place 6 calls at once. If more calls are completed successfully than the available agents, the remaining calls are held waiting on the queue. The boost factor is applied only on the number of calls that should be made to saturate agents that do not have a call currently in progress for them. Please note that the boost factor can then be managed on running campaign from the Live page, so you can tweak it dynamically as needed - see Boost factor on the Live page - or you can have WD manage it for you based on current traffic conditions - see Adaptive boosting
- **Max waiting calls**: if there are more than this number of calls waiting on the queue, then stop making calls. The number of waiting calls is computed as the number of available channels on the queue minus the numbers of calls currently waiting, as Asterisk will report a call to be waiting even when it's being connected. In theory there should never be calls queued, as they follow the number of available agents, but it is possible that either some agent logs off after being counted or some calls reach a queue without passing through WD. This also acts as a counterweight to high "boost factor" values, to avoid having too large a backlog of calls to process if we are having a "lucky streak" in terms of calls answered.
- **Reverse dialing**: check to make this EP use Reverse dialing (see below for "Dialing mode")
- **Manual Preview**: check to use Preview mode (only valid in Reverse mode)
- **Find** and **Replace**: sometimes - notably on FreePBX systems - agents that are on a queue cannot be dialed directly, and therefore do not work in Reverse Dialing mode. You find that your queue has agents in the format *Local/4851@from-queue* but what you would like to do is to dial *Local/4851@from-internal* instead. By setting *@from-queue* as the Find item and *@from-internal* as the Replace item, you can easily obtain this result.

In addition to these parameters, the maximum capacity of the EPs is used - so if you have a queue with 100 agents but you set the EPs capacity to 10, WD will never use more than 10 lines on this EP at once.

General Asterisk tips for using Queue endpoints

In order to use WD effectively with a queue, the following guidelines are best followed:

- Though WD works with static member channels, if you want your calls to go through to agents who may or may not be available (e.g. some days they may be sick) it is strongly advisable to use dynamic agents who log on and off from the queue.
- As an agent cannot be physically available at all times during the day, it is important that they have a way to pause themselves, be it to run "wrap up" activities after calls or to take breaks. The QueueMetrics web interface offers an excellent panel that lets you add pause codes as well.

- The queue must provide informational "events" about agent activities. On Asterisk 12+ this happens automatically, but on older versions it must be enabled by explicitly setting `eventswhencalled=true` and `eventmemberstatus=true`. It is also important that extension presence is correctly observed - e.g. if an end-point is busy because the agent is making a personal call, its queue status should immediately reflect this. Whether this happens or not on your system is a matter of Asterisk version and type of channel that is used to reach the agent - with recent versions of Asterisk and SIP channels this should work automatically. You can make sure this is working correctly by observing the queue status as described in Controlling the Dialer
- The queue should connect calls to agent as efficiently as possible when there are multiple calls waiting and multiple available agents, so it should have the "auto-fill" option set to true.
- Do not use the queue wrap-up feature. As Asterisk does not publish wrap-up events though the AMI interface, WD sees the agent as "idle" and tries routing calls to paused agents.

API-driven queue end-points

An API driven end-point behaves exactly like a Queue end-point, but will not actively monitor the queue located on its Asterisk server; in fact, it will use the queue name as a simple placeholder, and will wait for you to update its status through a periodical API call (see API documentation). Like every other queue end-point, you can see it in real-time - and check that your API updates are working - from the Dialer State page.

Why would you want to use something like this? for example, because you have WD use a server to dial out, but then you trunk calls from the main server to another one holding the actual queues being used. Or you use WD to dial out, but actual queues or end-points are on a server that is not Asterisk based, and you want to control how quickly WD is dialing out.

The way it works is that, on agent status changes, you send WD:

- A list of agent channels that are currenty IDLE, DIALLING, BUSY or PAUSED
- A list of calls that are queued on the remote system (that is, not yet assigned to an agent, but about to)

If you use reverse dialling, it is important that the channels you publish let the local Asterisk reach the remote agent, so they must go through a trunk (e.g. `SIP/destinationserver/123`).

If you just use direct dialling, you do not need to tell WD about the exact agents being involved; you just need to tell it how many agents are in whatever state. The API offers a simmplified call format to handle this common scenario.

The biggest challenge you face in using API updates is making sure that you send WD an updated picture on time, so that it can make sensible decisions at the right time.

For example, if you tell WD that there are two free agents, but fail to notify it that those agents got busy, it will keep on trying until agents do get busy - that is, never. This may or may not be what you are looking for.

Dialing modes

Depending on how you set up your end-point, WombatDialer offers different dialing modes.

- *Direct dialing* is the default and works with extension and queue end-points. When doing direct dialing, WD will first try to connect the callee and will then route the call to a local extension, that may or may not be a queue. This is fine for delivering voice messages or for situations where any agent can process any call. When working with call queues, this dialing mode might introduce a slight delay, as the call has to be answered by the agent - not usually a big deal, but it might be there.
- *Reverse dialing* has WombatDialer connecting the agent first and then placing the call. This is less effective in terms of agent efficiency than direct dialing, as the agent has to wait for the call to connect. The big advantage is that when the callee picks up, he is immediately on line with your agent. WD picks an agent at random from the available pool in order to share the load with all available agents.
- *Reverse preview dialing* has the agent reviewing the call before the call is made. This may happen through the API, or by using a special page that WD offers. A number is "reserved" when an agent is to dial it, and the agent has 10 minutes to accept it (and have it placed) or skip it. Calls skipped are marked as such and not retried, unless you set up a reschedule rule to have them retried. Calls for which a decision is not made within 10 minutes are simply returned to the pool of callable numbers.

When using reverse dialing (vanilla or preview) WombatDialer uses a queue to keep track of agent presence. This lets you manage log ons, log offs, agent pause and unpauses the same way you would for inbound queues. Also, as agent state is shared across multiple queues, you can have agents working on multiple queues at once. In reverse dialing, through, calls are NOT connected through queues, as WD decides which agent is to receive which call; in order to do this, the agent channels are connected directly without going through the queue.

When using preview dialing, the agent must reserve a call before it can be placed. In order to do this, WD offers a rich API for your integration software see Preview API.

WombatDialer includes a simple preview panel that can be called as a web page and lets you reserve calls and open external URLs for previewing - typically you will use it to preview a call in your CRM software before it goes live. See the Preview Page for more information.

A simple example tutorial on preview dialing can be found in Preview dialing with Elastix, WombatDialer, QueueMetrics and SugarCRM.

Safety measures when running in reverse dialing

Though most calls will correctly be handled in Reverse or Reverse Preview mode, it is always possible that a call gets "stuck" because the agent that was to serve it was not able to complete it or to answer the phone (this often happens with remote agents becoming suddenly unavailable because of network problems, or because their soft-phones crash. It's wise to make plans in advance).

In this case, WD has a set of built-in safety measures, including:

- If a call stays reserved for more than 10 minutes, it is unreserved and put back in the pool
- If an agent is not able to answer to a call that they asked for within 60 seconds, the call is closed with state RS_NOAGENT. You should usually add a reschedule rule in order to "recycle" those numbers.

Testing end-points

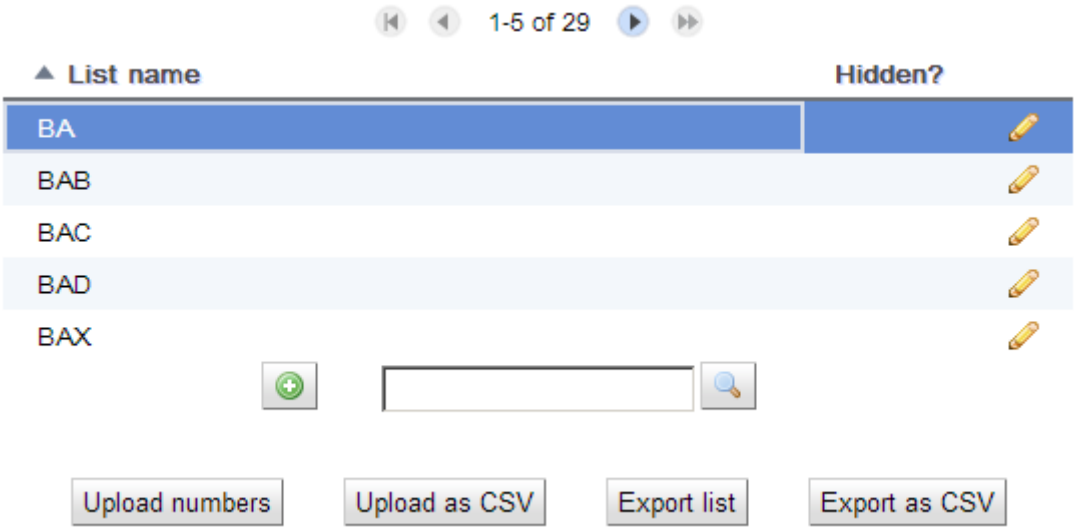
The **Test connection** button allows you to check your End-point. In order to do this, WD:

- first, makes sure that there is a working Asterisk connection
- if the EP is of type queue, checks that a queue with the name defined in **Queue name** actually exists on the server
- then it tries dialing a channel on the end-point, as defined in "extension" and "context". If all goes well, the number is called and a whistling "milliwatt" sound is played.

When running a test on a queue, make sure that your agents are aware that a test call playing "milliwatt" will be sent to one of the available agents.

Call lists and call records

Call lists are sets of numbers (or more technically, sets of call records) ready to be dialed.



After being created they cannot be deleted, so you will end up having quite a number of them. In order to avoid having too many of them as visible/selectable items, it is possible to set a "hidden" flag so that they disappear from normal views.

As happens with the campaigns, you can enter a tilde "~" symbol in the search box to see all lists, including hidden ones. All searches happen on both visible and hidden lists.

If you add calls to running campaigns through the APIs, WD will create a hidden list called "CampaignName/AUTO" to which all of your new items will be added to.

> This list behaves a bit differently than ordinary lists when run - see Automatic lists.

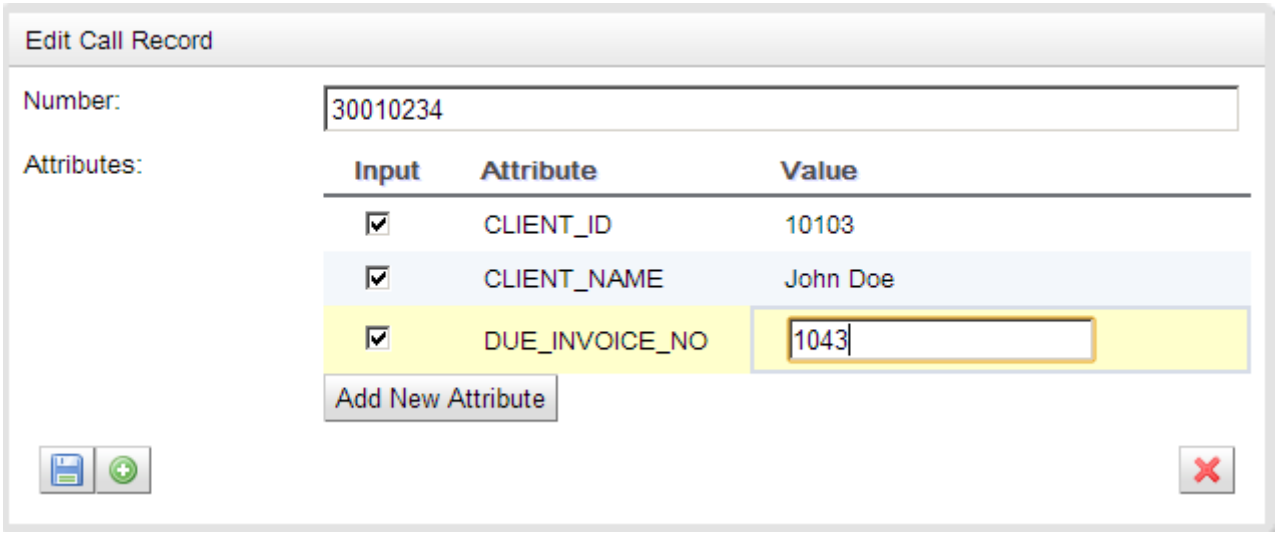
It is possible to import and export data from lists through the GUI, and it is possible to create new lists based on the final state of existing campaign runs from the Reports page.

A call list will store multiple instances of the same number and will dial them in succession; so if you upload the same set of numbers multiple times, you get them called multiple times.

As with other WD resources, call lists can be protected by Security keys.

Call records

A call record represents a single number to be dialed by WD.



- **Number** is the number to be dialed
- **Attributes** are an (optional) set of variables that are sent or read from Asterisk during the call processing phase.

Attributes make WD very powerful: **input** attributes are sent to Asterisk along the number and are available at the dialplan level as standard channel variables, and can also be used to compose the internal or external caller-id. **Output** attributes instead are values set by Asterisk on this call and are meant for data collection.

You can manually edit the number or the attributes from the web GUI, though you cannot delete an existing call record.

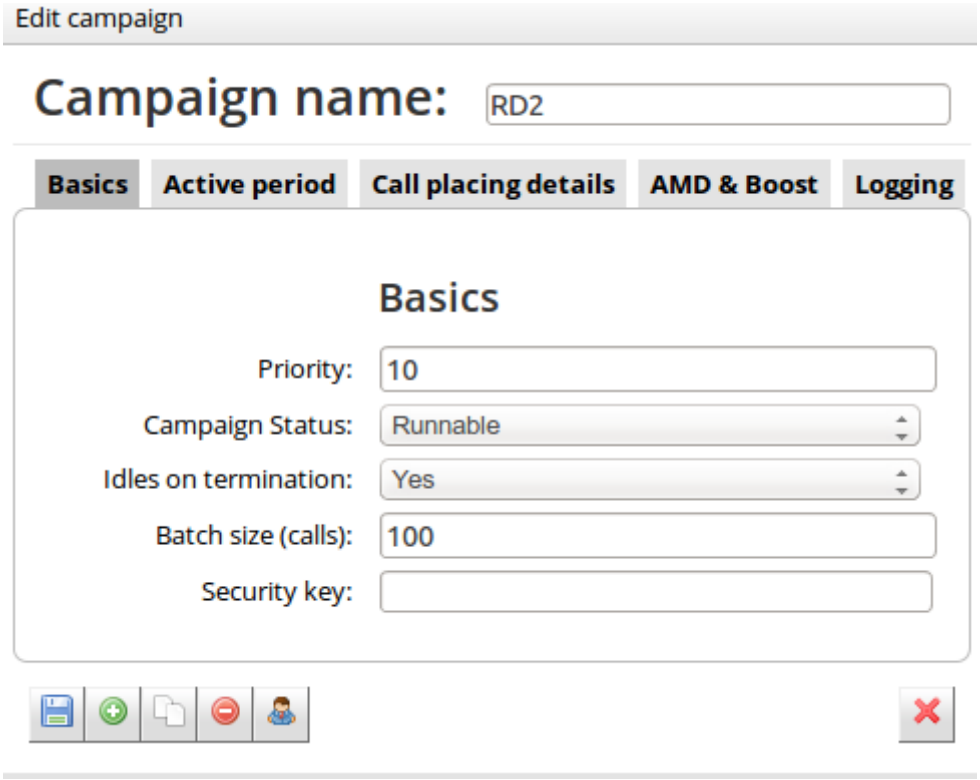
Campaigns

A campaign is the basic unit of work of WD. It behaves as a template for running an actual campaign, that we call a Campaign Run. A campaign defines:

- a set of general properties
- a set of trunks

- a set of end-points
- a set of list (zero or more). They are further classified as normal or black lists.
- a set of reschedule rules
- a set of disposition rules
- a set of opening hours

A campaign can only be run one at a time - before running it again, you must make sure that any actual runs are terminated. Trunks, end-points and list are instead shared entities - you can have multiple campaigns using them at the same time.



If you are running a campaign that does not appear to work, remember to check the Dialer state page that has a deeper view of the decisions WD takes for each specific campaign.

The following properties are defined for a campaign:

Basics

- **Name**: a name to display in WD
- **Priority**: the relative importance of this campaign against any other (see below)
- **Status**: this field lets you decide what to do with the campaign:
  - RUNNABLE: this campaign can be run
  - CLOSED: this campaign cannot be run, but it is still visible in the default editor
  - HIDDEN: this campaign is not visible anymore unless you set the flag "Display hidden campaigns"

    - ERROR: this campaign is marked as an error. Cannot be started but visible.
- **Idles on termination**: whether this campaign terminates when out of numbers, or will live on waiting for more numbers to be added via the API
- **Batch size**: The number of calls to be placed that are cached in memory when reading from the database. They should be roughly 2x the size of outgoing trunks. The higher your batch size, the less disk accesses to find new numbers are necessary; the lower it is, the more "snappy" Wombat is in picking up new numbers or reschedules.
- **Security key** is the optional key that protects this resource.

Active Period

- **Start active period**: the time of day after which this campaign can place calls
- **End active period**: the time of day before which this campaign can place calls
- **Allowed Days of Week**: the days of the week that the campaign is allowed to run on

Call Placing Details

- **Answer timeout**: if the number dialed does not answer within this period (expressed in milliseconds), consider the call to be a NOANSWER
- **Forced closure**: the maximum length of this call, in seconds. If reached, the call is forcibly closed and set to status TIMEOUT. Set to zero to turn off.
- **Dial CLID**: the caller-id to use for this campaign. This may be overridden by your provider.
- **Agent CLID:** the caller-id that will be set on the end-point. This might for example be the internal code of the campaign, or the name of the called person. If the Agent CLID is not displayed correctly, try adding a short delay between the start of the EP leg and the queuing, so that WD has a bit of time to set it up before the call is queued.
- **Dial account**: the account code that will be used by Asterisk when writing CDR records
- **Dial presentation**: The number to set as "call presentation", that is the number that callees are expected to see as the caller-id. This may be overridden by your provider. Currently not used - use Dial CLID instead.
- **Auto-pause**: if this is set and the campaign has queue end-points, each agent will be automatically paused when the call terminates (so that they can process their wrap-up activities). The agent will then have to manually unpause when he is ready to take a new call. (Note: this is not currently implemented).
- **Campaign variables**: a set of variables defined for this specific campaign. They should be in the format "A:B,C:D" to set a dialplan variable "A" to "B" and "C" to "D".

Amd & Boosting

- **Boosting model**: turn this on to enable Adaptive mode (see Adaptive boosting) or leave to OFF for manual mode (controlled by the default boost factor and the manual boost factor on the Live page).
- **Initial boost factor**: This is the extra boost factor applied when a campaign runs. This value can be modified dynamically through the Live page.

- **AMD FAX tracking mode**: Determines which AMD/FAX tracking mode is to be used. If AMD or fax are to be detected, the dialplan variable AMD_MODE is set to the required mode. See AMD and fax tracking.
- **Extra AMD settings**: A string of additional parameters to be used when doing AMD detection. This is translated into the dialplan variable AMD_EXTRA. You may or may not want to use this.
- **Audio file to send on AMD**: The name of a file to be sent to the client on AMD detection. This is passed to the dialplan under the name AMD_FILE
- **TIFF file to send on fax**: The name of a TIFF file to be sent to the client on fax detection. This is passed to the dialplan under the name FAX_FILE

Logging

- **Additional logging**: set to QM_COMPATIBLE to have the campaign log to *queue_log* on the Asterisk server
- **Alias for logging**: if this field is set, this is the name that this campaign will be logged under on the *queue_log*. If empty, the name of the campaign is used. This way you can have multiple WD campaigns log as the same Asterisk queue.
- **Attributes to be logged as QM variables**: which attributes (if any) are to be sent to QueueMetrics. See Logging of attributes for a detailed explanation.
- **HTTP notification URL**: the URL to be called when a call has a state change in WD
- **Send campaign events by e-mail**: whether WD should send life-cycle notifications by e-mail. Can be set to:
  - NO: No notifications.
  - ALL: All campaign life-cycle changes.
  - FINISH: Send only on campaign completions.
- **E-mail addresses**: a set of e-mail addresses to receive notifications for this campaign.

A campaign has a **priority** so that you can have multiple running campaigns at the same time. Priorities are taken into consideration from the lowest to the highest, where each priority level has a go to fill in all available channels; if some available channels are left over, campaigns with a higher priority number are processed. For example, imagine you have a campaign of priority 1 linked to a queue (for human outbound) and then a quality review automated campaign running at priority 10. If there are available agents, it is just natural that the campaign at priority 1 has its go first at placing calls. But if for example some of your agents are paused, then not all outbound lines are used - in this case, they are used by the campaign at priority 10. As soon as your agents go back on line, calls for them are dialed first.

If you have multiple campaigns at the same priority level, they are offered a fair chance of placing calls, so you would expect them to place roughly the same number of calls if calling an homogeneous set of callees. In practice the numbers may differ based on call length, call completion ratio and average answer times.

You can define an **active period** for calling, so that you can e.g. tell WD to place calls between 9 AM and 4 PM of working days. Any reschedules will be placed only in the active period. If the campaign is in its active period, then any Opening Hours linked to the campaign will be processed. They will

be processed in the order they are defined, and in case of no match, the value from the last rule will be used to decide whether the campaign can run or not. See Opening Hours.

There is no guarantee as which trunks and end-points will be chosen when a campaign is running. Call lists instead are processed in order from the first to the last.

If you want WD to send you e-mail when something happens on a campaign, you should make sure that you configured the SMTP parameters as explained in Configuring e-mail. You can have WD send you notifications for all campaign life-cycle event changes, or simply when the campaign completes.

What happens to hidden campaigns?

Hidden campaigns are removed from the editor so that you don't have to see them all of the time. They are still present on the database, and may be found again by:

- entering a search string. It will match all campaigns, including hidden ones (this way it easy to access them and un-hide them if necessary)
- entering a single tilde "~" in the campaign search box. This will display all campaigns, whether they are hidden or not.

Using attributes in Caller-Ids

WombatDialer lets you enter placeholder values in the Dial CLID, Agent CLID, Dial account, and Caller presentation fields. These values are expanded when a call is actually being connected using the values of attributes set for the number dialed.

For example, you may be dialing number 5551234 to reach Mr. White. You may upload a list of numbers setting the attribute NAME to the name of the person called, and you may want the caller-id changed when the call reaches your agents so that they see "WHITE" instead of the campaign's caller id. Or you may dial a list of numbers by setting an unique call presentation for each of them.

In order to do this, you have to specify attributes to be expanded. For example, if you set the agent CLID of your campaign to "C1 ${NAME}", agents will see on their phone "C1 WHITE", "C1 SMITH" and so on. You may use multiple variables in the same ID, so that you can pass along a practice ID, or the code used to find the person called in your CRM.

Together with the custom attributes you manually define for each number, WombatDialer will also expand:

- ${NUM} to the number being called
- ${LST} to the name of the list that the number belongs to

Adaptive boosting

WombatDialer has a zero-configuration adaptive-boosting mode that lets Wombat manage the over-dialing rate of a queue-based campaign. This way, you can start with a rough idea of the required over-dialing rate (say, 3x) and Wombat will tweak it continuously based on recent traffic, state of the underlying queue, current completion rates and nuisance calls detected.

The model is based on the fact that the quality of traffic on the same campaign, with leads coming from the same list, is strongly correlated in time; so by looking at recent data it is able to extract reasonable call patterns and adapt the current over-dialing rate.

The adaptive model is allowed to boost calls in the range from 1x to 4x; this way, if you are running a campaign with a default boost factor of 2X, the booster will have it run between 2X and 8X. On top of this, you can always use the Boosting control from the Live page to further correct the current amount of applied boosting.

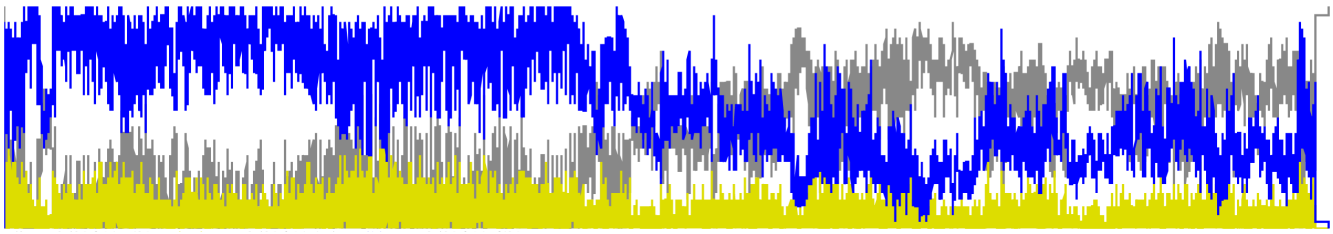
Please note that the adaptive booster will try counterbalancing any change you make manually, so you should use the manual toggle only to "speed up" the adaptive convergency behavior in case the dialer should not be aggressive enough at pursuing it.

As with all adaptive and predictive models, the quality of its predictions is strongly linked to how much data it has and how many calls are being made at once, as higher numbers tend to make the system more stable and smoothen its variance.

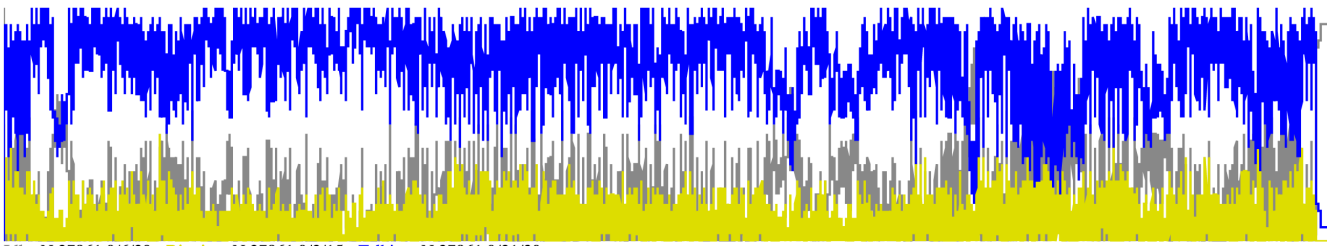
It works well if you have a good number of calls (dialing at least 40 channels simultaneously) where it can usually get you an agent occupation of 60% to 80% with a nuisance rate of about 1-3%, and it scales back reasonably well for smaller systems with about 10 channels. Of course these figures may be significantly different based on your traffic patterns.

An adaptive boosting example

The following simulation is based on actual real-life data; it shows you a situation where you have 30 agents and you are initially over-dialing at a rate of 10X, for a period of a few hours, placing a total of about 100,000 calls. The blue line in the graph is the actual number of agents that are in conversation.



As you can see, the over-dialing rate is initially adequate, but the blue line often has "valleys" where the over-dialing rate is not adequate. Especially in the second part of the campaign, traffic patterns change and Wombat is not dialing effectively anymore. At the end of the campaign, agents have an utilization of about 56% and a nuisance rate of about 3%.



By using the adaptive model, you can see that the blue line is consistently higher, and that it

matches trends in calls by counterbalancing the quality of traffic. By the end of the campaign we have a total utilization of 73% (+30% over the basic model) with a nuisance rate of 4%.

This example was chosen because the quality of the leads was intentionally pretty bad, and this is the case where you have the highest improvement in using a predictive model.

Reschedule rules

It is a fact of life: most calls placed on an outgoing campaign are destined to fail. Maybe the user is not available, maybe your provider has a temporary failure, or maybe your PBX (or even WD itself!) crashes while calling. It is advisable to take this into consideration when programming a campaign. For example, you could say that:

- if a number is busy, you retry two times after 5 minutes each
- if a number does not answer, you retry two times after 30 minutes each
- if a call has a technical glitch and ends in error, or is lost due to a PBX crash, then retry it once in 10 minutes

The number of retries is computed after the call is first attempted - so if you have a retry rule of 2, the call is first tried once and then retried twice, for a maximum of three times if it goes wrong every time. All retries are attempted in the active period of the campaign - so if a call is rescheduled in 20 minutes at 5:50 PM and the campaign is not allowed to run after 6 PM, then it is retried the next day.

WD in general tries first to obey any applicable reschedules and then fetches new calls form call lists, so you can expect the retry period to be quite accurate in most scenarios. Still there is no hard guarantee that a call will be placed at exactly the time it was rescheduled for.

Parameters are set as follows:

- **On status**: the call status this rule applies to
- **With custom status**: the custom status to consider (see below). Custom statuses can be set through the API - see Controlling WD from Asterisk.
- **Max Attempts**: the maximum attempts this rule applies to. If zero, it always applies and the reschedule counter is not incremented (see below).
- **Retry after**: the number of seconds to retry after
- **With mode**: the way to compute the retry period

Each rule is evaluated against the **current** number of attempts for the call, so it does not take into consideration the statuses for previous attempts.

When rescheduling, you can set the **mode** to FIXED or MULTI. In fixed mode, if you set the retry time to 5 minutes, it tries after 5 minutes at every attempt. In multiplicative mode, the retry period is computed multiplying the number of the current attempt by the number of attempt it's trying - so it would be 5 minutes on the first attempt, 10 minutes on the second, 15 minutes on the third and so on.

If a call has a normal completion or is over the maximum number of retries, then it is not

rescheduled. You can look-up the status of the last attempt in order to know why it was not rescheduled.

Writing advanced reschedule rules

It is valid to have multiple reschedule rules that pertain to the same status code - in this case, WD will find the rule that matches. For example, imagine we have two rules:

```
             Attempts   Retry in
             --------   --------
RS_BUSY         2         300s
RS_BUSY         5        1800s
```

Up to the second BUSY attempt, WD will retry in 5 minutes (300 seconds); from the third to the fifth, it will retry in 1800 seconds (30 minutes).

The status code considered is always the current status code; so for example given this set of rules:

```
             Attempts   Retry in
             --------   --------
RS_BUSY         3         300s
RS_NONASWER     1         600s
```

If we get a BUSY on first attempt and a NOANSWER on second attempt, as the NOANSWER retries only once, the call is not retried.

You can also have an extended status set through the APIs - if that is present on the call, the rule matches only if the extended status matches - see Controlling WD from Asterisk.

If you specify an ExtStatus for the call, then only the same ExtStatus will match. If you do NOT specify an ExtStatus, then the rule will match any ExtStatus. If you need the rule to match only on an empty ExtStatus, then you must set it to *$*.

Handling technical errors through Reschedule Rules

Sometimes you do not want to treat technical errors (e.g. statuses like RS_LOST, RS_NOAGENT and RS_ERROR) as normal recalls: you want to keep track of the number of recalls and handle them accordingly only for calls that actually try to reach a person.

In this case, you can enter zero as the maximum number of attempts; the rule will match no matter how many recalls you are having (and will be preferred to other rules), and the number of attempts will not be incremented on the next recall.

When you run a report for the number dialed, it will then be possible to see a number of calls having the same "attempt number"; this is correct, it means the rule matched and was applied successfully.

As these calls do not increment the recall counter, this feature can send calls into

> an infinite loop; therefore it is important that you only handle **transient** technical errors with this, and that the reschedule time is high enough that the problem will likely be resolved before the call is attempted again.

Disposition rules

Disposition rules are like Reschedule Rules; the main difference is that while a Reschedule Rule is applied on each call tried, a Disposition Rule is matched only when the call is not rescheduled further.

For example, let's say that you are dialing out on a campaign where there is a reschedule rule to retry twice in 10 seconds on busy. You dial a number, and the number is busy; it is rescheduled in 10 seconds, retried and it's busy again; it's rescheduled again, and again it's busy. At this point the call is "complete", meaning that it is not meant to be rescheduled again. So, the call will be matched against any disposition rules with the current status of "busy".

With disposition rules, you can have actions that are more complex than reschedule rules and are meant to interact with external systems. You can:

- Call an HTTP service of type GET or POST, with a variable payload.
- Send an email (for example, if you want someone to be notified when some rare status happens)
- Add the number to an existing list,
- Add the number to an existing blacklist. This is the same thing as adding the number to a list, but with an optional parameter to control the amount of time the number is to be blacklisted.
- Pipe the number to be recalled on a different run. The run must be active at the time the number is added.

Multiple rules may match the same call, so for example you might be sending two separate HTTP notifications and an email, or you could reschedule the number to be recalled and add it to a new list at the same time. Matching happens on the basis of the (last) dialing state. If the rule specifies an extended state, it is to be matched as well.

When a Disposition Rule is matched, sometimes it is useful or necessary to access some information from the call and put it in the request, for example to specify the number called in the email body. WombatDialer offers a number of variables that you can use for this purpose:

- NUM is the number dialed
- N_RETRY is the number of retries made on this call
- LIST is the list name
- STATE is the current state
- EXTSTATE is the extended status (if set)
- CAMPAIGN is the name of the campaign that this call was dialed upon
- RUNNAME is the name of the current campaign run
- ATTR is a prefix that lets you access call attributes. They can be inbound or outbound attributes.

So if you set an HTTP GET URL of *'http://server/page.php?num=${NUM}&var=${ATTRV}'* will be rewritten with the current number and the current value of attribute V.

When doing a disposition to a list or a different run, you can control which attributes are to be set on the new number. You can decide if you want ALL, NONE, INBOUND or OUTBOUND attributes; plus you can manually specify a set of attributes which values will be set on the new call.

Blacklists

WombatDialer has the concept of Blacklists; they are lists like any other but are used to collect numbers that are not to be called. A discursive explanation of how they work and interact with the rest of the system is available at Understanding blacklists .

When WombatDialer is loading numbers to be dialed, they are checked against all black lists defined for that campaign. This is done automatically and behind the scenes; if a number is found, it is logged as dialed in state `BLACKLIST` without actually trying to dial it.

Blacklists are checked dynamically when a number is first scheduled; so if you add numbers to a blacklist while a campaign is running, new numbers to be scheduled will be checked against the blacklist.

Starting with WombatDialer 19+, black lists are checked two times:

- every time a number is fetched to be dialed, and
- just before actually dialling it, every time this is supposed to happen.

This way, if you add a number to a blacklist, future retries may be blacklisted even if the original call was not.

You may have multiple blacklists on a campaign; their order in unimportant. All numbers are checked against all defined blacklists at once.

A number may be added to a blacklist up to a specific point in time; this means that the number will be a valid match for the blacklist only until the date passed is in the future. In order to do this, you need to set a call attribute called `BLACKLISTED_UNTIL` with a valid date in the format "YYYY-MM-DD.hh:mm:ss" or "YYYY-MM-DD". You may also have an expiry date computed for you by using a Disposition Rule to add the call.

Note that if you use the shorter date format, the number is supposed to be blacklisted until the midnight of the given day. If the date format is not valid, the number will be blacklisted forever.

It is perfectly acceptable to add a call multiple times to a blacklist, each time having a different expiry date. The number will be blacklisted up to the maximum date specified.

If you need to cancel a blacklisted number, you might remove the number from the list or, better, add a special attribute `BLACKLISTED_CANCELLED` that we suggest to use to record the reason why blacklisting was cancelled. This attribute only happens to the specific number, so if the same number is present multiple times, then each of them must be cancelled separately. See also Cancel

numbers from a blacklist.

When using blacklists, we suggest using an attribute to store the reason why a number was blacklisted, and a reson why the number was removed from a blacklist. This will prove invaluable for your future self when tasked with auditing why a call was or was not actually placed.

Automatic lists

You may add calls to a running campaign through the API; those numbers will be added to a list called `NameOfCampaign/AUTO` that is created automatically. This list is just meant as a container to store those numbers and is always present after you add at least one number through the API.

Automatic list are very useful, but they come with a couple of warnings. The first is that **an auto list is never explicitly linked to a campaign** – it just exists. It acts like a kind of placeholder for calls that were scheduled dynamically – as each and every number has to be a part of a list, your automatic list is the place they will go to.

The second is that the **content is not fixed** like a call list you upload from a file - it grown as numbers come in while a run is in progress, so it's final content is the one at the point you stop adding a new number to it.

This has a couple of consequences that may - but most likely will not be - what you want:

- If you add an existing automatic list to a different campaign, it is executed as a normal list, up to the point when it contains no new numbers. At that point, the campaign will consider it done and stop running it. If you add some new numbers after the list has been completed, the campaign will not process them (exactly the way it would do with a normal list to which you add some numbers after it was completely run).
- An auto list must not be added explicitly to its owner campaign. If you do, what happens is that the campaign will run (again!) all the numbers that are on the list. And like in the previous case, any numbers that were not included at the point the list was terminated will not be run again.

So, in general, it is OK to add an automatic list from a closed campaign to another campaign, but it's not wise to add an automatic list for a campaign that is still running to another campaign - because you definitely want to know in advance which numbers will be processed and which ones won't.

What you most definitely don't want to do, is add explicity, at setup time or during its execution, an automatic list to the very campaign it belongs to.

As a general policy, we suggest to **never add automatic lists to campaigns**. Don't do it. If you need to recall all numbers in a different campaign, create a new list out of the same numbers and use that instead.

AMD & Fax detection

WombatDialer allows you to track answering machines (AMD) and faxes. Following the general WombatDialer philosophy of leveraging the capabilities of your PBX, WombatDialer sets a series of

additional dialplan variables that are to be used in order to detect them and react accordingly. This way:

- You configure AMD and/or fax detection on your PBX
- When either detection triggers, you have variables that tell the dialplan how to handle those cases

In general, we suggest using an extended status of "AMD" when an answering machine is detected, and "AMDSENT" when a message is left on the machine; and "FAX" when a fax is detected and "FAXSENT" when a fax is successfully sent.

It is also wise to offer your agents a quick blind transfer hook that lets them transfer calls to AMD or FAX routines in case the detection went wrong. See Call transfers.

Working with AMD and fax can be tricky. Make sure you can talk to an Asterisk consultant with a proven experience before running a large-scale campaign.

Detecting answering machines

Asterisk ships with an application called AMD that will analyze audio and try to guess whether the other party is a live person or an answering machine. Keep in mind that the call does not "go through" while the analysis is in progress, so there is a definite trade-off between a long and accurate detection versus a quick but less accurate detection.

AMD receives a set of parameters to discriminate what is an answering machine and what is a live person, so you may want to experiment a bit to find a setup that works for your country and for your callers.

A good starting point would be to set in amd.conf the following parameters:

```
initial_silence            = 2500
greeting                   = 1500
after_greeting_silence     = 300
total_analysis_time        = 5000
min_word_length            = 120
between_words_silence      = 50
maximum_number_of_words    = 4
silence_threshold          = 384
```

It is also customary to send a background tone (be it blank or a beep) to the called party to help set-up the line on SIP circuits. This way the accuracy of AMD is reported to be increased considerably.

So you would:

- Play audio, e.g. via *Background(beep)*
- Run AMD, optionally passing the contents of AMD_EXTRA if you want to override the default parameters
- Check the status of the dialplan variable AMDSTATUS. If set to "MACHINE", go to the AMD

processor; else route to a live agent. You may want to log the AMDCAUSE as well; it contains an explanation of the reason why Asterisk decided the call was of the specific kind.

- On the AMD processor, set the extstatus to "AMD", so that Wombat logs this call as an Answering Machine
- Wait for silence (so the greeting message is terminated)
- Play the audio file defined in AMD_FILE
- Set the extstatus to AMD_SENT
- Hang up.

An example is available in Answering-machine detection.

Detecting faxes

Asterisk is natively able to detect faxes on DAHDI and SIP channels; all you have to do is tell it to run the detection engine.

- For DAHDI, you have to turn on the option "faxdetect" in chan_dahdi.conf by setting it to "incoming", "outgoing" or "both".
- For SIP, you need to turn on "faxdetect" in sip.conf. Valid options are "cng", "t38" or "yes" for both.

When a fax is detected, the dialplan will jump to the extension "fax". At this point:

- you set the extstatus of the call to "FAX" in Wombat (so it can be traced)
- you send a fax by issuing the command *SendFAX(/path/to/FAX_FILE,d)*, where FAX_FILE is the file name set through the GUI
- you set the extstatus to "FAXSENT" in Wombat
- you hang up the call

The file must be a valid TIFF file that is compatible with Asterisk. Digium's "Fax For Asterisk Administration Manual" documents a process for converting a PDF file into a TIFF using commonly available Linux command-line tools.

Campaign runs

Campaign runs are real, out-calling instances of campaigns. You start them from the Live page, by selecting one of the available campaigns. They are named after their parent campaign and the time when they were started.

The system displays a set of information on the Live page:

```
Campaign name: C1
Started at: Wed Oct 17 15:54:22 CEST 2012
Current state: COMPLETED
Priority: 10
```

```
Calls placed: 40 - Items in call cache: 0
Calls terminated: 10
Life-cycle termination rate: 25% - Reschedule rate: 75%
Est. remaining calls: 0
Running for: 00:00:05 - Estimated completion in: 00:00:00
Attempts per hour: 72720 - Completions per hour: 72720
High-water mark: 20 in L2
```

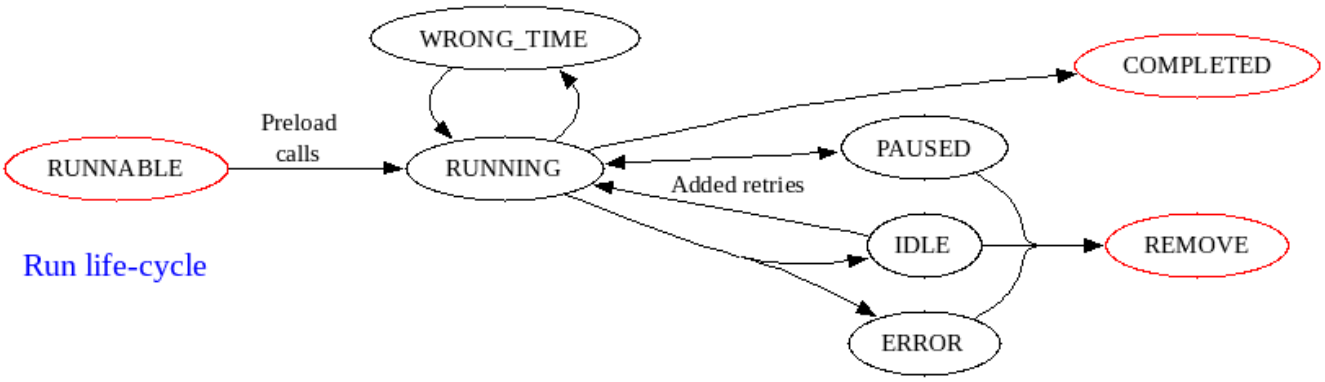
The run name is actually made of the Campaign name plus the time it was started. This uniquely identifies a run in the system.

The other parameters are as follows:

- **Current state**: is the state the run is in (see below).
- **Priority**: is the campaign's priority
- **Calls placed**: is the total number of call attempts made
- **Items in call cache**: is the number of calls currently held in the hopper plus any open reschedules
- **Calls terminated**: is the numbers of calls that have either been successful or gone through the last possible reschedule, so they will not be retried
- **Life-cycle termination rate**: is the percentage of calls that are not to be retried (terminated)
- **Reschedule rate**: is the percentage of calls that are to be retried
- **Est. remaining calls**: this is a rough estimate of calls that remain to be placed. Might be rather inaccurate - consider it only a basic indicator that will converge to zero as the run terminates.
- **Running for**: is the total time that this run has been going.
- **Estimated completion**: tries to display the remaining time to completions. This time may actually vary strongly from what is displayed depending on what happens during the campaign. Estimates will be produced after a few calls have completed.
- **Attempts per hour**: its the average number of calls attempted per hour on this run
- **Completions per hour**: is the average number of calls completed per hour
- **High-water mark**: the last call record added to the cache

A run's life-cycle

When a run is first started, it goes through a set of stages.



Initially the run will be made RUNNABLE, WD will prepare to run it and will put it in RUNNING state. A run stays in state RUNNING as long as it has retries to complete or calls not yet placed. When a RUNNING campaign is out of the allowed time period, it is put to WRONG_TIME; from here it goes back automatically to RUNNING state when time conditions (hour and day of week) are successfully matched.

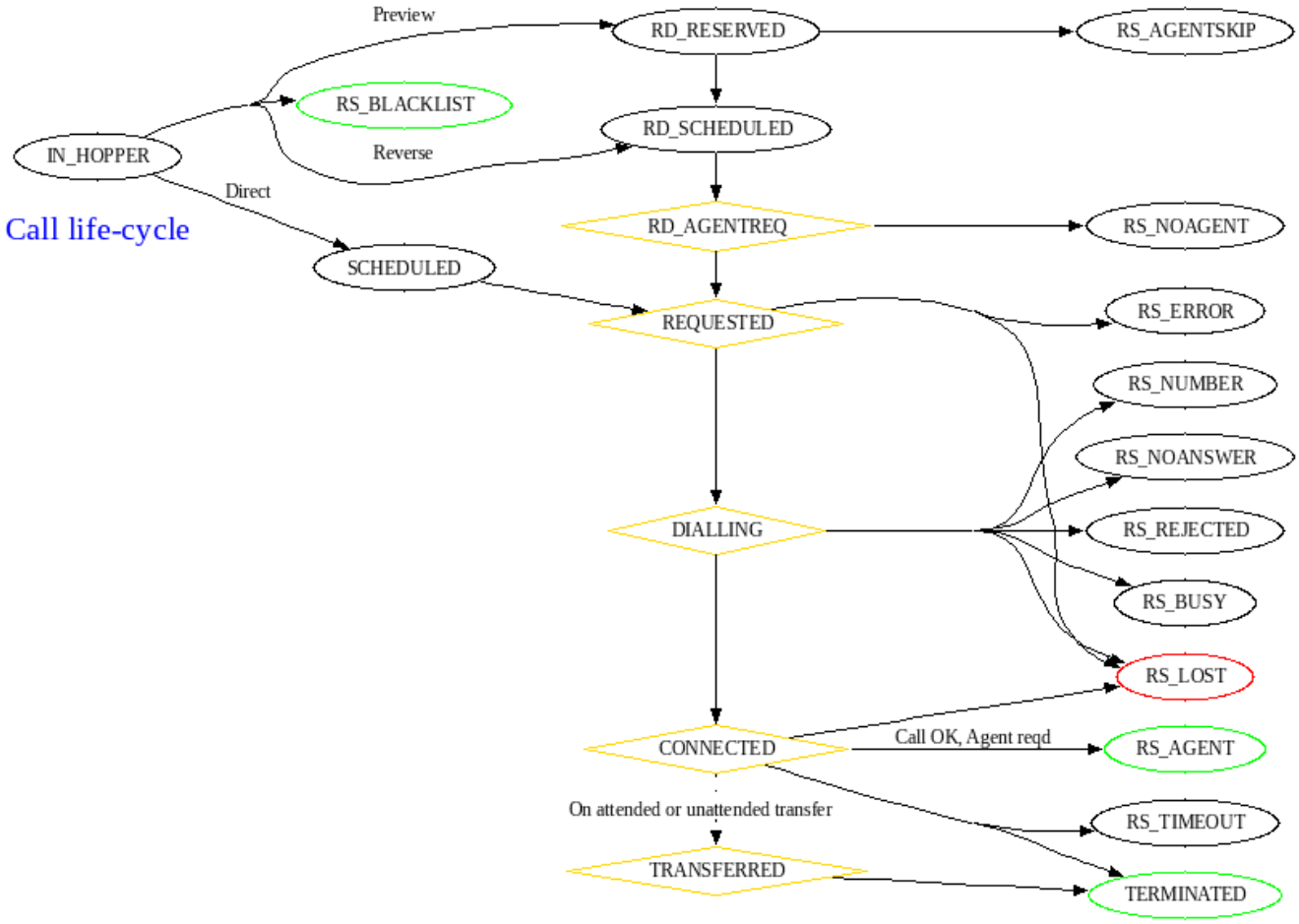
When out of calls, a run can either COMPLETE or become IDLE; when IDLE it waits for new calls to be added through the API and goes back to RUNNING mode to process them.

A RUNNING campaign run may be manually PAUSED and from PAUSED it can manually be made RUNNING again.

If a run is no longer needed, it can be manually set to REMOVE; when in REMOVE status the run is terminated and cannot be restarted.

A call's life-cycle

When a call is started, it is first loaded in a cache called *hopper* that contains calls that are to be dialed soon. This way it is not necessary to consult the database for each and every call to be made.



When WD is about to place the call, it marks it as SCHEDULED and sends it to Asterisk for processing; if all goes well, it then goes from REQUESTED to DIALLING to CONNECTED and then TERMINATED.

Of course it may not be possible to start the call (so you get BUSY, NUMBER and NOANSWER states), or the call might be ended by WD because it exceeded the maximum allowed duration (TIMEOUT).

When using reverse dialing, a call starts its life being RD_SCHEDULED, so that the agent can be called. When this happens, the call progresses forward. In preview mode, the call is first RD_RESERVED and when the agent approves it, it is placed.

Initial states

IN_HOPPER

Call will be placed soon - not visible to user

SCHEDULED

WD requested the call to be placed

RD_RESERVED

When preview dialing, an agent has reserved this call but has not processed it yet

RD_AGENTREQ

When doing reverse dialing, WD is connecting the to an agent before the call is actually placed

Call processed

REQUESTED

The request was sent to Asterisk for processing

RD_REQUESTED

Asterisk is processing this call when reverse dialing

DIALLING

Asterisk confirmed the call was started

CONNECTED

The opposite side picked up the call

TRANSFERRED

Agent transferred the call somewhere else. The agent is free but the trunk's channel stays in use. This call is not controlled by WD anymore.

Error states

RS_ERROR

A technical error happened while dialing

RS_LOST

WD lost track of this call. Usually happens only on system crashes.

RS_NOAGENT

An agent that was being pre-dialed in reverse mode did not answer. You should reschedule those calls.

Calls that could not go through

RS_REJECTED

The call was rejected by the network. This is usually caused by the upstream provider returning *'Congestion*' (all circuits busy), *'Off-hook dialing*' with analog interfaces, or your upstream provider terminating a call before it's answered without providing any status code.

RS_BUSY

Number called was busy.

RS_NUMBER

Number called appears to to be invalid. Asterisk also raises this error if it cannot allocate a new channel for the call.

RS_NOANSWER

Number did not answer within the *Answer timeout* period set on the Campaign

Completion states

TERMINATED

Call completed successfully

RS_TIMEOUT

Call was forcibly closed because it exceeded the maximum allowed duration set on the Campaign.

RS_AGENTSKIP

Agent decided to skip this call.

RS_BLACKLIST

The call was skipped as the number was blacklisted.

States not already implemented

RS_AGENT

Agent requested special retry.

Call transfers

This feature is currently experimental and may be subject to changes in future versions.

This feature requires Asterisk 13 or newer, and won't work on earlier versions.

WombatDialer is able to track calls that are transferred by the connected agent. Once a call is transferred, it keeps existing in WombatDialer, but the agent is freed so they can take another call.

The agent can transfer a call using two ways:

- With a **Blind Transfer**, that is, forwarding the call to a fixed destination, for example a second-level queue. This is usually implemented using the feature code `##` in FreePBX, but your PBX may be set for a different feature code
- With an **Attended Transfer**, where the agent dials an extensions, speaks to the receiver, and either confirms the transfer or goes back to the original call. In a stock FreePBX system, this uses feature code `*2` to start the transfer and `*2` to confirm.

When a call is transferred:

- **Direct** calls keep existing on the Live page (in state `TRANSFERRED`), and are logged until their full completion (time talking to the agent plus time until call completes); still the agent is freed at the moment the transfer is confirmed and can take another call
- **Reverse** calls are closed by the moment the transfer is acknowledged, so the call length is only the one spent in conversation with the agent.

Opening Hours

It is very often needed to have a finer level of control on the opening hours of a campaign.

Typical scenarios include:

- Having different opening and closing times for a campaign during the week. For example,you might want to dial out from 9:30 to 11:30 and 13:30 to 17:00 Monday to Thursday, and only 9:30 to 13:00 on Fridays.
- Marking specific time periods as invalid; for example, public holidays.

It is also often needed to maintain and share these "opening hours" across campaigns; for example, by having one single instance of public holidays to be shared across all campaigns.

Opening Hours work by defining a set of items that must match for a rule to be considered valid.

- Rules are scanned from top to bottom, in order
- If a rule matches, its output will be passed to the campaign.
- If no rule matches, the campaign can ask a rule for a default value.

For example, let us say that we want to implement the multiple hours scenario as described above:

```
Rule  DayOfWeek      DateFrom DateTo    TimeFrom   TimeTo    State

 #1   Mo Tu We Th    -        -          9:30:00   11:30:00  OPEN
 #2   Mo Tu We Th    -        -         13:30:00   17:00:00  OPEN
 #3   Fr             -        -          9:30:00   13:00:00  OPEN
```

You could also have a separate rule based on dates to flag some days as always invalid:

```
Rule  DayOfWeek      DateFrom DateTo    TimeFrom   TimeTo    State

 #1   -              25/12    26/12     -          -         CLOSED
 #2   -               1/01     1/01     -          -         CLOSED
```

Rule 1 tells us that December 25 and 26 are always invalid; and January 1 is as well.

When running on an existing campaign, you first have to make sure that the campaign is allowed to run in the specific time period.

In most cases, you will want to keep the campaign always active and tweak associated Opening Hours instead. This lets you use campaign rules for very simple campaigns and Opening Hours for more complex scenarios.

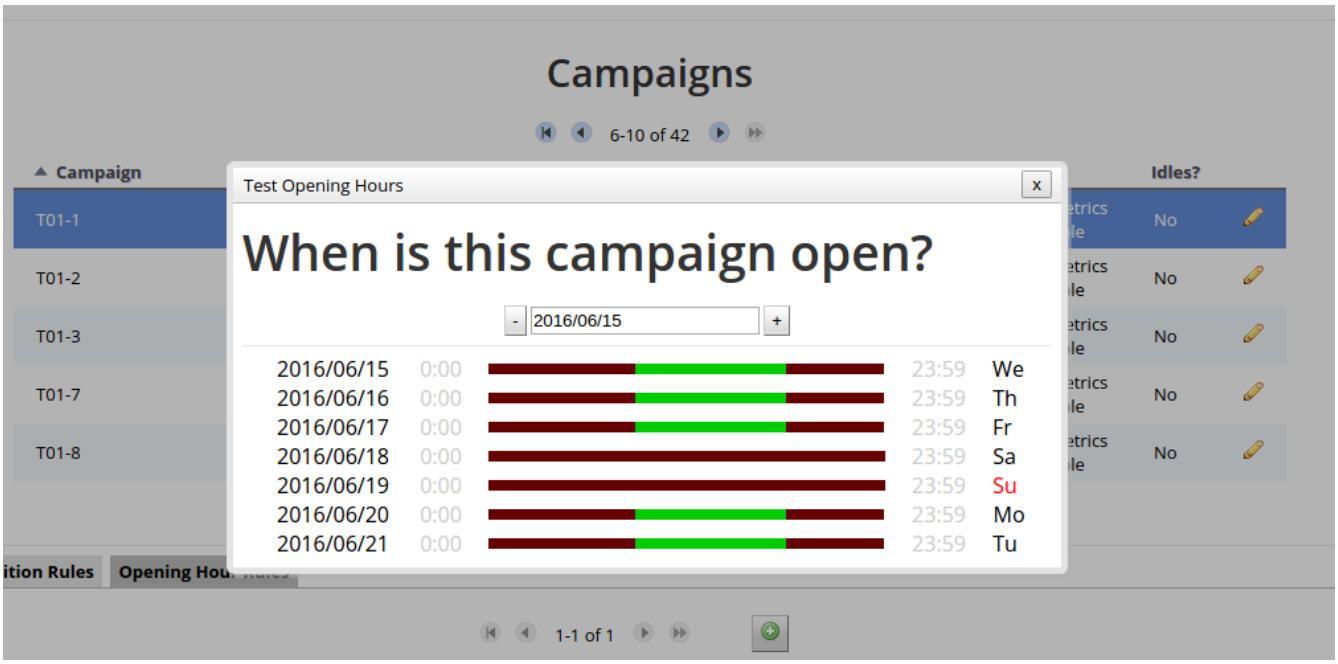
Then you would add:

- First the Opening Hours based on dates. If any of these match, we know that we cannot run.
- Then you would add the rule about time periods by day, with a default of CLOSED.

The order in which you add Opening Hours to a campaign is important, and so is the order of rules within an existing Opening Hour.

Opening Hours Inspector

Under the Opening Hours tab, inside of the Campaign Editor page, you can see a button named "Test Opening Hours". This button opens the Opening Hours Inspector, a tool designed to preview the effects of Opening Hours rules on your campaign.



As you can see in the picture, the inspector shows a colored bar for each day of the week, each color representing the state of your campaign in different times of the day. You can choose the date from the calendar widget at the top of the inspector.

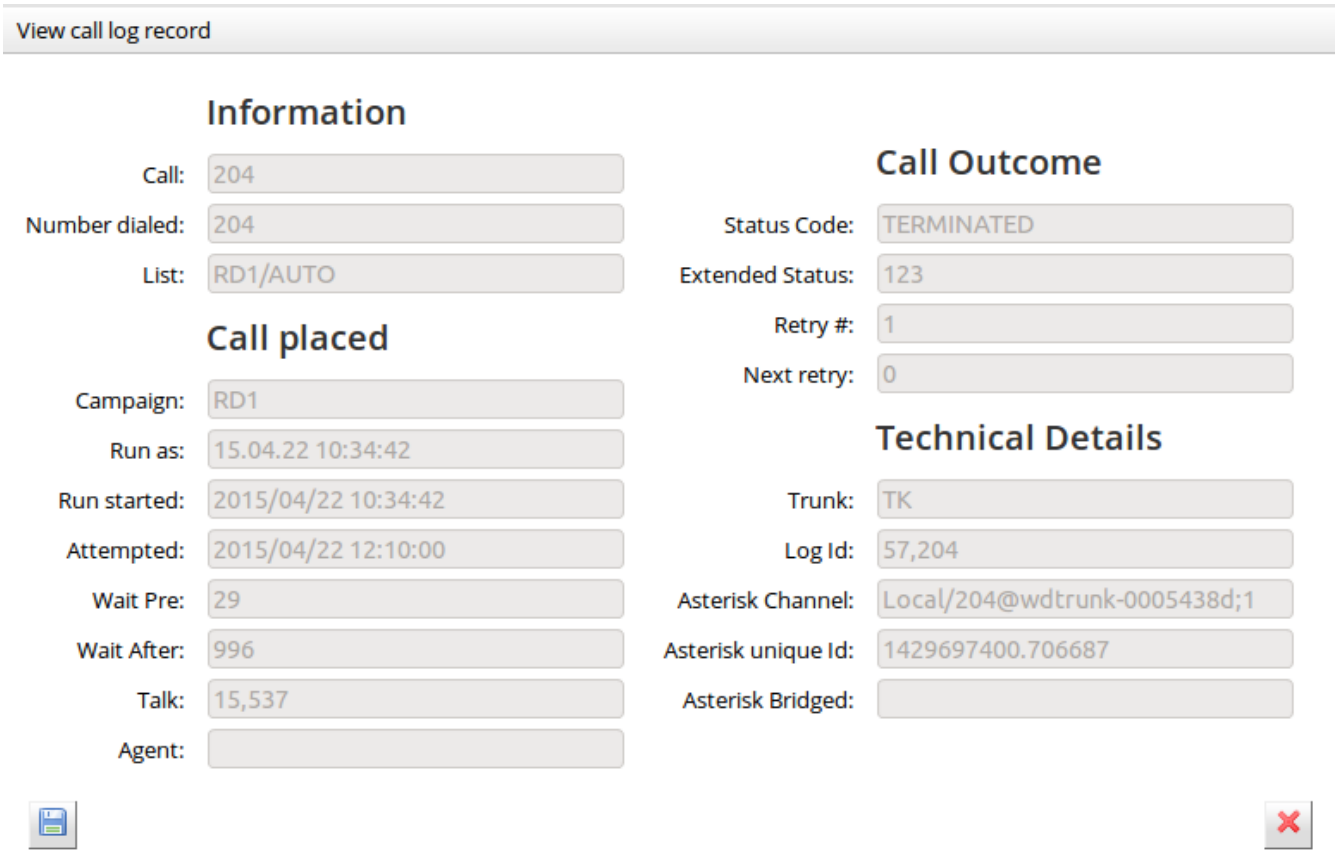
If you fly with your mouse over the colored bars, WombatDialer will show you a pop-up text with the state of the campaign, the name of the rule that is being applied and the name of the matching sub-rule. If a sub rule has no name is automatically named after a number in a sequential fashion.

Here is a list of the possible states for a campaign:

- **OPEN:** The campaign is open due to the campaign active period (no Opening hours rule applied).
- **CLOSE_DAY:** The campaign is closed due to the campaign active period (no Opening hours rule applied).
- **CLOSE_HOUR:** The campaign is closed due to the campaign active period (no Opening hours rule applied).
- **CLOSE_RULE:** The campaign is closed due to an explicit Opening Hours rule.
- **OPEN_RULE:** The campaign is open due to an explicit Opening Hours rule.
- **DEFAULT_OPEN:** The campaign is open due to the Def. Mode of the last Opening Hours rule
- **DEFAULT_CLOSE:** The campaign is closed due to the Def. Mode of the last Opening Hour rule.

Call logs

After a call record is processed, it leaves a "trail" in the form of a call log.



From here we can see:

- **Call**: this section displays the number that was called and any attributes that are set on that call record. Any output attributes (coming from Asterisk) are marked with a ">" prefix.
- **Campaign**: the name of the campaign that this call was placed on
- **Run as**: the name of the campaign run that had this call placed
- **Run started**: when the campaign run was started
- **List**: The list this number belongs to
- **Attempted**: when the call was actually attempted
- **Wait Pre**: the difference (in milliseconds) between call stages REQUESTED and DIALLING
- **Wait After**: the difference (in milliseconds) between call stages DIALLING and CONNECTED
- **Talk**: the duration (in milliseconds) of the active call, that is, while the callee was connected
- **Status code**: the call status code when it was closed
- **Extended status**: the extended status set by Asterisk
- **Retry #**: The number of retry this call was on (0: initial attempt)
- **Next retry**: When the next retry is scheduled. If set to zero, this means the call was not set to be rescheduled.

- **Trunk**: The trunk the call was placed on
- **Log Id**: The internal log id
- **Asterisk channel**: The Asterisk channel that was created for the first leg of the call
- **Asterisk unique**: The Asterisk internal Unique-Id that was used

Users and security

WombatDialer offers a powerful and pervasive security model that is similar to the one used in QueueMetrics. It is built around the concepts of users, classes and keys.

The idea is that each user has a keyring and all features are controlled by keys. Every time WombatDialer has a possible feature to show the user (e.g. editing servers) it checks whether the current user holds the correct key in its keyring. If he does not have it, the feature is hidden or grayed out.

Each user has a keyring that is composed of their personal keys plus all the keys for their class. This way you can organize multiple groups of users (e.g. administrators, monitors, etc) with different grants, and then give each specific user additional keys to fine-tune each person's profile.

A set of default users and of security keys is available below.

The security model

You can set security keys on most entities in WombatDialer - you can have them on servers, trunks, end-points, campaigns and lists. They will be enforced for:

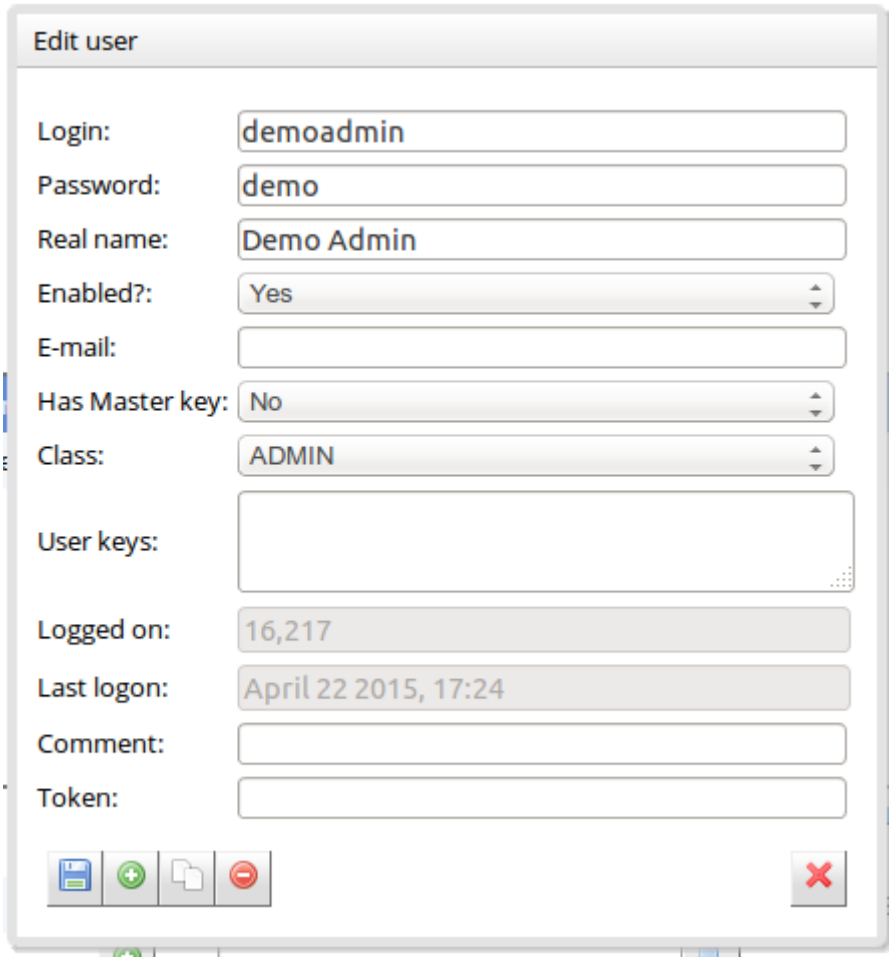
- Selection of items to be added to campaigns
- Live monitoring
- Reporting

The following rules apply:

- Elements are visible both to users holding the required key and to their creator, even if he does not hold the required key (in order to avoid "locking yourself out")
- Elements having no keys will be visible to everyone
- Key security is not transitive, that is, if you can observe a campaign, you can observe it in its entirety, even if (say) one of the trunks it uses is protected with a key you do not hold.

Users

A user is some person who can log on to WombatDialer.



They have the following properties:

- **Login** and **Password** are used to log on from the main page
- **Real Name** is displayed on the WombatDialer page
- **Enabled** may be toggled to avoid a user logging on without deleting it.
- **E-mail** The user's e-mail address
- **Master key**: if set to "yes", all security checks are bypassed. *You should only set it to yes for testing and debugging.*
- **Class** is a pre-defined set of keys that is used for this user. Any user must have a class.
- **User keys** are additional keys granted to this user. They are space.-separated. If keys are prefixed with a minus sign, they are revoked if present in their class and ignored otherwise.
- **Logged on** are the number of successful log-ons for this user and **Last logon** is the time this user last logged on successfully.
- **Comment** and **Token** are free fields you can use to keep track of your users.

Password security

WombatDialer can store passwords either in an encrypted form made of a SHA with random salt (level 2) or plain text (level 1). Encrypted passwords are not recoverable. The defaults are that all default passwords are stored as plain text, while any password changed by the user is stored in encrypted form.

This behavior is controlled by two parameters in `tpf.properties`:

```
pwd.defaultLevel=2
pwd.minAllowedLevel=1
```

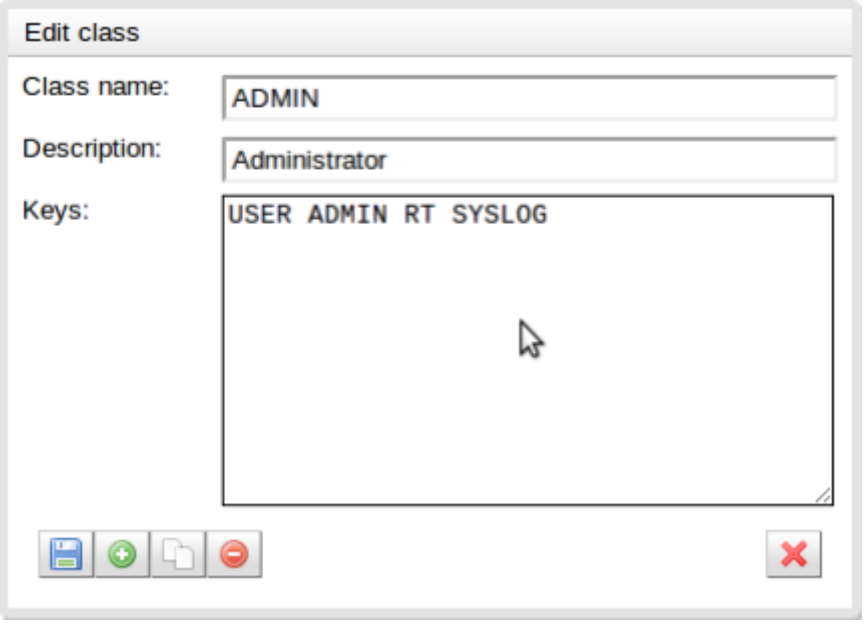
The first parameter defines the format of new passwords, while the second one decides which is the minimum level for a valid password. There is usually no need to modify the defaults.

If those parameters are missing, both of them are level 1 - unencrypted passwords.

There is really **no way** of recovering an encrypted password. If you lock yourself out of a working system, Loway technicians can be asked to connect to your server and set a temporary password for you. This is a paid service.

Classes

Classes are common profiles to be given to all users of a certain kind. So you do not have to remember which keys to give users for each specific functionality, but you can group them all together.



Any class has the following properties:

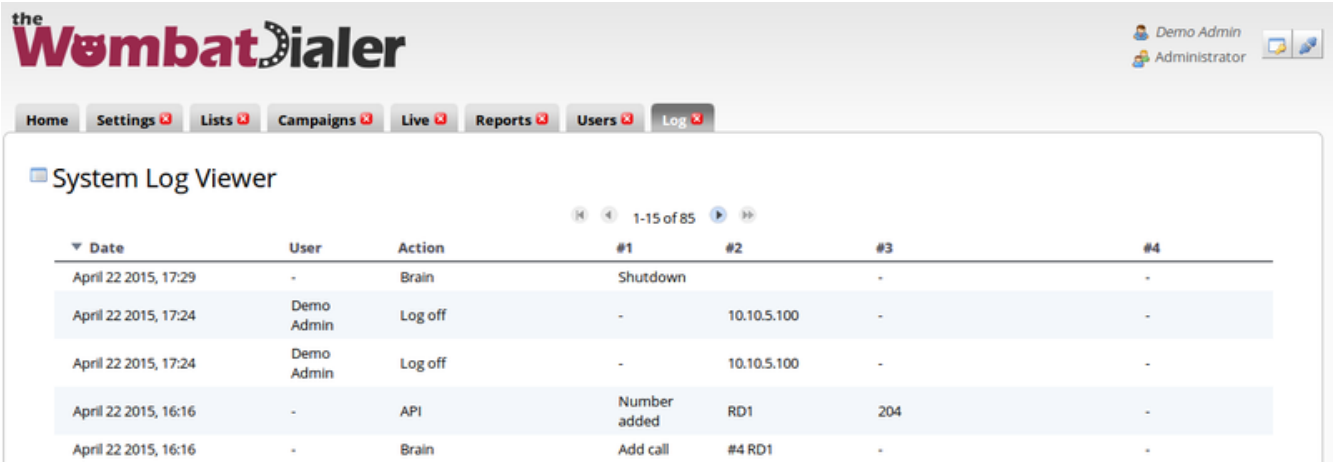
- **Class name** is the name used in the User Editor to choose the class.
- **Description** is a long version of the class name
- **Keys** are a space-separated set of keys that are granted to all users of this class.

You can create new classes than the ones that ship with WD in order to fine-tune access controls for your instance.

The system log

When something important happens in WombatDialer, it is written to the System Log. The system log tracks:

- Campaign life-cycle events
- User logins and log-offs
- System errors



Running WombatDialer

Understanding the GUI

The WombatDialer GUI is made up of a set of editors - there can be multiple editors on the same page.

An editor can:

- display a set of records (e.g a list of servers, campaigns or end-points)
- select a record in the set (usually in order to load a dependent editor - e.g. when you select a List then the editor for the numbers of that list will be enabled)
- edit a record

Using editors

Editors let you search by text in their items and let you sort data by clicking on the relevant columns.

When you are editing an item, the following functions might be available:

From left to right, these icons mean:

- Save current record
- Add a new record
- Clone the current record
- Delete the current record
- Know who created the current record, when it was created, who modified it last time and when it was last modified

Sometimes when an editor has an item selected and it's impossible to unselect it, it is then necessary to close the relevant tab and reopen it from the Home page.

Hiding and deleting entities

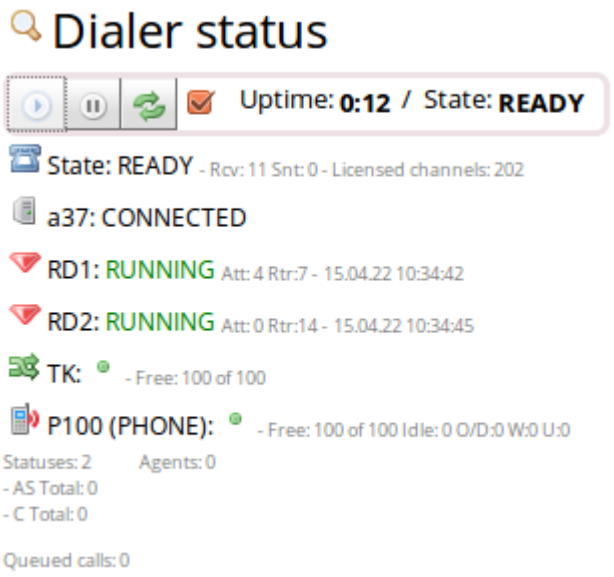
In many cases it will not be possible to delete records when you created them. This is because such records are referenced by other records that cannot be deleted - for example, a campaign that has run cannot be delete anymore, as it is referenced by the logs created during the run.

Still, it would sometimes be useful to hide things you do not currently need. So campaigns and lists have the concept of visibility - you can make them invisible so they won't appear anymore in selection menus and on the main screens.

They will not be deleted though - you can still search them by name, and you can get a complete view of all records (including deleted ones) by searching for "~".

Controlling the dialer

From the main page, you can control the status of the dialer process.



You can see how long the dialer has been used, its current status and the entities currently loaded (that is the campaigns, trunks and end-points being used). The dialer is "lazy" and loads entities only when needed, so entities do not appear until they are actually used.

The check box next to the Reload button is used to set the control in auto-reload mode; this way the current state of the dialer is refreshed automatically every 5 seconds.

You should see:

- for the dialer, the current state (usually DOWN or READY) and the number of licensed channels.
- For each PBX, whether it is successfully connected or not
- For each running campaign, the current status, the total number of calls placed ("Att") and the size of the current reschedule buffer ("Rtr"), plus the date when the run was started
- For each trunk, the total capacity and the current used channels. A green / red icon shows whether the relevant PBX is on-line or not
- For each EP, the total and used capacity. A green / red icon shows whether the relevant PBX is on-line or not.

For Queue EPs, a number of additional entries are displayed:

- You can see the number of free agents in respect to the correct queue capacity, the number of idle entities, the over-dialing channels ("O/D"), the number of queued calls that are ringing or waiting to be connected ("W") and the used channels ("U").
- If dialing is currently not allowed because the queue has too many waiting calls, the string

"BK_W" is displayed.

- A list of statuses that are reported by the queue ("AS") and a set of statuses that are computed by Wombat ("C")
- A list of queued calls (if any)
- A list of agents working on the queue, and their current status (Paused, Talking, Ringing, Idle or Error)

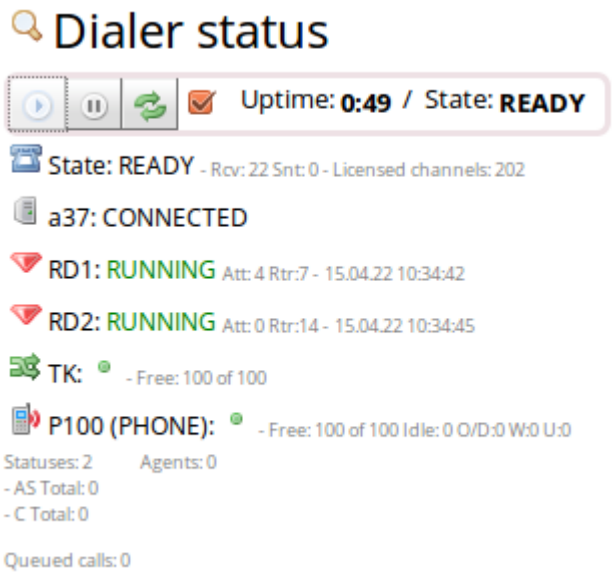
When you restart the dialer, all state is synchronized to disk and all entities are reloaded.

If you want to have a deeper view about what the dialer is doing, especially when it's not dialing out, you should check the Dialer state page that has a deeper view of the decisions Wombat took for each specific campaign.

In general, the dialer is supposed to start automatically when the system starts - see Dialer startup for more information on the issue.

Observing Asterisk queues

When an EP of type Queue is used, its status should immediately reflect the state of the underlying queue.



Before using those EPs, it is advisable to run a limited test to make sure the configuration is in working order.

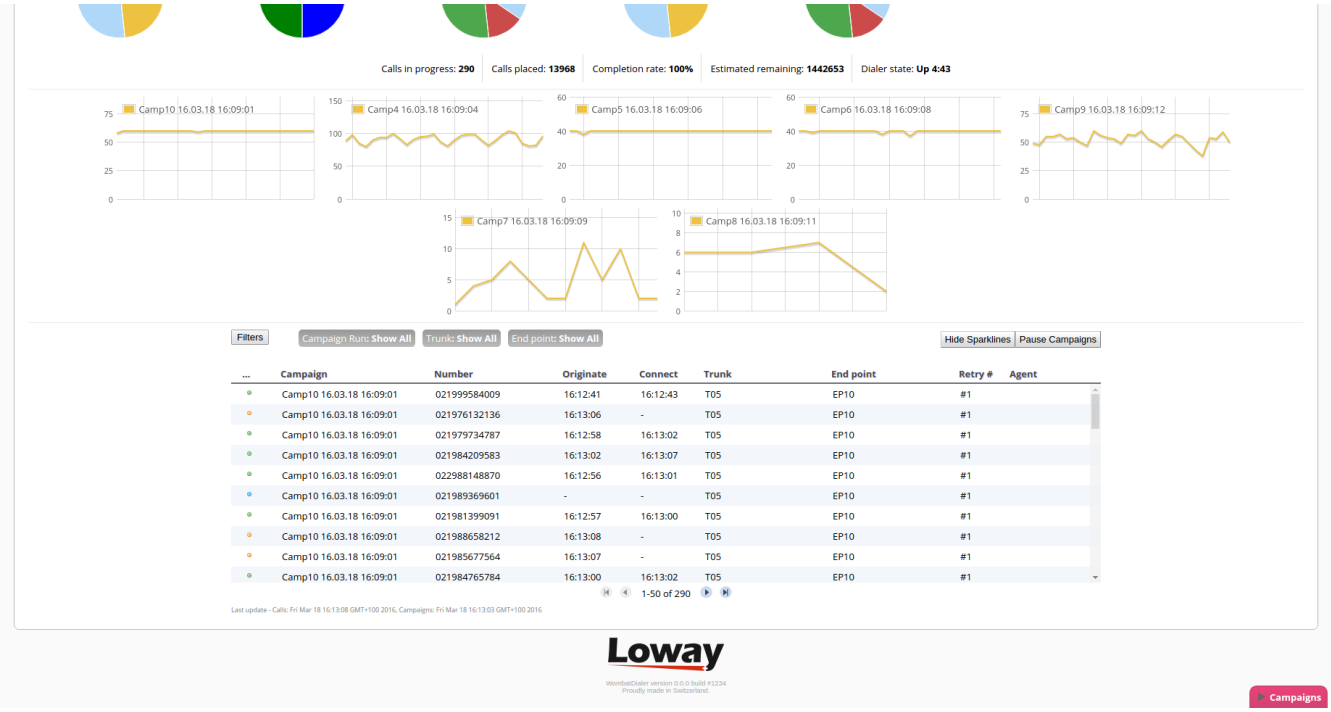
In order to run such a test:

- create a campaign with a Queue endpoint. You may create it IDLE and add no call lists, as we do not need the campaign to do anything special
- run that campaign
- reload the Dialer status in order to see if the queue is being observed (you have to click on the reload icon manually each time).

- If the queue is present, you should see it something saying "Free 4 of 7 W:2". This means that WD is seeing 7 agents connected of which 4 are free (where 4 is the result of multiplying the actual number of observed channels by its boost factor), and that there are 2 calls waiting on the queue.
- You should be able to see the current agents and waiting calls - if any - the queue is processing.
- try and log on, log off, pause and unpause an agent. You should see the number of free and available channels change accordingly. Try also sending calls to the queue and see if the number of free agents and of waiting calls is correct.
- try also placing calls from some agent extensions and see if the number of free channels reflects this correctly.
- if you plan to have agents working on multiple queues at once, run the tests above while the agents are logged on in at least two queues and make sure statues are updated correctly.

The Live page

The Live page lets you interactively control and monitor what the dialer is doing. You will be able to see at once all the Runs of Campaigns that you can access.



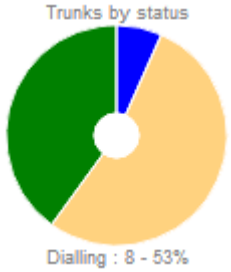
The top part of the page displays a set of "doughnut" graphs, showing:

- The active channels per Asterisk servers involved. Each Asterisk server is displayed in a different color and you can see which is which by hovering over it
- The status of calls on trunks. This shows how many calls are in each state.
- The usage of trunks by campaigns
- The relative number of calls per each trunk
- The usage of end-points by campaign
- An estimated number of remaining calls, divided by campaigns. This number is *estimated* as it is

impossible to know in advance how many recalls will be necessary to complete each campaign.

- The status of each running campaign

If you fly over each pie slice in the graphs, then a legend will be shown explaining what that slice means. Note that for live calls and campaign runs, the color codes are fixed to make them easily recognizable at a glance.



Just below the graphs, there is a row displaying (for all campaigns):

- The number of open calls
- The total number of calls placed on running campaigns (since they were started)
- The call termination rate, that is the percentage of calls attempted for which a reschedule was not necessary
- An estimated backlog of calls that have to be placed for running campaigns. This only counts calls in the dialing buffer

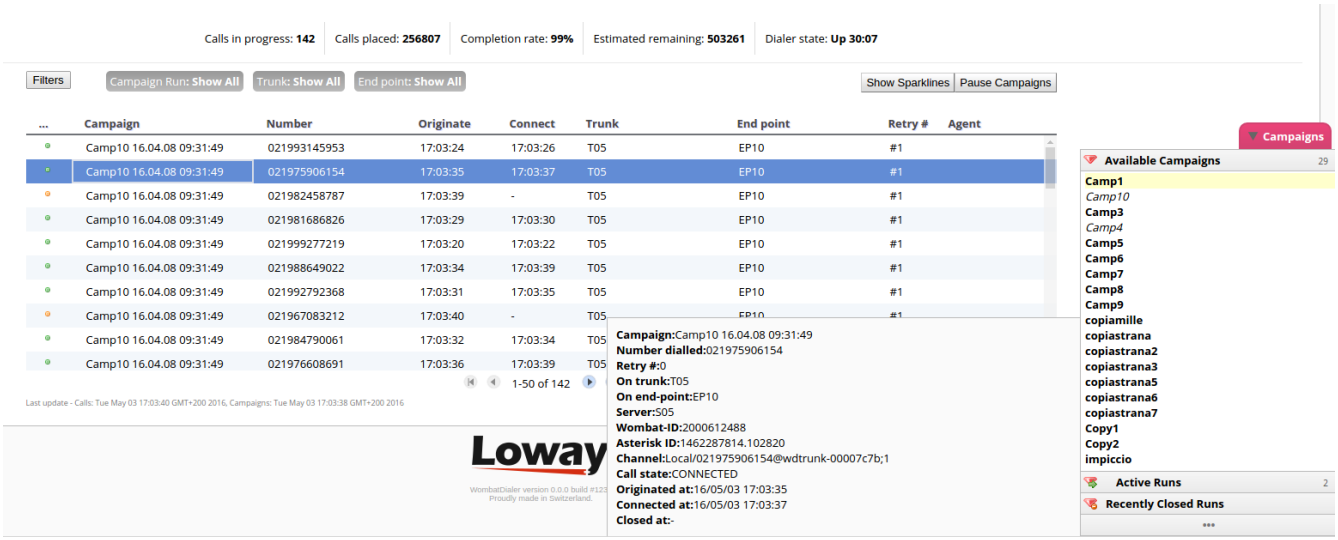
Below is a pageable list of live calls, that is refreshed every few seconds. For each call you can see:

- The number dialed
- When the call was originated
- When the call was answered
- The trunk used
- The end-point the call was connected to
- The number of times this call was attempted
- If present, the actual agent taking the call (depending on whether the call is dialed in reverse or direct mode, the agent may appear immediately or only when the call connects).

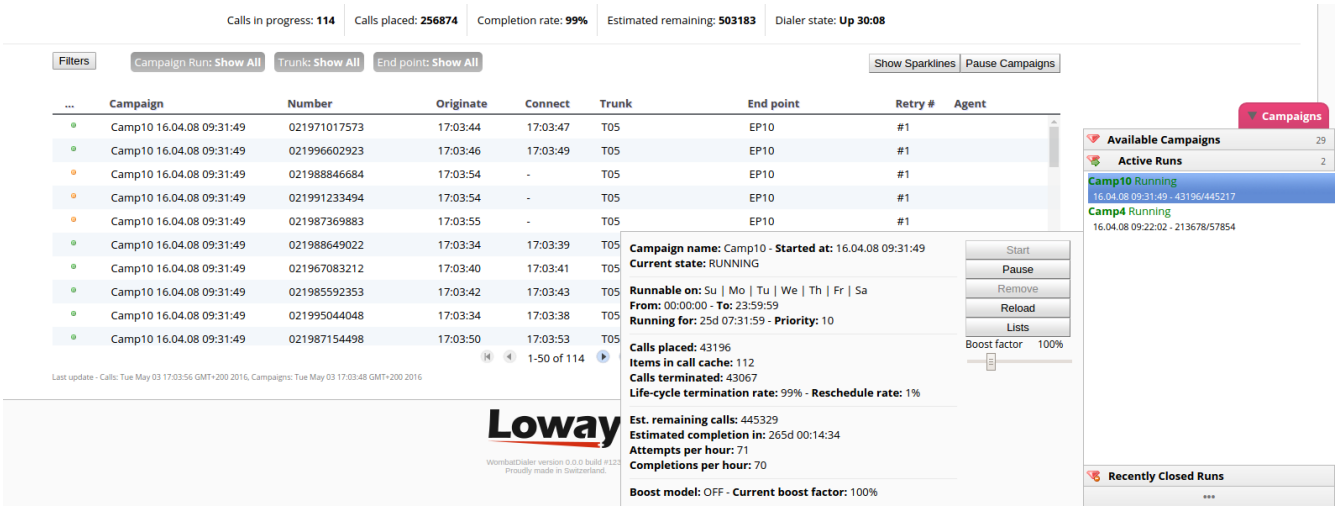
In this list only calls managed by WD are shown, so you can still use your PBX as usual and such calls will not be displayed.

By the bottom of the page there is a "Details" section, where you see the details for the last item you selected - be it a call, a campaign to be started, a running campaign or a closed campaign.

For example, here is what you would see by selecting a connected call:



And here is what you would see by selecting a running campaign:



Next to the "Details" section there are buttons with possible actions for the item being displayed - you may for example start and stop campaigns. When you perform an action, a pop-up will be displayed to confirm that the action was received.

Filters

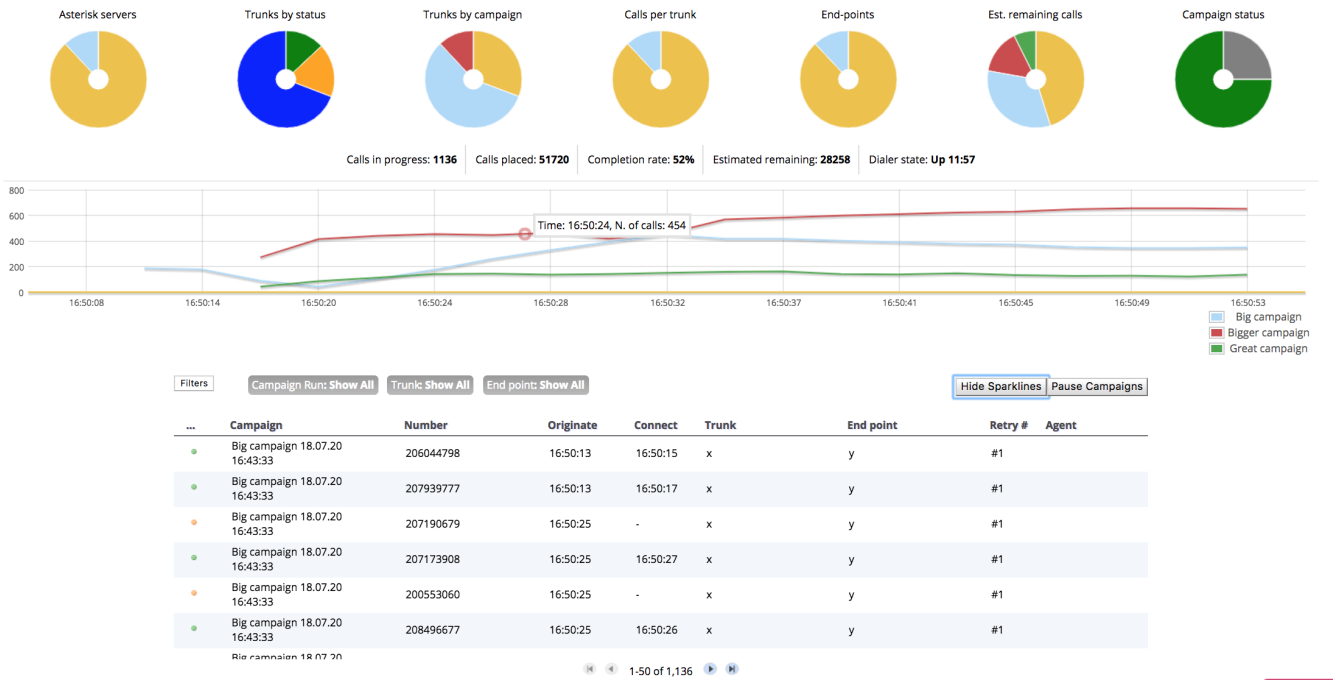
Right above the live Calls list we can see the "Filters" button that presents the user with the Filtering panel.

This feature focuses on giving the user the possibility to filter, in an useful way, the data displayed in the Live Calls Table. Essentially the Live page analyzes the data flowing through the table, and automatically calculates possible filters that might be of use. These filters include:

- **Campaign Runs**: Every time a call belonging to a new Campaign Run is analyzed, the filter system adds a new filter to the Campaign Run filter menu.
- **Trunks**: The same thing as above goes for new Trunks detected by the Filter System.
- **Endpoints**: The same goes also for new Endpoints.
- **Search**: A search bar component has been added, allowing the user to display only the lines containing the words typed in the search bar.

Sparklines

Next to the Filters Button you can see the Show Sparklines button. This button presents the user with the Sparkline System panel.



For every active Campaign Run detected (where active means that is currently holding at least one call), the Live Page generates a SparkLine Graph that shows the number of calls over time, concerning that particular campaign.

If you hover with the mouse pointer over one of the Sparklines, a pop-up label shows you the number of calls active at that particular moment. If the number of active calls belonging to a particular campaign reaches zero, the corresponding Sparkline is removed from the page.

Multiple pauses

Finally, next to the Show Sparklines button, you can see the Pause Campaigns button.

This button prompts a pop up that allows the user to select multiple campaigns to be paused. This is useful if you want to pause all running campaigns in one simple instruction.

The Live page does not work unless the dialer is running (state READY). So if you try starting a campaign but the dialer is not working, you will see the notification but nothing will happen.

Campaigns and runs

On the right-hand side of the screen there is a pull-up overlay box made up of three sections plus one, labeled "Campaigns". It is usually hidden and you have to open it manually in order to see it.

On the right-hand side of the screen there is a panel with:

- Available campaigns: the set of campaigns that can actually be run. As a campaign can be run

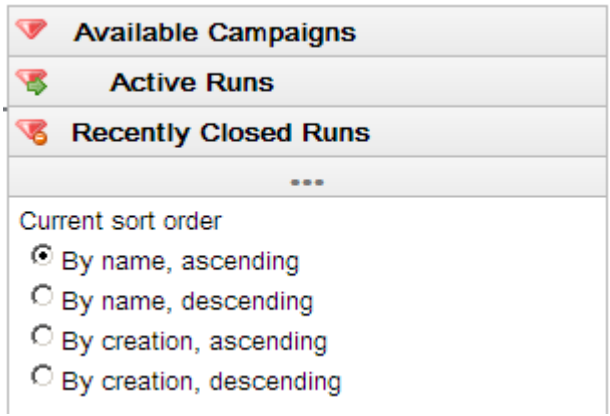
only once, campaigns that are running and therefore not startable are displayed in italics.

- Running campaigns: these campaigns are running now, or could be running if some condition was met - e.g. having some more calls appended for IDLE campaigns, or a different moment in time for WRONG_TIME campaigns
- Recently closed campaigns: here are displayed the details of each recent run

For each campaign run, a color code is used to display the state it is in. You will also see:

- when the run was started (as to distinguish it from other runs)
- the number of calls placed
- an estimate of the current backlog for this run

The section labeled "..." lets you modify the order under which campaigns and runs appear.



The selection you make will be applied to all campaigns and runs while the page is open.

When you select a startable, running or recently closed run you will see a panel showing its details plus available actions. You can start, pause and remove runs, plus you can control the dialer's "boost factor".

Controlling the boost factor

You can manually "tune" the boosting currently applied to a campaign by dynamically setting an additional factor that will be used to decide how many channels to dial.

The way this works is:

- The number of free agents on a queue will be multiplied by the EP's boost factor
- The result will be again multiplied by the run's boost factor. Boost factors below 1 will reduce the number of available channels, while values over 1 will increase it.

So if you have a queue with 2 idle agents, with an EP's boost factor of 2, and your current boost factor on the Live page is set to 1.5, WD will try placing 2 * 2 * 1.5 = 6 calls. If you tune the boost factor on the Live page to 0.5, WD will only place 3 calls.

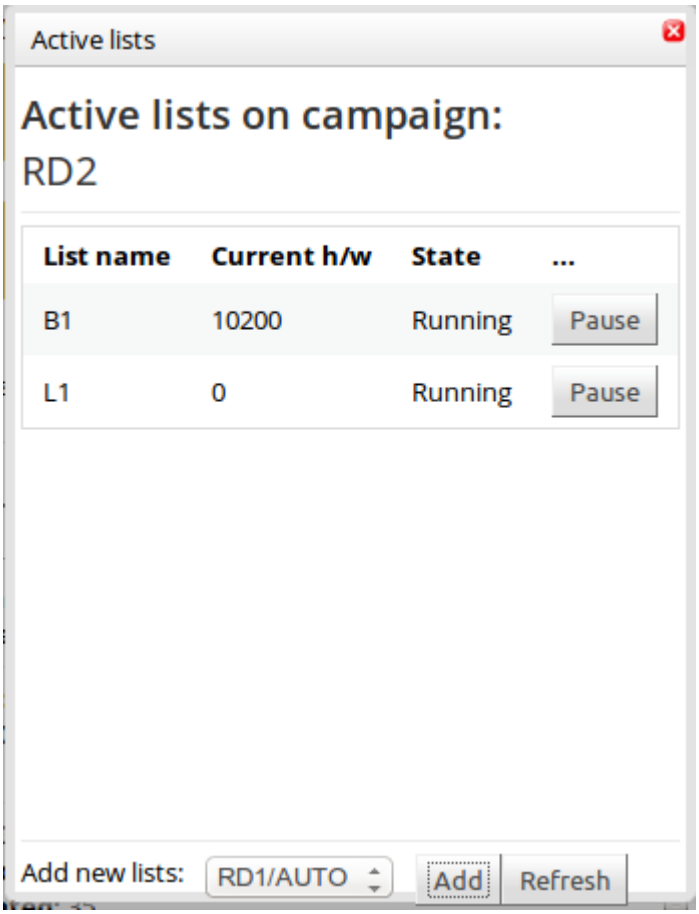
When a run starts, its boost factor is always set to 1. In general, you should change the boost factor from the Live page only in cases where the pattern of answered calls changes strongly during the

day. WD is build for unattended operation, so there should be no need to have a person constantly changing this value.

You will get best results by changing this value slowly and waiting for a while after each change to let it stabilize. By constantly moving between very high and very low factors, you will send Wombat into a self-oscillating mode where you will have a lot of idle agents and a lot of lost calls. In any case, Wombat converges automatically towards a stable state, so if bad things happen, just wait a couple of minutes.

Using dynamic lists

Right from the Live page, you can control the lists that are run on a campaign.



You can access this information by clicking on the "Lists" button next to a campaign run details.

You can add new lists to a running campaign, pause and unpause lists on existing campaigns, and see how far lists were processed so far.

If you stop all lists on a run that does not IDLE on termination, the campaign run will simply terminate - make sure this is what you want to do.

Please note that if you change lists on a running campaign and reload it, the new lists (or any removed list) are NOT added or removed from a running campaign. This is an expected behavior as, by forcing a list reload, you could override the finer list control you have from the Live page itself. The next time your campaign runs, though, its initial set of lists will be taken from the current

configuration.

Remember that automatic lists behave differently - they won't appear in this panel, and if they do, it's most likely a mistake - see Automatic lists for details.

Using multiple numbers per call

WombatDialer allows you to set additional numbers for a call to be placed. The idea is that if a call has two additional numbers, the number chosen takes into consideration the current retry, so that:

- The first attempt is always placed on the main number
- The second attempt is on the first multi-number
- The third attempt is on the second multi-number
- The fourth attempt is on the main number again
- The fifth attempt is on the first multi-number
- ...and so on

In order to define a multi-number on a call, you need to provide attributes which names beginning with MULTINUM - eg MULTINUM1, MULTINUM2 and so on. If a MULTINUM is empty or not present, it is skipped - so it is valid to upload a call with MULTINUM1=123, MULTINUM2 is empty and MULTINUM3=456.

When reporting on a call where a multi-number was used, it will appear as "123 / 456", where this means that the call is the one which main number is "123" but the actual number dialed was "456". Still, the main number will only appear as "123".

When applying a blacklist to a multinum, only the "main" number will be checked against the blacklist; if that is present, then all multinums are considered blacklisted at once.

Color codes used for live calls

Table 1. Live calls

State	Color
RD_RESERVED	purple
RD_REQUESTED	pink
REQUESTED	blue
DIALLING	orange
CONNECTED	green
TRANSFERRED	light green
TERMINATED	black
Any other	red

A complete description of a call's life-cycle can be found in A call's life cycle.

Color codes used for campaign runs

Table 2. Campaign runs

State	Color
COMPLETED	black
ERROR	red
IDLE	gold
PAUSED	gray
RUNNABLE	blue
RUNNING	green
WRONG_TIME	maroon
Any other	pink

A complete description of a call's life-cycle can be found in chapter A campaign run's life cycle.

The Copy Campaigns Panel

This feature allows you to create a new campaign, using an existing campaign as a template. The page prompts you to choose an existing campaign and loads its data. You can then modify some essential data, see the call lists connected to it, remove them , add some existing lists to it or, eventually, create new lists uploading them from a CSV file. Finally, the page allows you to simply create the new campaign or, if you choose to, to create it and run it directly.

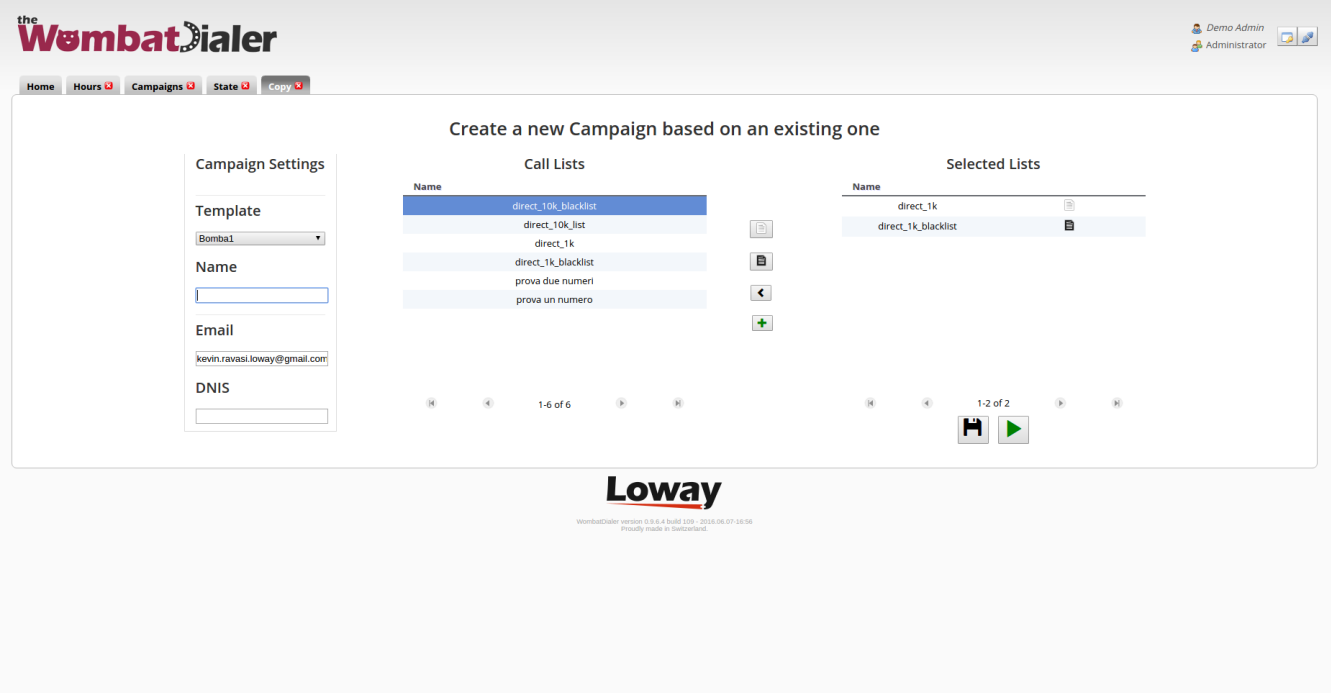
On the left side of the panel you can see a List Box named "Template", which contains all the campaigns available for copying. If you select a template, the Selected Lists Table will be filled by default with the same call lists and black lists originally tied to the campaign you chose as a template. Below the Template List you can see the Name text box. This is where you need to specify the name of the new campaign you are creating. The Email text Field is located right under the Name Field, this is where you set the email connected to the new campaign. The same thing goes for the campaign's DNIS, by typing in the DNIS Text Field located under the Email Field.

In the middle section of the panel, we can see a list containing all the available call lists that can be attached to the new campaign you are creating. To add or remove these call lists you can use the control buttons located between the Available Lists Table and the Selected Lists Table.These control buttons are listed below.

Add list: The green Ok Check button adds the selected list to the Selected Lists Table as a normal call list. Add blacklist: The red Cancel button adds the selected list to the Selected Lists Table as a blacklist. Remove List: The black Left Arrow button removes the selected list from the Selected Lists Table. Create List: The green Plus Button prompts a pop-up containing a Call List creation wizard that allows you to create a new call list by uploading a CSV file.

Once you have inserted all the data you can save the new campaign by clicking on the Floppy Disk Button in the bottom right of the panel, or you can save and run the campaign directly by clicking on the Play Button next to the save button. By doing this Wombat Dialer will open the Live page

Automatically.

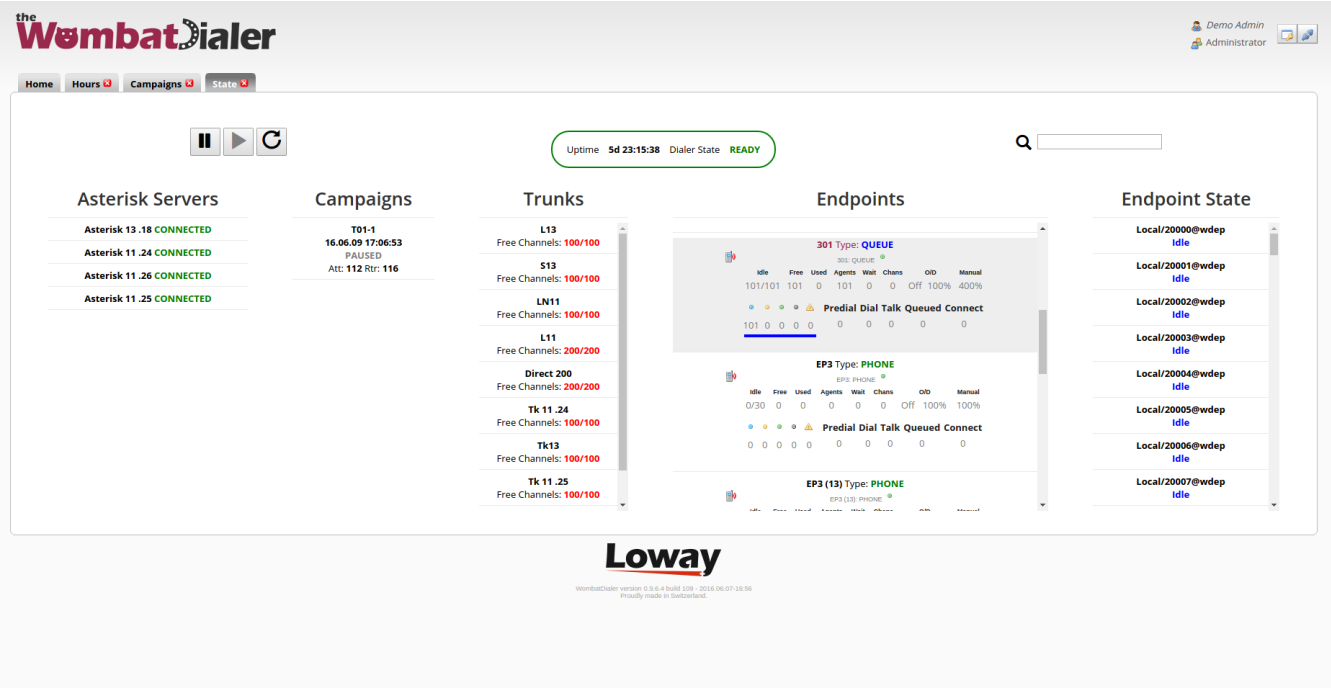


The Dialer State Page

The new Dialer State page provides the user with more detailed information about the dialer status, also showing synthetic information about Campaigns, Trunks, Endpoints, Agents and Waiting Calls.

The Wombat Controller Box on the Home Page has been simplified, and a substantial part of the information has migrated to the new Dialer State Page.

The new page shows different columns containing significant Data. The Endpoints column in particular, can be clicked upon, selecting each endpoint. If the endpoint type is QUEUE, information regarding the Agents logged on the queue and the calls currently waiting to be answered.

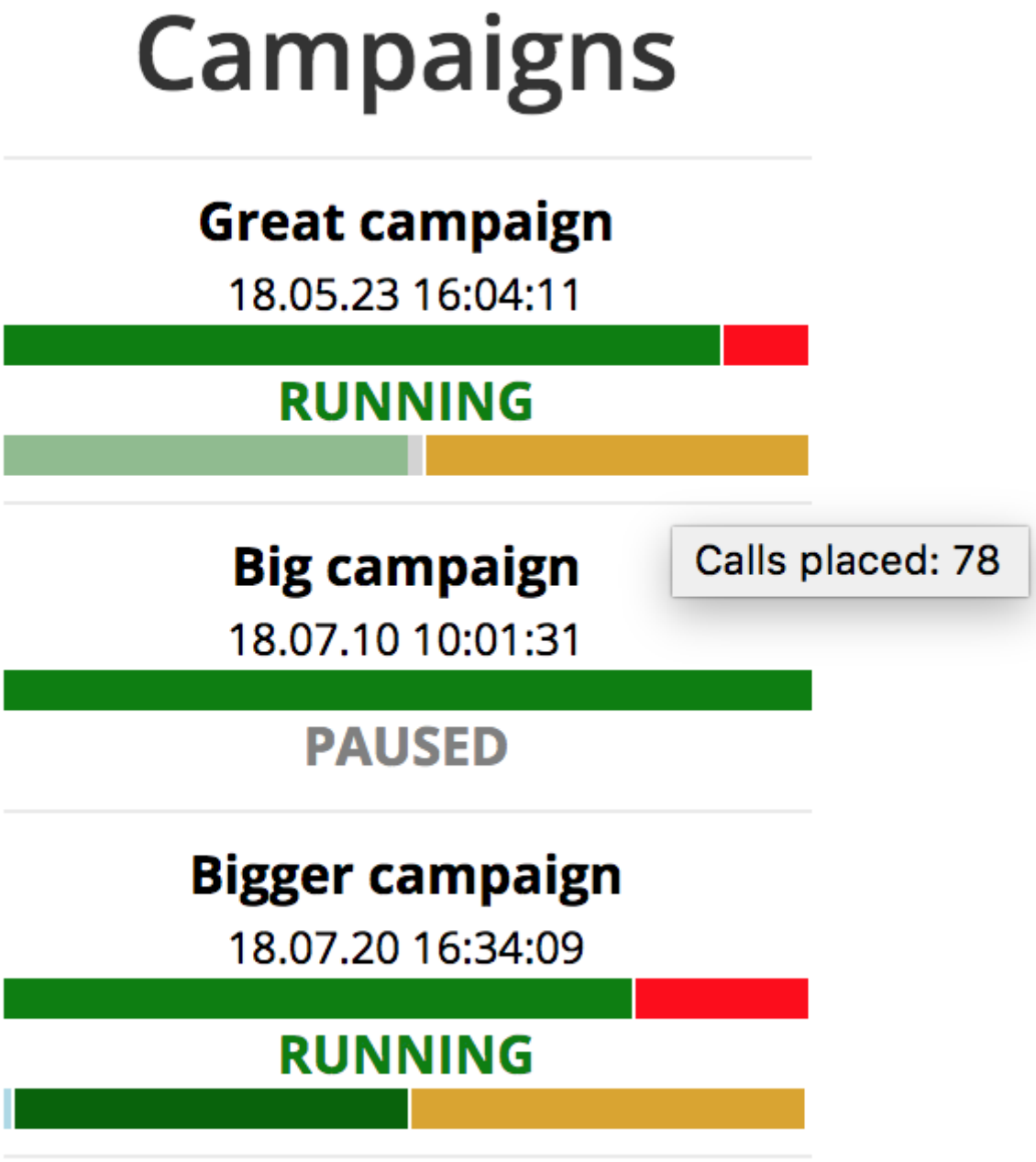


In order to start seing any data on this page, you need to have a running campaign. If there is no need to actively use them, Wombat won't try and monitor servers, queues or extensions.

Understanding WombatDialer decisions

If you want to understand what WombatDialer is actually doing, under each running campaign in the Dialer State page there is an information bar that displays the latest decisions that Wombat took related to that campaign.

Wombat runs repeatedly for each active campaign run, and based on the state of current calls, trunks and end-points tries to decide whether more calls should be made. The color bar under a campaign displays those decisions and lets you understand what Wombat is doing and why it is dialing or not.



To read the graph, you need to find the most common states and read them according to the list below:

- **NO ENDPOINTS FOR RUN**: This run does not appear to have any valid end-point configured.
- **NO TRUNKS FOR RUN**: This run does not appear to have any valid trunk configured.
- **NO CHANNELS ON TRUNK**: There are zero free channels on trunks.
- **NO EP FOUND WITH FREE CHANNELS**: No EP has free channels - they are all used. On an EP of type queue, it might mean that WD is not receiving events from the queue, so it has no way of knowing when to dial out.
- **ALL LICENSED CHANNELS USED**: This run cannot temporarily run as there are no free licensed channels - usually because some other campaign is using them all.

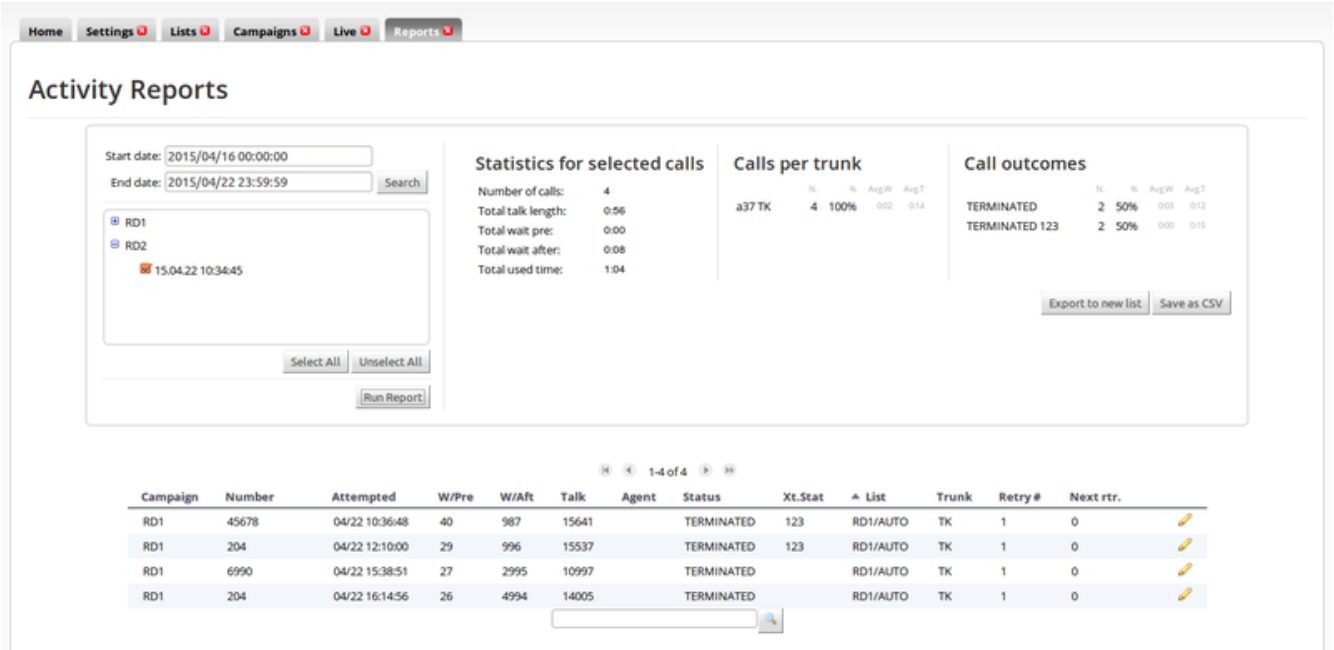
- **NO CALLS FETCHED**: WD is trying to fetch new calls from lists or recalls, but it's finding nothing. Usually appears before campaign termination.
- **LIMITED BY EP SIZE**: WD would be able to place more calls, but it has to limit itself to the number of available EP channels.
- **LIMITED BY LICENSED CHANS**: WD would be able to place more calls if you had a larger license key.
- **LIMITED BY TRUNK RATE**: WD would be able to place more calls, but you are asking it not to overcome a fixed CPS threshold (see Trunk CPS limits).
- **LIMITED BY ACTUAL FETCH**: WD would be able to dial more calls, but it did not receive enough calls from the database. If this happens often, increase the batch size settings on your campaign.
- **CALLS NOT NEEDED**: No need to place new calls.
- **CALLS PLACED**: Some calls were placed.

Sometimes an error may be displayed as a static number (e.g., 49) representing a 30-second time interval; the number remains stable because new errors replace previous ones dating back 30 seconds and will remain at this peak until the problem is resolved.

It is normal that not all runs end with some calls placed. Ideally, most runs should end with CALLS PLACED or NO CALLS NEEDED.

Running campaign reports

You can use WombatDialer to get hands-on details on the activities it performed.



Reports are performed **per run**, so the first thing you have to do is to search a set of runs by selecting a suitable time period and selecting runs of campaigns that were started within that time

period. Each run is identified by the date and time it was started.

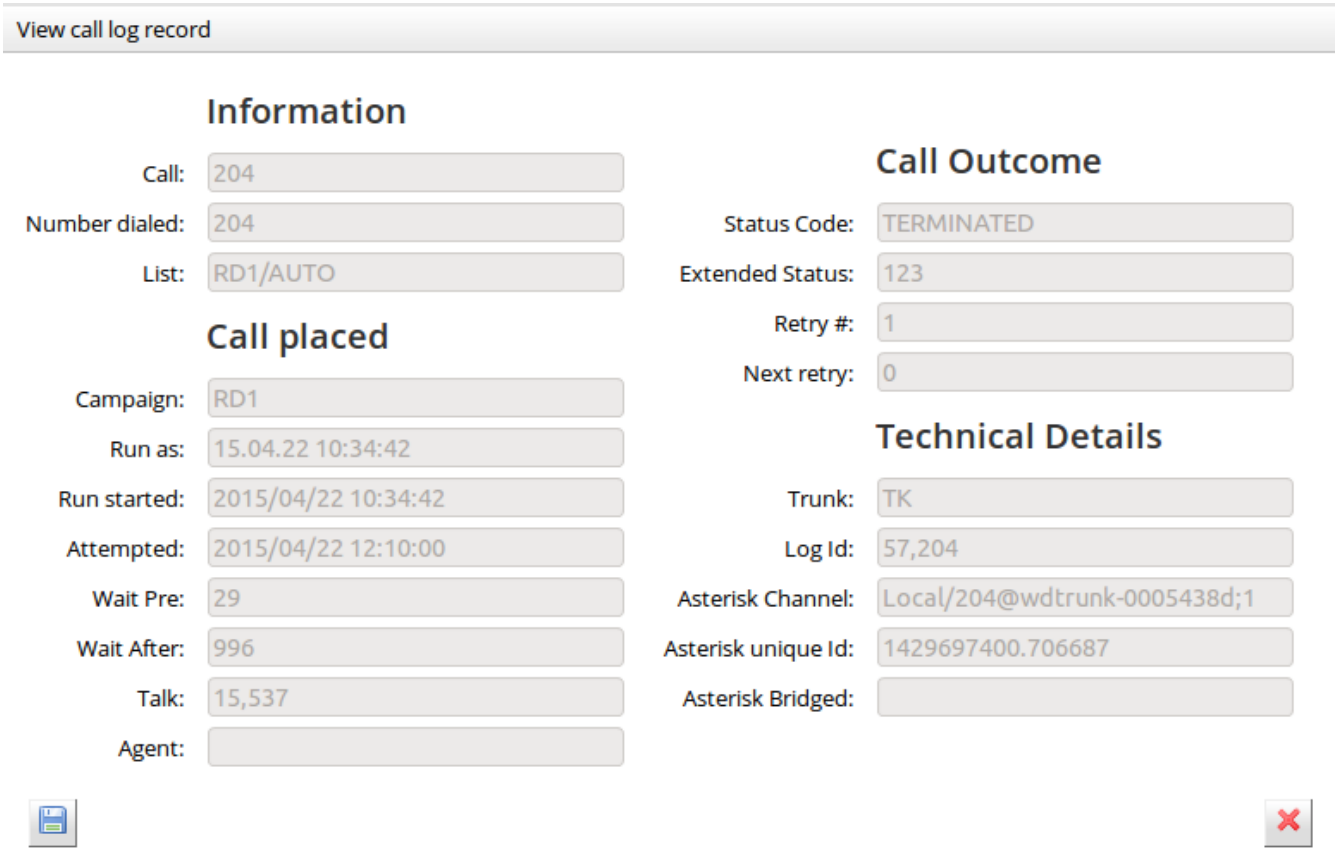
If you see no data for a campaign that you know has placed calls, you need to enlarge the time period for when it was started.

When you click on ">>>", statistics are computed and displayed in the right-hand side of the screen.

You can see:

- The total number of calls placed
- The total talk duration (in seconds)
- The total "wait pre" and "wait after" times (in seconds). "Wait pre" is linked to PBX latency, while "Wait After" is actual time waiting for a call to be picked up. See Call logs.
- The total conversation time (in seconds)
- The number of calls that were placed on each trunk - trunk names are prefixed with the server they belong to
- The number of calls that got each specific outcome

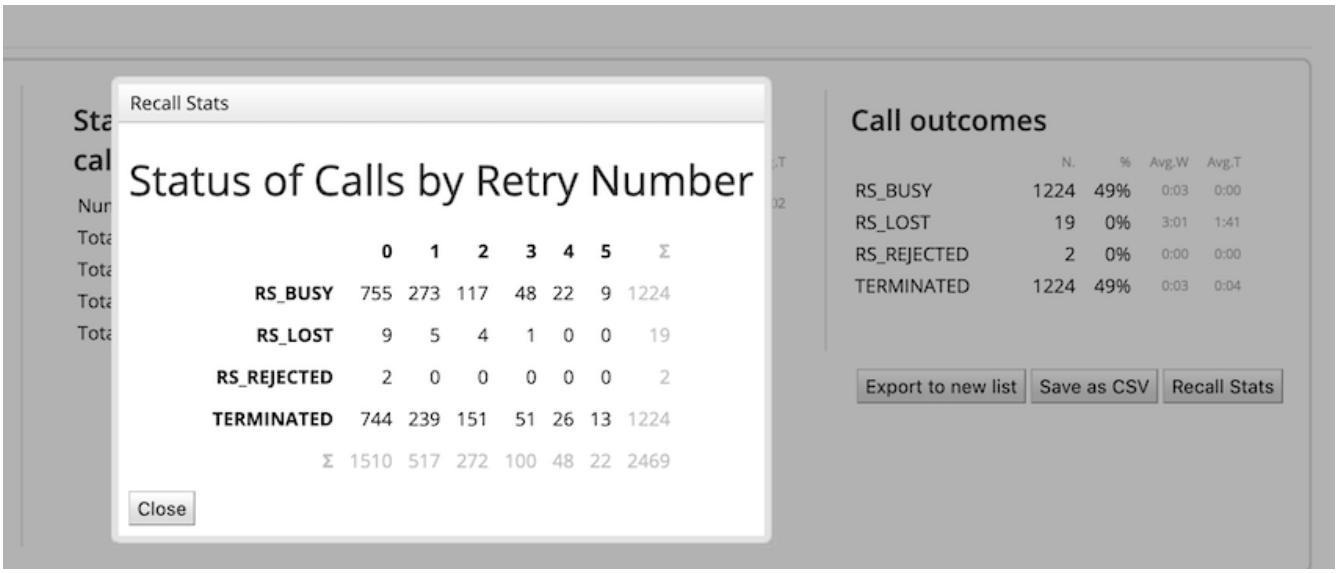
By the bottom of the page you can then see a paged list of calls that belong to the set you selected. You can use the search tool to zoom in on some specific number, and by clicking on the Pencil icon you can get a complete display of the call logs.



Tracking outcomes at the recall level

If you want, you can drill down on the outcomes that the dialer got for each attempt - so how many

calls were successful on the first attempt, how many on the second, etc. - you may want to click on the "Recall Stats" button. This show you the same information that is shown under "Cal Outcomes" but with an extra level of detail.



For each set of runs, you will see:

- a column for each "recall level" present in the data set
- the list of outcomes, one for each row, and how many calls belong to each
- in the bottom line, how many calls were placed to serve each recall
- in the last column, how many calls in total ended in specific outcome

This is quite handy to understand at a glance if your recall policies are "good enough" or you are losing out possible contacts because you are not retrying enough times.

Recall number 0 is of course the first attempt at dialling; technically only recalls numbered from 1 upwards are "recalls".

Exporting campaign lists

Looking at the list of dialed calls interactively is useful to understand what went on, but sometimes you want to save the details of all calls into a spreadsheet for later review.

This can easily be obtained by clicking on the "Export to CSV" button - you will get a spreadsheet of all the calls belonging to your selected runs.

Most spreadsheet packages cannot manipulate more than 50,000 records in a single file. Though WombatDialer will not enforce this, make sure you avoid exporting data files that are too large to be usable.

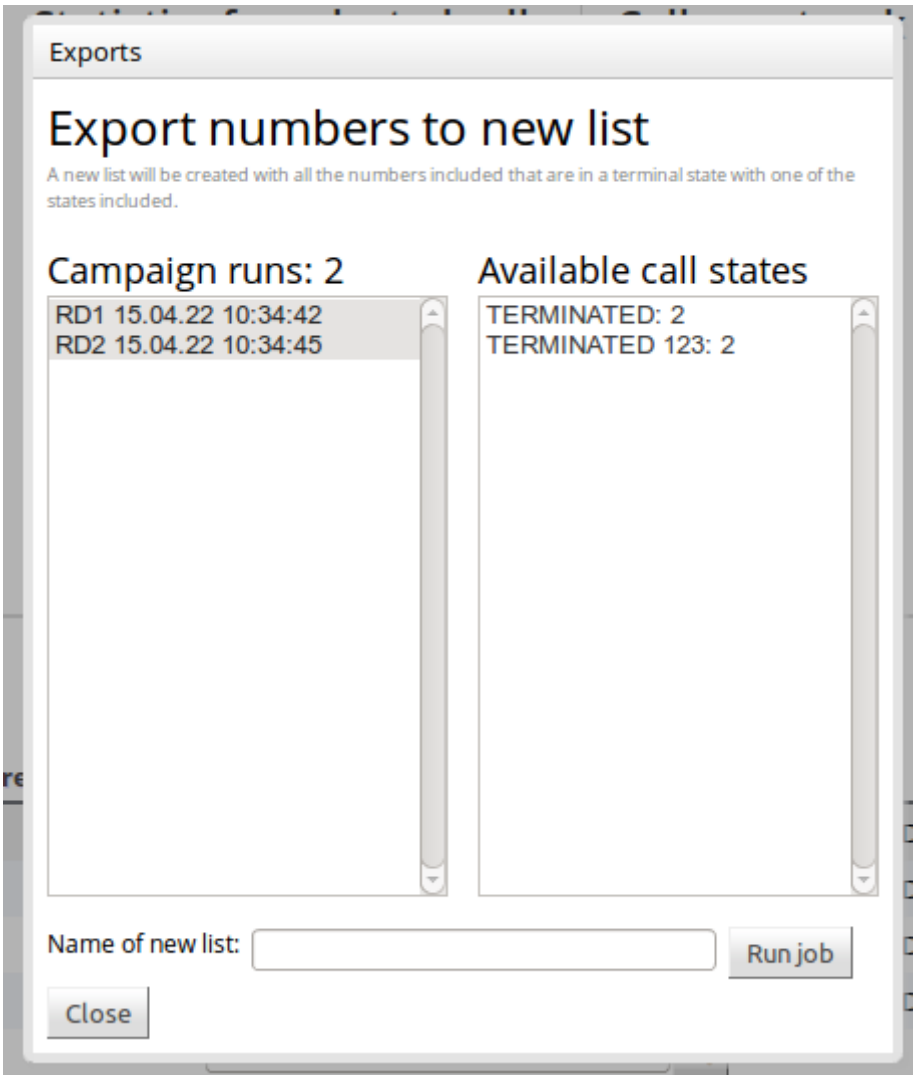
Building new call lists out of run results

When you run a campaign, you usually set reschedule rules so that call can be retried in case of errors.

Still, it is often handy to operate on a different time-scale so that you can:

- run a campaign to completion
- get the set of calls that would not complete
- reschedule those calls as a separate campaign at a later time

This can be done using the "Export to new list" button on the Reports page.



When you click on that icon, calls are counted by their final status, that is the status of the last retry. For example, if a call was tried at 10AM and got BUSY, and was retried at 11AM and completed successfully, its final status will be TERMINATED.

As terminal states are not available before this transaction runs, the states you see in the list and the number of calls for each state are about all possible states of all possible calls within the selected runs. So it is quite common to find that the numbers of calls actually created is way lower than the numbers shown, or that a specific state has actually no terminal calls.

We first select a set of runs from the ones we selected in the Reports page, and select a set of termination codes. Then you enter a name for the new list to be created and click on "Run job".

A new list will be created under the new name and containing the records for the calls you just selected. In order to see it, you must go to the Lists page (it might be necessary to close the Lists tab

and reopen it).

You can also use this feature to create lists of numbers that were answered correctly in order to "prune" existing lists of old and invalid entries.

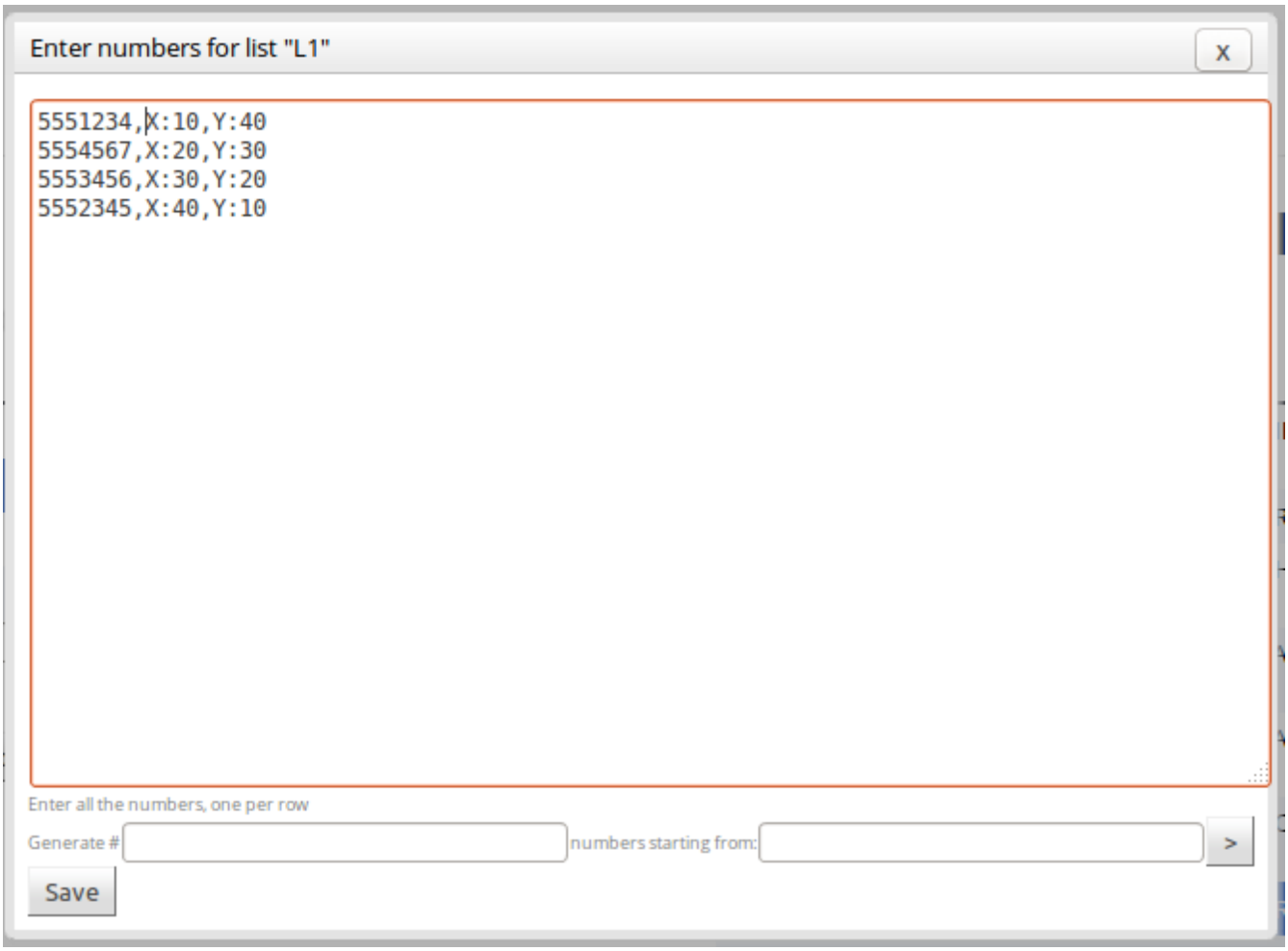
By creating a call list numbers you know to be invalid, you can add it to your campaigns as a black list so that those numbers will be immediately discarded.

Importing and exporting call lists

From the Lists page, you can import and export data to and from call lists. WombatDialer lets you select either native or CSV format.

Importing call lists - Native mode

By clicking on "Upload list of numbers" when a list is selected, you can copy and paste a set of numbers to be added to an existing list. As numbers might have associated call attributes, there is no control over possible duplicate numbers as you may want to dial the same number multiple times.



Numbers should be uploaded as a set of rows, each starting with the number to be dialed. If you want to add attributes to be passed to the PBX, you should have them in the format "ATTR:VALUE" separated by a comma.

For example, the line:

```
5551234,A:1,B:HELLO
```

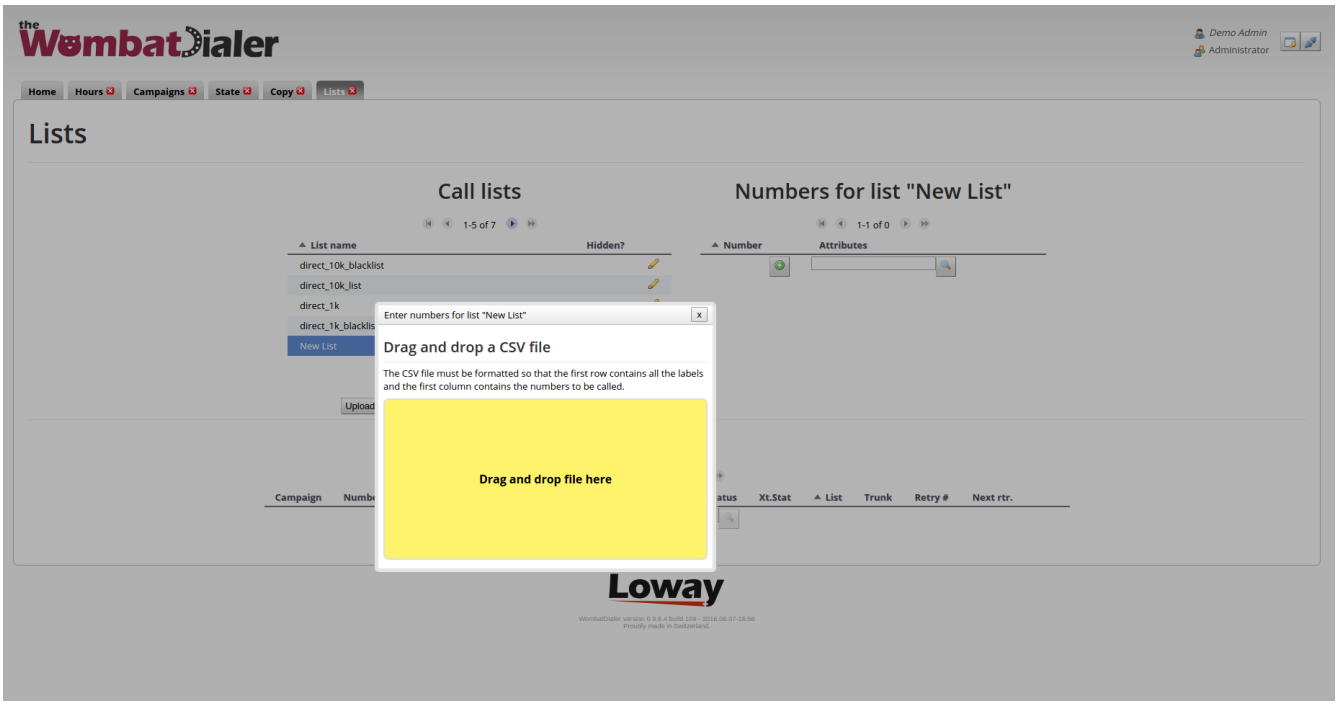
Loads the number "5551234" with attribute A set to "1" and attribute B set to "HELLO".

You can also use a wizard to generate a set of progressive numbers, e.g. generate 1000 numbers from 5550000 onwards. The generator is smart enough to handle the generation of numbers starting with one or more zeros.

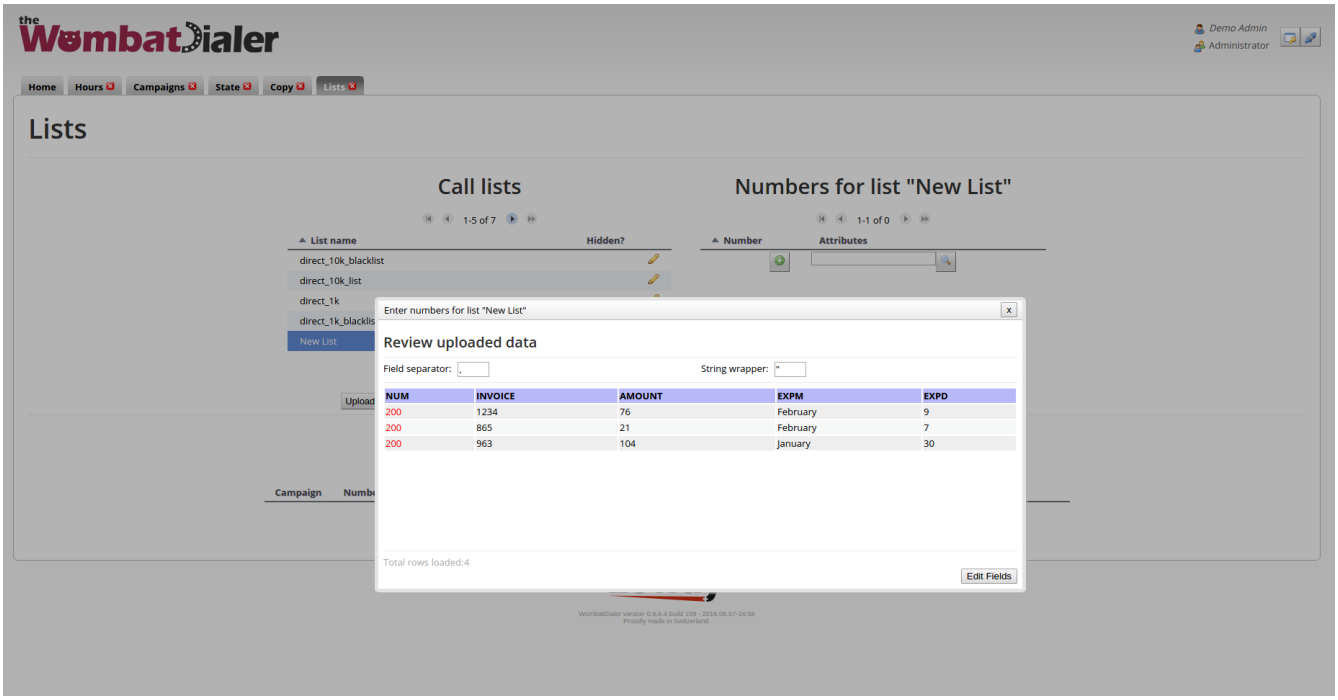
the maximum set uploadable in one batch is about 2Mb, or about 100,000 numbers. It would be advisable to use smaller sets as to avoid causing CPU spikes.

Importing call lists - CSV mode

By clicking on "Upload CSV", you will be able to drag and drop a CSV file containing numbers and attributes to be added to the selected list. You can easily create CSV files from your favorite spreadsheet.

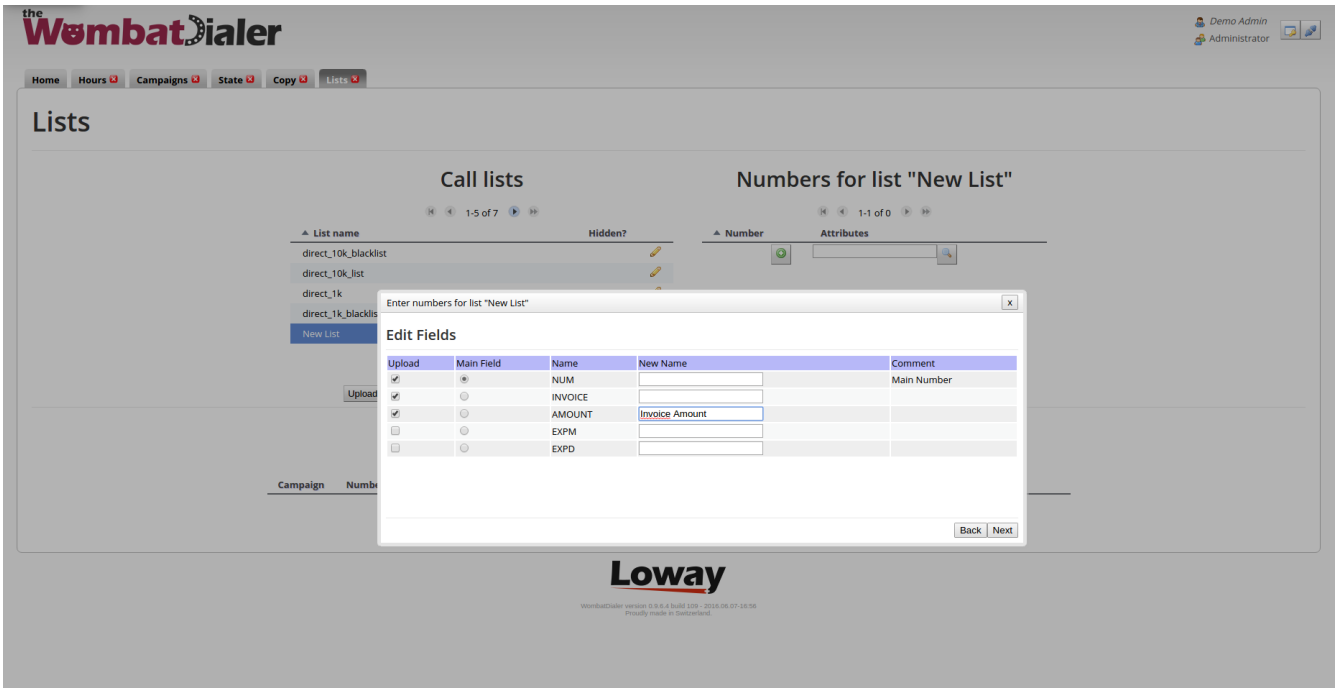


You simply drag and drop the CSV file to the yellow box on screen. When done, the file will be read and previewed.



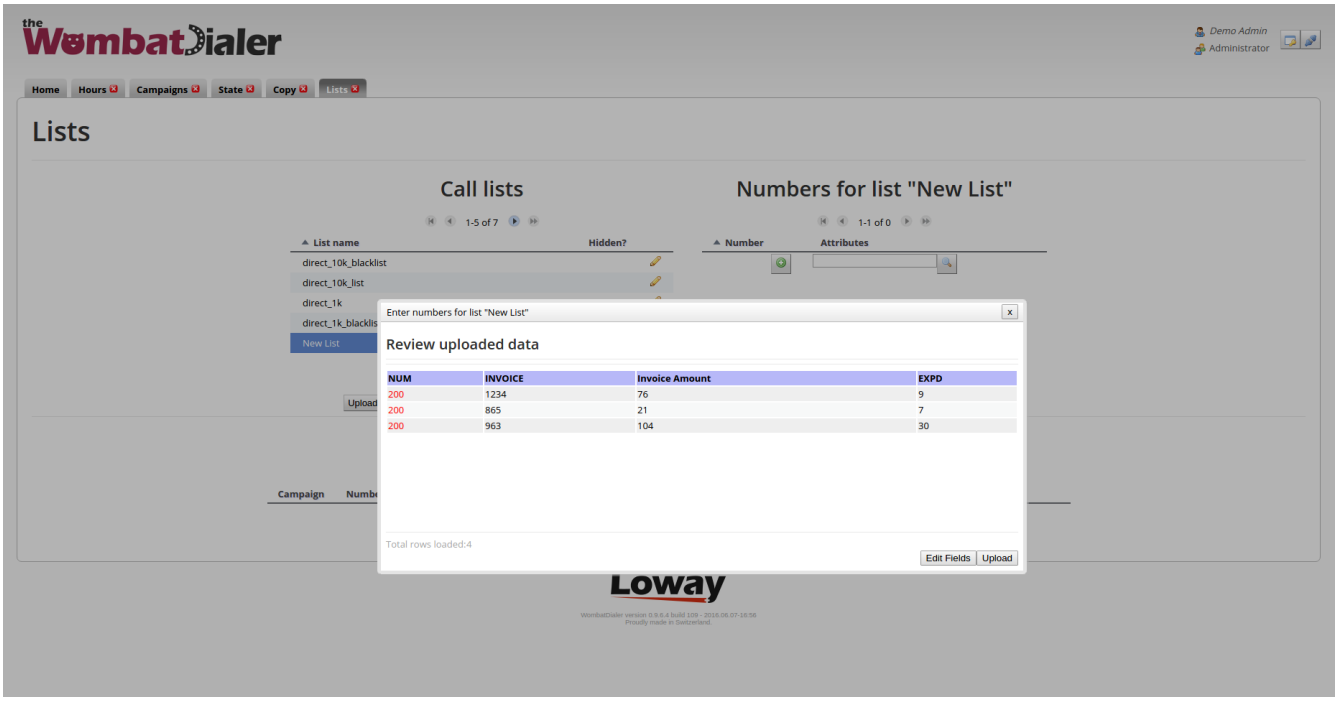
As your CSV file might have different field delimiters, once you have uploaded the file you will be able to change the delimiters until the file looks right. The numbers to be called will be shown in red.

After reviewing you file, you can edit any field in the next step of the process. By clicking on the "Edit Fields" button, you will jump to the fields editor page, through which you can modify the structure of the CSV data by selecting the field that will represent the “Main Number” field (that is the number that will be called when campaigns implementing this list will run). You can also rename eventual fields or decide not to upload some of them in particular.



In order for this to work, your browser must support the HTML5 File API. This may not work on older browsers.

Once you apply the desired change, by clicking on the "next" button, you can go to the final preview page. Here you will see the results of your CSV customization (if any at all). From here you can go back to the "Edit Fields" page, or you can upload the file to WombatDialer.



Exporting call lists

You can export a call list in textual format. By pressing on the "Export calls" button while a list is selected, you get a new text page with data in the format:

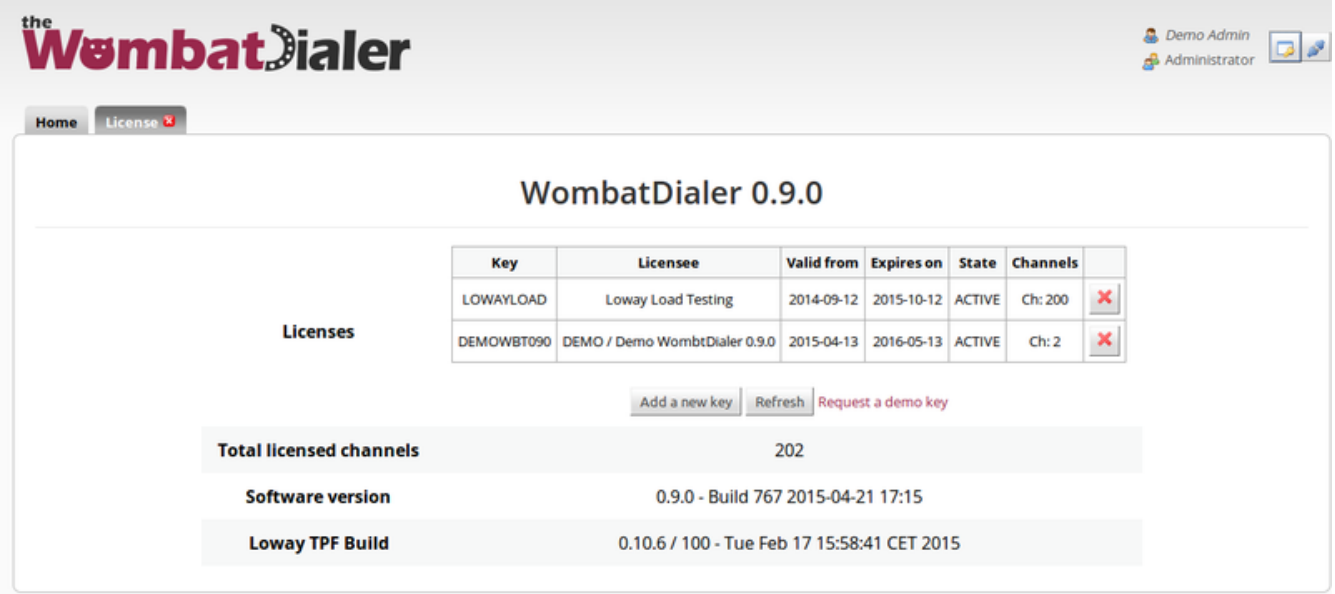
```
5551234,A:1,B:2
5551235,Y:20,X:10
5551236,Y:20,X:20
5551237,Y:30,X:10
```

All current attributes (inbound and outbound) are reported.

If you click on "Export CSV" the export file will be downloaded as a CSV file.

The License page

You can access the License page by clicking on the "key" icon on the top-right of the WombatDialer instance.



It displays:

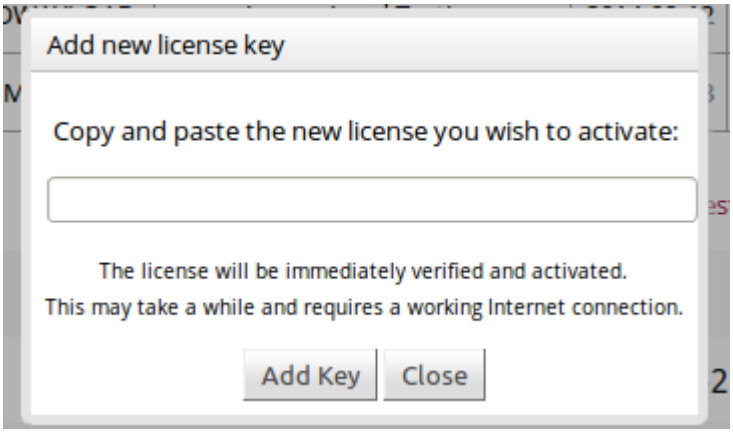
- The set of license keys installed on your WombatDialer system. For each license you can see:
  - The name of the license
  - The licensee
  - From when to when this license is valid. Note that you can install immediately a license that will only be valid in the future
  - The status of the license
  - The number of licensed channels
- The total number of licensed channels across all licenses
- The current version of the dialer, when it was built and the versions of the embedded TPF frameworks.

Installing a new license key

If you get a new license key from Loway, be it a full license or a demo key, it will look like the following item:

```
YOURCOMPANYNAME.12345678-23456789
```

You can install it by clicking on the "Add license key" button and entering the new code. You will need a working Internet connection from the server, as the new license will be downloaded and activated. This may take a few seconds to complete.



You may also force a check of all licenses by restarting the WombatDialer webapp.

Please note that:

- commercial licenses are additive, this means you can install two 10-channel licenses together in order to get a 20-channel license (though they will disable previous demo keys)
- demo license are exclusive, this means that they will disable other demo keys as only one can be active at a given time.

Any disabled keys will appear in state **REVOKED** (or possibly **DISABLED** if there is a license violation issue). Licenses that are past their expiry date will appear as **EXPIRED**, while licenses that are not yet active - for example, if you renew a license a month in advance - will appear as **FUTURE**, and will automatically become **ACTIVE** when their time comes.

If a license appears in state **W/P** (Waiting for license) or **ACTIVE_P** (Active pending verification), it means that WombatDialer cannot talk to the licensing server to download, activate or verify the license key. In this case, if the license was downloaded in the past, the license can be still be used but only for a a limited period until it is verified, and thence it will display an incorrect expiry date. WombatDialer tries verifying it up to that point in time, but if that's not possible, it is shut down.

WombatDialer requires outbound HTTPS internet connectivity to our licensing servers and cannot run without. See Outbound proxy to know more.

QueueMetrics integration

QueueMetrics is a powerful call-center reporting suite that can be used together with WD for extensive and accurate reporting of the whole call-center activity (inbound and outbound).

QueueMetrics offers:

- strong analytic capabilities, with over 150 elements computed
- live monitoring of agent status
- quality review and assessment of calls
- access to call recordings
- a powerful agent panel
- ...and many more features.

QueueMetrics can be found at https://www.queuemetrics.com

WombatDialer and QueueMetrics do different things, but together can be a very powerful call-center solution.

- WombatDialer is able to use a Queue as and end-point in order to connect calls to a set of agents
- QueueMetrics is able to monitor extensively the queue and provides a convenient agent interface that works well with WombatDialer

Installation

Both QueueMetrics and WD can be installed on the same server, that may or may not be the same server running the PBX. If installed automatically through *yum*, they will share the same Tomcat instance.

As QueueMetrics is a CPU- and memory-intensive application (as you could run reports on millions of calls at once) it might be better to put each application on its own virtual or physical machine in order to avoid slowing down WD - for which timely responses to what is happening on the PBX are paramount.

Understanding QueueMetrics integration

In order to turn on QueueMetrics integration, you simply need to enable the "QM_COMPATIBLE" logging option for WD campaigns. This will tell WD to produce a log on each Asterisk system that closely resembles the one created by inbound calls on queues.

This log will be created with the following peculiarities:

- the name of the queue is the name of the campaign - that is the name that needs to be configured in QueueMetrics
- the "agent" name answering the call corresponds to the end-point being used to place the call

- the extended call status reported to WD will be logged
- attributes will be logged (see below)
- all calls have the TAG set as their list

The most common case of QueueMetrics integration happens when you use an end-point of type QUEUE to have outbound calls routed to agents through a queue. In this case, you will have two different queue entities producing data for QueueMetrics:

- The queue that matches the campaign: here you will see all calls that WD attempted - the successful as well as the unsuccessful ones. This is basically the activity that WD performed and the actual telephone usage durations. In a real-life scenario, the vast majority of calls will be unanswered.
- If you run a report for the physical queue used, you will see information about calls that were successfully connected and were actually queued. You would expect to have a very low unanswered rate here - if it is high, it means something is not working as expected. You will of course have some lost calls, e.g. because the called person hung up before the agent was connected.

The difference between the number of successful calls in the campaign versus the total number of calls in the physical queue is due to calls that were hung-up before reaching the queue, e.g. during an initial IVR phase.

When you use a queue for inbound, it is often better to avoid playing music-on-hold and to offer ringing instead.

Logging of attributes

Attributes can be logged to QueueMetrics as IVRs, FEATURES and and VARIABLES.

- Features are ideal to track specific yes/no actions in the call
- IVRs are perfect to track IVR sequences, and will be extensively reported by QueueMetrics
- VARs are perfect for unstructured data you want to see associated to a call.

As you generally do not want all available attributes to be sent to QueueMetrics, you have to cherry-pick the ones you want logged by entering them in the **Attributes to be logged as QM variables** field in the Campaign definition.

In order to determine how an attribute is to be logged, you will prepend the name of the attribute with a sigil specifying the way it is to be logged.

- No sigil: the attribute is a VAR
- Dollar ($) sigil: the attribute is a feature
- Bang (!) sigil: the attribute is an IVR

So in the following set of attributes:

```
NAME $MYLIST OPTION !MYIVR
```

The attributes NAME and OPTION will be sent as vars, MYLIST will be a feature and MYIVR will be an IVR.

All call attributes can be logged; so everything that comes from the current list, plus campaign attributes, plus incoming attributes read from Asterisk. Attributes are logged at the beginning of the call and when the call is disposed (if changed). The call tag is automatically logged and matches the list name.

Logging of attributes on direct queue end-points

If WD is working with a queue in direct mode (that is, when calls are sent to agents by the queue and not by WD itself), and QM integration is turned on:

- As soon as the "synthetic" call is started, or when attributes are set, then they will be logged on the synthetic call. This works for reporting, but is not useful if you have agents using QueueMetrics who are working on the queue.
- As soon as an agent is connected (but no sooner), all previous attributes are logged on the queue as well. So in QueueMetrics, they will be available for screen pops or in general on the agent page. This happens automatically.

Any attributes set after an agent is connected will be logged to both calls.

Real-time monitoring

You will be mostly interested in monitoring only the physical queue - here actual agent activity is displayed, and agent presence information (log-ins, log-offs and pauses) is immediately available.

As all calls have their TAG set as the originating call list, it is easy to see the relative efficiency of lists.

Reporting

You might be interested in running reports on both the campaign and the physical queue in order to gather information on actual PBX usage times versus agent activity.

Using the agent’s page

QueueMetrics offers an interesting Agent’s page, from where the agent can get a screen pop to an external application (e.g. a CRM) when they receive an inbound call.

If you use the telephone number as the main key that is used to connect to an external CRM, then this will work natively with the basic WD integration, as the caller-id us usually preserved.

You may want to have finer control on the CRM pop-up by tracing not only the number dialed, but also the campaign ID or some of the variables that WD sends along the call. This can be achieved by

rewriting the caller-id field in Asterisk and using a decoder script that will in turn launch the actual CRM application. An example can be found in the Cookbook.

Using the agent's page for preview dialing

WombatDialer includes an HTML preview panel that lets your agents preview, process or skip calls.

It can be reached at the URL *http://myserver:8080/wombat/agents/rd_pop.jsp* and accepts the following parameters:

- **agent**: the channel being used, as present on the queue. It must match exactly the Asterisk id.
- **url**: an URL that can be opened for this call. Note that special characters must be quoted, e.g. "?" must be written as %3F and "&" will appear as %26. Any variables will be quoted when written like "<NUMBER>"
- **inset**: if 0, the page will display a list of known variables for this call and a link to the URL. If 1, the URL will be opened within an IFRAME right in the page.
- **baseUrl**: if Wombat displays a timeout trying to display this page, then you must set this parameter to the default URL for Wombat as seen from Wombat itself, e.g. *http://127.0.0.1:8080/wombat*

For example, to have calls opened from the agents' page in QM, you would use one of the programmable buttons to link to WD, like in:

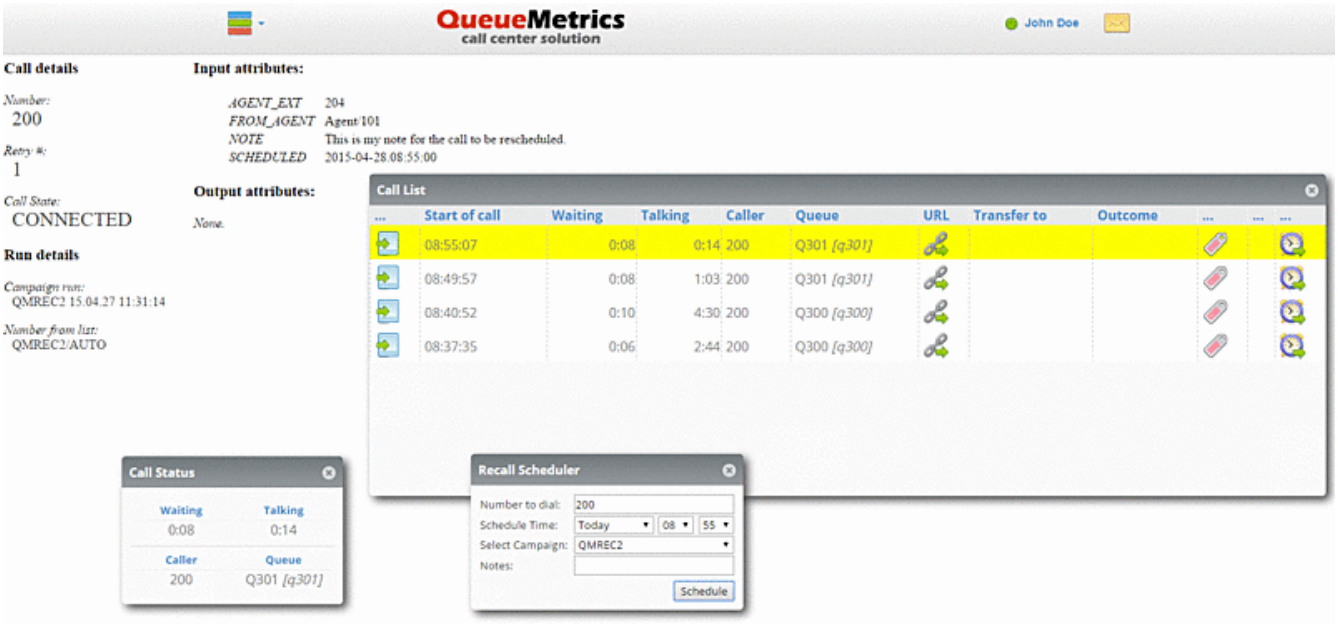
```
realtime.agent_button_1.enabled=true
realtime.agent_button_1.caption=Wombat
realtime.agent_button_1.url=http://10.10.5.30:8080/wombat/agents/rd_pop.jsp
                  ?agent=SIP/[a]
                  &url=http://10.10.5.31/sugarcrm/index.php%3faction=UnifiedSearch
                       %26module=Home<%26query_string=<NUMBER>%26name=<NAME>
                  &inset=1
```

The code above will link to WombatDialer on 10.10.5.30, will replace the agent code in the URL and will link to an embedded instance of SugarCRM running on server 10.10.5.31 passing along the number to be called.

Using the agent's page for regular dialing

WombatDialer includes an HTML panel to inquire/display the status of a live call. This integrates

nicely with QueueMetrics, as QueueMetrics allows for recalls to be scheduled right from the Agent's page.



It can be reached at the URL http://myserver:8080/wombat/agents/rd_pop.jsp and accepts the following parameters:

- **agent**: the Agent's identifier.
- **wid**: the WombatID. It can be written with a prefix (as it appears in QM logs) or without them.
- **unqid**: the Asterisk UniqueID for the call.

For example, you could use a Queue URL like the following:

```
http://myserver:8080/wombat/agents/live_call.jsp?agent=[A]&wid=[U]
```

To have the page opened in Icon. If you are taking the ID from an entry that is logged by Wombat you should use the "wid" code; if instead you are reading from a genuine app_queue entry you should use "unqid" instead.

A WombatDialer Cookbook

This section of the manual contains a few examples of WombatDialer deployments that show common usage patterns.

Table 3. Recipes

Title	Diff.	EPP	EPQ	API	NOT	ATT	INP	EVT	TTS	QM
Social Media Dialer	*	X		X		X				
Helping Wombats	* *		X				X	X		X
Outbound IVR	* *	X		X	X		X		X	X
Queue Metrics integration	* * *		X	X						X
Queue call-backs	* *		X						X	X
Understanding Queue EP	* *		X							
Automated Recalls	* *		X	X						X
Preview Dialing	*		X			X				X
Answering-machine detect	* *	X				X				
Understanding blacklists	*			X		X	X			

Diff

The level of difficulty - * Beginner - ** Intermediate - *** Advanced

EPP

Uses PHONE end-points

EPQ

Uses QUEUE end-points

API

Shows HTTP APIs

NOT

Shows HTTP notifications

ATT

Shows call attributes

INP

Importing and exporting call lists

EVT

Call events

TTS

Using Text-to-Speech engines

QM

QueueMetrics integration

A social media dialer

Let's imagine that we work for *ACME Social*, a company that specializes in tracking the success of its clients on social networks like Facebook or Google+. So, every time someone befriends one of their clients, we want WombatDialer to call the client telling them their current number of friends. The call would say something like *"Hello ! Customer 1234 has 127 friends. Goodbye!"*

This example shows a couple of features that are not trivial to implement on most dialers, notably:

- The message will be customized with a number of parameters, so that each client receives a personalized version and not just a pre-recorded note
- The dialer starts calling on-demand when something happens and handles reschedules internally

In order to implement this, we start by editing the Asterisk dialplan and create a couple of new contexts. The first one is called "telecast" and is used to generate the message being played:

```
[telecast]
exten => 100,1,Wait(1)
exten => 100,2,Answer
exten => 100,n,Playback(hello-world)
```

```
exten => 100,n,Playback(agent-loginok)
exten => 100,n,SayDigits(${user})
exten => 100,n,Playback(vm-youhave)
exten => 100,n,SayDigits(${friends})
exten => 100,n,Playback(vm-Friends)
exten => 100,n,Playback(vm-goodbye)
exten => 100,n,Hangup
```

As you can see, the context above introduces two channel variables `${user}` and `${friends}` that are placeholders for the user-id of our client and the number of friends they currently have.

As a convenience when testing, we want all numbers dialed during the test phase to actually dial our own SIP phone; to do this we create a new context called `dialout` that routes any number to our extension:

```
[dialout]
exten => _X.,1,Dial(SIP/500)
exten => _X.,n,Hangup
```

We reload Asterisk to pick up the dial-plan changes. Then we log-in in WombatDialer and configure it, so that:

- `AS` is our Asterisk server
- We create a trunk called "Out" on server AS that points to our telephone. We enter `Local/${num}@dialout` as its dial string and set its capacity to 1 (so we never receive more than one call)
- We create an end-point called "Msg" on our server AS with extension `100@telecast` and a capacity that is enough for the campaign - let's say 10 lines.

Your configuration will look like the following screenshot:

At this point, we create a new List called "Test" and add only one number to it; you may enter the

number as:

```
0916300000,user:10,friends:100
```

This uploads the number and associates the variables "user" with value 10 and "friends" with value 100.

Then we create a new campaign - this is where all the pieces are tied together:

- We set its name as "Runme" - avoid long names or spaces if you plan to control it externally
- We set its priority to 10 - the priority is the order in which campaigns are queued when trying to assign free lines. A campaign with a lower number will run first, while campaigns with the same priority will have a fair share each.
- We set "Idle on termination?" to Yes - this way this campaign will not just stop when it is out of numbers but will wait for more.
- We set "Additional logging" to "QM compatible" so that you may use QueueMetrics to keep track of it.

After saving the campaign, we select it and then add:

- Trunks: Out
- Endpoints: Msg
- Lists:Test

As you can see, a campaign can have multiple trunks, multiple end-points and multiple call lists, and they may be shared between multiple campaigns.

It would be fair to add some rescheduling rules as well - for example, if we do not get an answer within 30 seconds, we want the system to retry placing the call exactly once after two minutes.

In the end, you would get a situation similar to the one here:

At this point we're ready to go:

- make sure that the WombatDialer engine is turned on (from the Home Page, click on the "Play" icon)
- go to the "View Live" page and select our campaign "Runme" under "Available campaigns"
- click on "Start"

If all goes well, within a few seconds you should receive a call to your SIP phone telling you that user 10 has 100 friends. Hooray!

After this, you should see your campaign being shown on the Live page in gold, with status IDLE; now, when our internal tracking system detects new friends for one of our valued customers, all it needs to do is to send a HTTP request to the WombatDialer, like we would manually from the command line:

```
curl "http://server:8080/wombat/api/calls/index.jsp?
   op=addcall&campaign=Runme
   &number=0916309765&attrs=user:107,friends:123"
```

If you do, in a few seconds the dialer will send this new call. You can send all the calls it needs to place, one at a time, and it will try scheduling them as soon as possible given the current system conditions, running campaigns and available outbound lines.

Now, if you want it to actually dial out and not just call our SIP phone, edit your Trunk "Out": set the dial string to something like `SIP/myprovider/${num}` and set its capacity as the total number of parallel calls you want to place. Then go to the Live page and reload the campaign.

When you want to stop the campaign, you have two choices:

- You can temporarily pause it
- You can remove it from running campaigns when it is paused

Of course, you can have this campaign run on multiple trunks and multiple endpoints, using multiple separate Asterisk servers, just by creating the relevant items and telling the campaign to use them. This way, handling hundreds of channels is just as easy as testing with your SIP phone!

Further evolution:

- You could create your own audio recordings - the example here was created with sounds present in a standard Asterisk distribution, but of course you should record your own messages.
- You can do even better and embed a Text-to-Speech engine script, so you are not limited to inserting numbers but can play back nearly anything
- You can add an IVR option to the "telecast" context, so that if the callee wants to talk to a live person, they are sent to a queue. By monitoring the number of lines that you are using on your trunk and the number of available agents, the WombatDialer acts as a simple progressive dialer.
- As all the configuration is GUI-agnostic, you can use your favorite Asterisk configuration GUI to create End-points - play messages, read IVRs, add time-dependent rules like you would for incoming calls. All you need to know is the point in the dialplan that they start from - e.g. internal number 123 in FreePBX is always available as `123@from-internal`.
- You can optionally add an *Active Period* to the running campaign, so that calls on it are placed only - say - between 9 AM and 6 PM no matter when the messages are queued.

Helping Wombats one carrot at a time

You know how it goes; so many good ideas in real-life end up being limited by the amount of funds you can raise to implement them. That's why Vicky, the energetic new president of "Friends of the Wombat", called you. "Friends of the Wombat" is a nonprofit institution that helps wombats in need, but their activities are limited by the rising cost of carrots; so they decided to start a volunteer fund-raising campaign.

What they would like to do is to call a list of known contacts that expressed an interest in wombats and ask them to make a donation. They started doing this manually with paper and pencil, but getting to a lead is really tiring and time-consuming - their volunteers spend most of the time dialing numbers and it is really hard for them to get through to somebody who is interested.

After a good cup of green tea, what you propose to do is to use WombatDialer to automate the process by dividing it into multiple stages:

- Create an electronic list of those leads
- Leads from their list are dialed
- If the call is answered, a message is played that explains what the campaign is for
- If the callee is interested, they press 1 to be put in conversation with a volunteer that will explain how to send a donation.

This setting leads to two huge efficiency boosters:

- As numbers are dialed automatically and error conditions are handled behind the scenes, a lot

of drudge work with paper and telephone keyboards is automated; no more post-it notes to recall a busy number!

- As they noticed that about 50% of the calls made manually result in calls to busy numbers or numbers where nobody picks up, and that only 50% of the callees are actually interested after an initial interview, they expect that 75 calls out of every 100 they make can be screened out automatically. So if they have 4 volunteers on shift, WombatDialer could start 16 calls at once on average, and have all of our volunteers busy most of the time with people who are actually interested in contributing.

In order to implement such a campaign with WombatDialer, we start by creating a call queue that will hold our agents and will send them calls. You can use any Asterisk GUI to do this - in the example below we use Elastix, but you can choose the one that suits you best.

We go to *Queues → Create new*, set the extension for the queue as 999, enter any name you want, and look on the settings below so that:

- Strategy is set to "rrmemory"
- Queue events: yes (this is very important - if you don't do this Wombat won't be able to observe your queue at all).
- Autofill: yes (all free agents are assigned calls at the same time)

If you create a queue manually without using a GUI, configure it with the parameters below so that WombatDialer can observe it.

```
[999]
autofill=yes
eventmemberstatus=yes
eventwhencalled=no
maxlen=0
strategy=rrmemory
```

Now we create a custom piece of dialplan to be called as an end-point for calls that connect successfully.

```
[friendscampaign]
exten => 100,1,Answer
exten => 100,n,Playback(campaign_message)
exten => 100,n,Read(type,,1)
exten => 100,n,GotoIf($["${type}" = "1"]?ok)
exten => 100,n,Goto(1)

exten => 100,n(ok),UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:1)
exten => 100,n,Queue(999,,120)
```

(In order to make this example shorter, we assume you have a basic familiarity with WombatDialer concepts such as Campaigns, Servers, Trunks and End-points.)

Now log-in to WombatDialer and create a new End-Point:

- Type: QUEUE
- Queue: 999
- Max channels: 10
- Boost factor: 2.0
- Max waiting callers: 2

This means that WombatDialer will try and place two calls for each agent available, but never more than 10, and will stop placing calls if there are two or more people waiting in queue. We expected to use a boost factor of three to four given the previous statistics, but it is better to start with a small figure and grow it if needed.

It is important to understand the impact of different boost factor settings; in general having more calls made than agents may result in calls waiting in queue when all of your agents are busy (in call-center parlance this is usually referred as "nuisance calls"). WombatDialer tries to minimize the impact of this case by continuously monitoring the number of calls waiting on the queue and avoiding placing new calls if there are too many. If you don't want to have cases where calls are not immediately connected to an agent, you should leave the boost factor to 1, that is, place no more calls than available agents. The trade-off here is that agents end up being underused, as they have to wait for a successful call to come in.

We then create a test trunk, upload a list of test numbers and create a campaign called "friends" as in the previous examples; we now start the dialer and start the campaign from the Live page. When we start running it we notice that it does not seem to work - the campaign is running but no calls are placed. How comes?

If we reload the current Dialer status, we see that it is saying that the queue has 0 channels available - this is because there are no agents on the queue.

If we log an agent on (e.g. by entering 999* on Elastix), we will notice that WombatDialer starts dialing. If you pause an agent or log him off, WombatDialer will immediately react and adapt to the changed number of available end-point channels.

Another thing that we do is to track (through a status code) which callers ask for being put in contact with a volunteer - this allows easy inspection of what goes where from the Campaign Report panel.

Now all we have to do is to set our campaign to use an actual telephone trunk and upload the "real" list of numbers and we are ready to go - it's time to buy some new carrots for our wombats!

Understanding statistics

If you run a report of our campaign with a queue analyzer like QueueMetrics, you will have two different views of the campaign:

- if you analyze the queue "friends" (the one that matches the name of your WombatDialer campaign) you will see the total "external" system activity - all calls placed on the campaign are tracked, and the wait time matches the external wait time (that is, the time between the dial

request and a successful connect). All calls appear to be connected on the end-point, as you could have multiple ones assigned to the same campaign.

- if you analyze the queue "999", you will see the human side of action - how many calls were queues for humans to interact with, how fast they were answered and by whom.

Further expansion

- You can use the same end-point in multiple parallel campaigns, if needed.
- It is possible that your campaign reaches someone who is interested but is currently doing something else so does not have the time to speak to your volunteers. They could tell you by hitting 2 - you could add a Reschedule Rule that reschedules the call in a few hours.
- You could easily generate personalized messages instead of one single recorded message by using a Text-to-Speech synthesizer

Please note that outbound telemarketing, whether it is of the good or evil persuasion, is regulated by your local law, so be sure you comply to its terms before you start calling millions of people!

Outbound IVRs and dr. Strangelove

A Dr. Strangelove just called, saying he needs your help for an automated appointment reminder system. Dr. Strangelove is tired of patients forgetting appointments, so he needs a way to call them the day before and making sure they will be there at the right time. Also, as he specializes in every possible thing, he needs to know what the appointment will be about so he can either have an operating room ready to remove your appendix or his psych couch cleaned and stocked with a large supply of paper towels. Do you think you can help?

After calling him, you understand that he wants a system that will not only connect to a list of numbers and handle common issues (busy calls, non-answers, etc) but also a system that is able to detect whether the callee actually confirms receipt of the message. If they do, that’s okay; if they don’t, a new call is placed after a while.

He also wants a system that is able to synthesize a custom message for each call, and that is able to gather data from the callee and pass it along to his office management system as it is collected.

Doing this with WombatDialer is easy; it is basically a matter of implementing an outbound IVR and tracking call parameters and call completion codes. Doing so will also let us show how WombatDialer handles call retries. As an added bonus, we’ll see how WombatDialer notifies other systems over HTTP.

To get us started, we create a list of telephone numbers called "Appointments" with custom attributes; in order to do this, we log in to WombatDialer, go to the Lists page and create a new list. After creation, we select it and upload a list of numbers like the following one:

```
5551234,HH:10,MM:30
5556785,HH:11,MM:00
5552012,HH:11,MM:30
```

This means that the person at number 5551234 is to be reminded an appointment for tomorrow at 10:30. We use two separate variables as this makes life easier for our Text-to-Speech engine.

We now have to program the outbound IVR context in Asterisk; we can do that easily with the help of WD call attributes so that we know what we have to tell our customer. In this example, we will also use the Google Text-to-Speech engine to synthesize audio on-demand.

```
[drstrangelove]
exten => s,1,Answer
exten => s,n,Set(TIMEOUT(response)=5)
exten => s,n,UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:0)
exten => s,n(start),agi(googletts.agi,"Your appointment with doctor
                strangelove is for tomorrow at ${HH} ${MM}")
exten => s,n,agi(googletts.agi,"Press 1 for to book for major surgery
                press 2 for psychiatric counseling.")
exten => s,n,Read(type,,1)
exten => s,n,GotoIf($["${type}" = "1"]?appe)
exten => s,n,GotoIf($["${type}" = "2"]?psyc)
exten => s,n,Goto(start)

exten => s,n(appe),UserEvent(ATTRIBUTE,Uniqueid:${UNIQUEID},APPT:APPE)
exten => s,n,agi(googletts.agi,"You choose the appendectomy.")
exten => s,n,Goto(confirm)

exten => s,n(psyc),UserEvent(ATTRIBUTE,Uniqueid:${UNIQUEID},APPT:PSYC)
exten => s,n,agi(googletts.agi,"You choose psychiatric counseling.")
exten => s,n,Goto(confirm)

exten => s,n(confirm),agi(googletts.agi,"Thank you! Now press 1
                to confirm the appointment")
exten => s,n,agi(googletts.agi,"or 2 to cancel it.")
exten => s,n,Read(conf,,1)
exten => s,n,GotoIf($["${conf}" = "1"]?ok)
exten => s,n,GotoIf($["${conf}" = "2"]?ko)
exten => s,n,Goto(confirm)

exten => s,n(ok),UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:1)
exten => s,n,agi(googletts.agi,"The appointment was confirmed.
                See you tomorrow.")
exten => s,n,Hangup
```

```
exten => s,n(ko),UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:2)
exten => s,n,agi(googletts.agi,"Boo the appointment was canceled.")
exten => s,n,Hangup
```

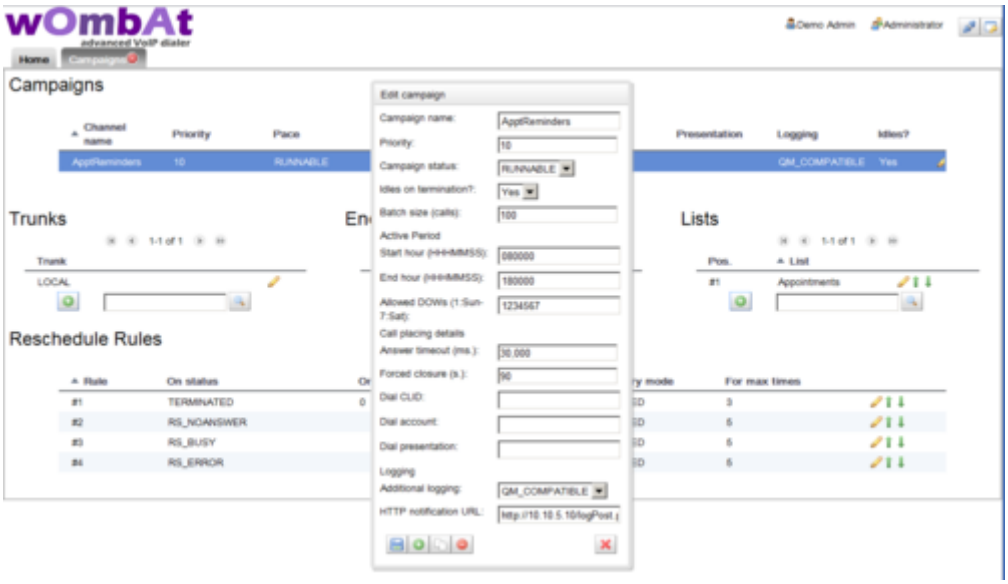
Notable points in the code above are:

- Custom call attributes `HH` and `MM` are passed along with the phone number so we can know what to tell each client.
- We set the call status to "0", "1" or "2". "0" means that the call was connected but no choice was made, "1" means that the appointment is confirmed and "2" is that it is canceled
- We generate some UserEvents in order to populate the outbound (that is, coming from outbound) attribute `APPT` with the patient's choice

We now have to create an end-point for our campaign that points at extension `s@drstrangelove` so that WombatDialer knows where to connect successful calls. We will also need at least one trunk to send calls through - to get us started it is advisable to have a "dummy" trunk routing everything to a local extension.

If you have not already done so, it would be advisable at this point to have a look at our previous tutorial - A social media dialer - to see how to create and control a simple campaign.

When done, we create a new Campaign to connect all pieces together.



We set the campaign to be idle on termination, so that we can add more numbers over HTTP when they become available. We also program it to be running only between 8AM and 6PM, every day of the week, so we don't call people in the middle of the night if a nightly job is used to load new appointments every day.

We also set a *forced closure* after 90 seconds so that if the call exceeds a duration of 90 seconds, it is automatically hung up to prevent using valuable resources on a call that is likely invalid.

As a last measure, we set the logging format to QM_COMPATIBLE (so that we can observe activity via QueueMetrics) and enter the HTTP notification URL of Dr. Strangelove's office management

system to be notified of call events (see below for more information).

We then complete the campaign by adding a trunk, our new end-point with the outbound IVR and our Appointments list.

As a last step, we create a set of Reschedule Rules that implement the retry logic.

- If the call is unanswered, busy or in error, we retry every 300 seconds for up to five times.
- If the call times out because of a forced closure, we retry after 120 seconds.
- If the call completes naturally but its extended status is 0 (no choice), we retry it for up to three times after 300 seconds each.
- If a call completes naturally but its extended status is not 0, then it is not retried.

Now it is time for us to start the campaign - make sure the WD is running, go to the Live page, select your campaign and run it. If all goes well, you should start receiving calls on your test extension.

While the campaign is running, you can add more calls to it via HTTP by issuing:

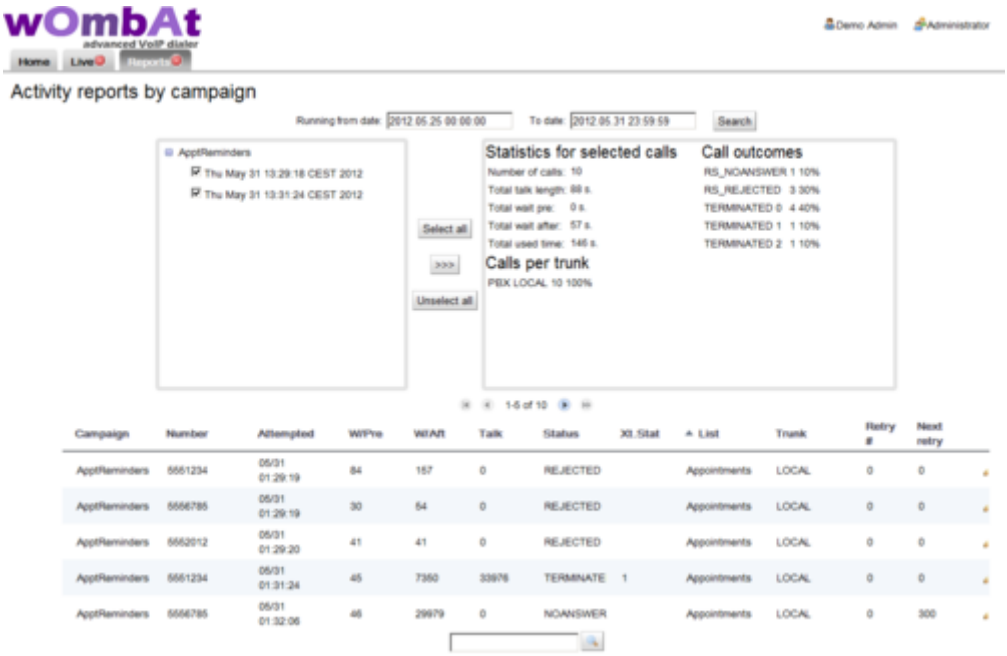
```
curl "http://server:8088/wombat/api/calls/index.jsp?
 op=addcall&campaign=AppRemiders&number=45678
 &attrs=HH:12,MM:15"
```

and you can even specify a minimum time for calls to be placed at, like e.g.

```
curl "http://server:8088/wombat/api/calls/index.jsp?
 op=addcall&campaign=AppRemiders&number=45678
 &attrs=HH:12,MM:15
 &schedule=2012-06-01.10:00:00"
```

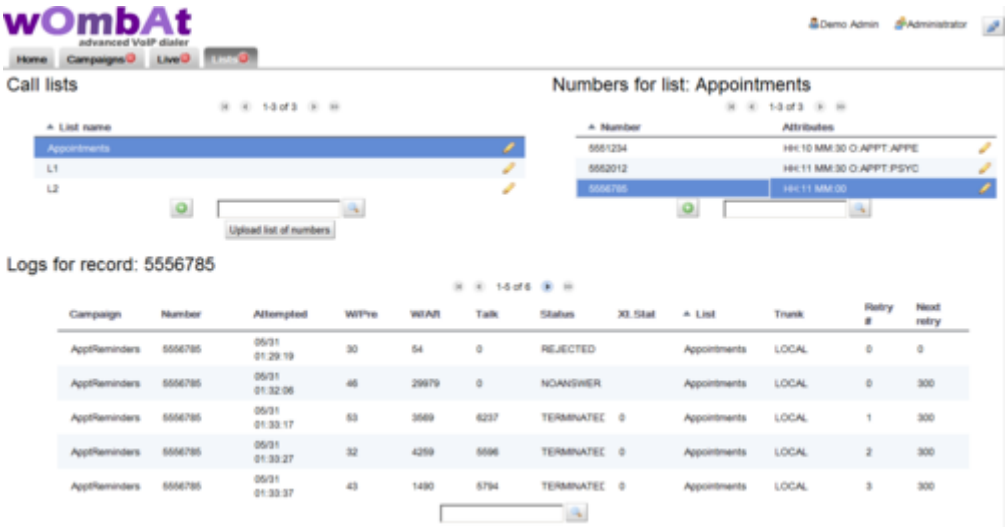
When the campaign terminates, it will be in IDLE state. In order to close it, first pause and then remove it.

After a successful run, you can see its statistics, by viewing the Campaign Report screen:



You will see that some calls appear as TERMINATED 0, some as TERMINATED 1 and some as TERMINATED 2, based on the extended call status entered through a user selection. Only calls in state TERMINATED 0 are retried.

You can also see the state of attributes for each call by going to the List editor:



You see that calls successfully placed will have an APPT attribute that is either `APPE` or `PSYC`; also you will see a complete log of activity for each number.

As a last item, you'll remember we enabled HTTP notification. This basically POSTs the result of each call to a HTTP server, where you could have a simple PHP script to parse it, like in the following example:

```
<?
$out = "";
foreach($_POST as $name => $value) {
   $out .= "$name:$value ";
}
print($out);
```

```
error_log("RQ: $out",0, "", "");
?>
```

The script above basically logs all activity on the HTTP error log. What you get is a sequence of calls like:

```
RQ: num:5551234 reschedule:0 I_MM:30 extstate:
     state:RS_REJECTED I_HH:10 retry:0
RQ: num:5556785 reschedule:0 I_MM:00 extstate:
     state:RS_REJECTED I_HH:11 retry:0
RQ: num:5552012 reschedule:0 I_MM:30 extstate:
     state:RS_REJECTED I_HH:11 retry:0
RQ: num:5551234 reschedule:0 I_MM:30 extstate:1
     state:TERMINATED I_HH:10 O_APPT:APPE retry:0
RQ: num:5556785 reschedule:300 I_MM:00 extstate:
     state:RS_NOANSWER I_HH:11 retry:0
RQ: num:5552012 reschedule:0 I_MM:30 extstate:2
     state:TERMINATED I_HH:11 O_APPT:PSYC retry:0
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:1
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:2
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:3
RQ: num:5556785 reschedule:0 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:4
```

You see that for each call:

- **num** is set to the number dialed
- **state** is the call state at its completion
- **extstate** is the call's extended state, if present
- **retry** is the retry counter
- **reschedule** is set to the time to be waited before a reschedule; if no reschedule is necessary, it will be set to zero
- all **inbound attributes** (the ones you set with the telephone number) are passed along prepended by I_
- all **outbound attributes** (the ones you read from the callee), if any, are passed along prepended by O_

This way you can easily create an integration script that stores the results of the call on a database or passes them along for further processing.

Further developments

- By connecting multiple trunks and end-points residing on multiple servers you can scale this

example up to hundreds of parallel lines

- If you have clients living in different time zones, you could have multiple campaigns active with different time windows to place calls
- It is often a good idea to set call attributes that are not actually used by Asterisk (e.g. a patient ID) but make life easier for third-party systems to find out what the call was about.

The Google-TTS script used is available at: http://zaf.github.com/asterisk-googletts/

Understanding queue end-points

An end-point of type queue in Wombat tries to determine the number of available channels based on the number of available agents on the queue it is observing. An agent is considered available if it is logged on to the queue, is not paused and is not currently in conversation.

After getting the number of available agents, it multiplies it by the "boost factor" (that is used to account for the success rate in connecting the numbers to be dialed) and tries to schedule as many calls. Therefore, if you have e.g. 7 available agents and a boost factor of 1.3, it will try to connect to 9 numbers at once (7 x 1.3 = 9.1).

It will also enforce two limits:

- If there are more than "max waiting calls" waiting on a queue, it will not try and place new calls - in theory there should never be calls queued, as they follow the number of available agents, but it is possible that either some agent logs off after being counted to place calls or some calls reach a queue without passing through Wombat. This also acts as a counterbalance to high "boost factor" values.
- it will never place more than "max channels" calls on the queue - this lets you use a shared queue where some calls come from inbound activity and some come from Wombat itself. Of course you can turn this off by setting "max channel" to a value higher than the total number of agents on the queue.

It is also important to notice that the queue is used only as a way to know how many calls can be placed on a queue, but calls will still be transferred to a point in the dial-plan that should lead to the queue. This lets you execute dialplan logic (e.g. playing pre-recorded messages or running IVRs) on the calls it just placed.

In order to use WombatDialer effectively with a queue, the following guidelines are best followed:

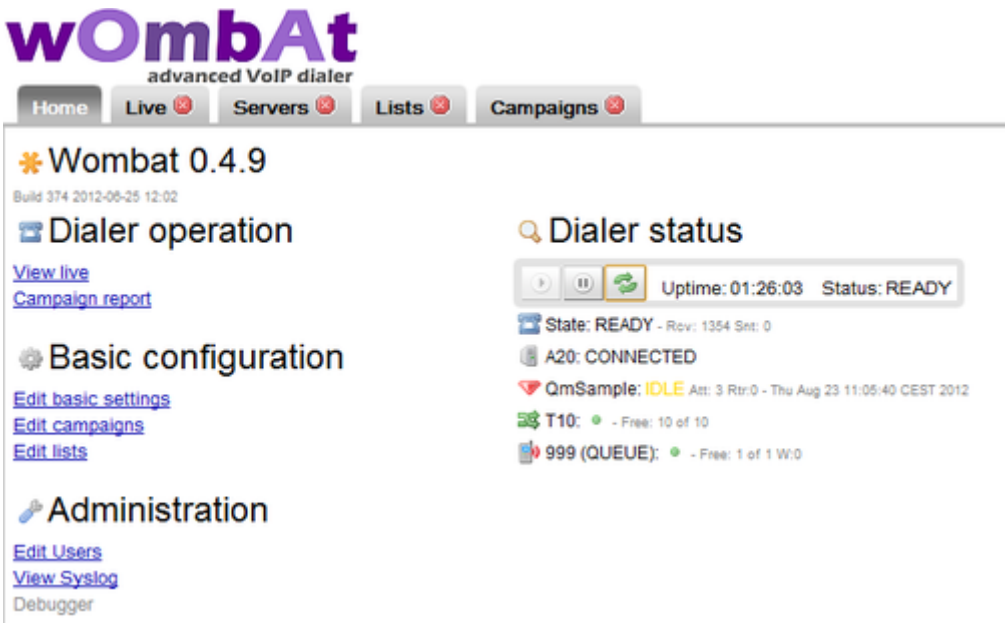
- though Wombat would work with static member channels, if you want your calls to go through to agents who may or may not be available (e.g. some days they may be sick) it is strongly advisable to use dynamic agents who log on and off from the queue.
- as an agent cannot be physically available at all times during the day, it is important that they have a way to pause themselves, be it to run "wrap up" activities after calls or to take breaks. The QueueMetrics web interface offers an excellent panel that lets you add pause codes as well
- the queue must provide events to Wombat about agent activities. On modern Asterisks (12+) it works natively; if not, you must set `eventswhencalled=true`, otherwise the queue will be unobservable. It is also important that extension presence is correctly observed - e.g. if an end-

point is busy because the agent is doing a personal call, its queue status should immediately reflect this. Whether this happens or not on your system is a matter of Asterisk version and type of channel that is used to reach the agent - with recent versions of Asterisk and SIP channels this should work automatically.

- the queue should connect calls to agent as efficiently as possible when there are multiple calls waiting and multiple available agents, so it should have the "autofill" option set to true.

In order to run a campaign with a Queue endpoint, it is best to:

- create a campaign with a Queue endpoint. You may create it IDLE and add no call lists, so the campaign does not actually do anything until fed some numbers.
- run that campaign
- reload the Dialer status in order to see if the queue is being observed (you have to click on the reload icon manually each time).
- If the queue is present, you should see it something saying "Free 4 of 7 W:2". This means that WD is seeing 7 agents connected of which 4 are free (where 4 is the result of multiplying the actual number of observed channels by its boost factor), and that there are 2 calls waiting on the queue.
- try and log on, log off, pause and unpause an agent. You should see the number of free and available channels change accordingly. Try also sending calls to the queue and see if the number of free agents and of waiting calls is correct.
- try also placing calls from some agent extensions and see if the number of free channels reflects this correctly.
- if you plan to have agents working on multiple queues at once, run the tests above while the agents are logged on in at least two queues and make sure statues are updated correctly.



The Queue EPs let you use WD as a powerful progressive dialer and can lead to very complex integration scenarios, but as it involves actual people being called and answering the phone, it is better to understand it well before actually using it in production.

A custom QueueMetrics integration

The WombatDialer can be used as a stand-alone product for message broadcasting, but it was built to integrate easily with QueueMetrics, the premier call-center monitoring and reporting tool for the Asterisk PBX. WombatDialer and QueueMetrics do different things, but together can be a very powerful call-center solution.

- WombatDialer is able to use a Queue as and end-point in order to connect calls to a set of agents
- QueueMetrics is able to monitor extensively the queue and provides a convenient agent interface that works well with WombatDialer

In this example, we improve the scenario described in a previous tutorial - Helping Wombats one carrot at a time - imagining that the want to:

- use a custom CRM interface so that agents can immediately see the name of the person being called and can open up an external CRM application for each client so they can gather donation data
- monitor each agent's performance
- use a dynamic list of people to be called so that as soon as you know of an interesting lead, you can add it to the list of persons to be called.

You can have WombatDialer and QueueMetrics installed on the same server, unless you have a very high load or a very high number of lines.

As a first step, we'll have to create a small interface script that will be our CRM application. We will call it wombatPopup.php and will store it on one of our servers as `http://10.10.5.10/lenz/wombatPopup.php`

```
Plain WombatDialer Agent Panel<hr>
<?
$number = $_REQUEST["caller"];
$agent  = $_REQUEST["agent"];
$unique = $_REQUEST["unique"];

preg_match('/#(\d+)\s(.+)/', $number, $matches);
$id = $matches[1];
$name = $matches[2];
?>

Person is: <?= $name ?> (ID #<?= $id ?>)<p>
Agent is: <?= $agent ?> <p>
Call unique is: <?= $unique ?>
```

This page basically displays the person's name and ID, so we can display it immediately - or this could redirect to an external CRM application.

At the Asterisk level, if you have not already done so, create a queue "999" with the correct parameters to be monitored by Wombat - see Understanding queue end-points . Then edit your

extensions_custom.conf file so that you have an extension "1235" that leads to the queue:

```
[from-internal-custom]
....
exten => 1235,1,Answer
exten => 1235,n,Set(CALLERID(num)=#${ID} ${PERS})
exten => 1235,n,Goto(from-internal,999,1)
```

This rewrites the incoming caller-id for the queue to e.g. "#1234 John_Doe", so this is what the agent sees - even before the agent pop-up is opened. This also is recorded in QueueMetrics, so this is what you will see when you run call reports. Note that if you run FreePBX you cannot call the queue directly using the Queue() command but you'll have to go through its dialplan in order to have all parameters correctly set.

In QueueMetrics, set the following properties in configuration.properties:

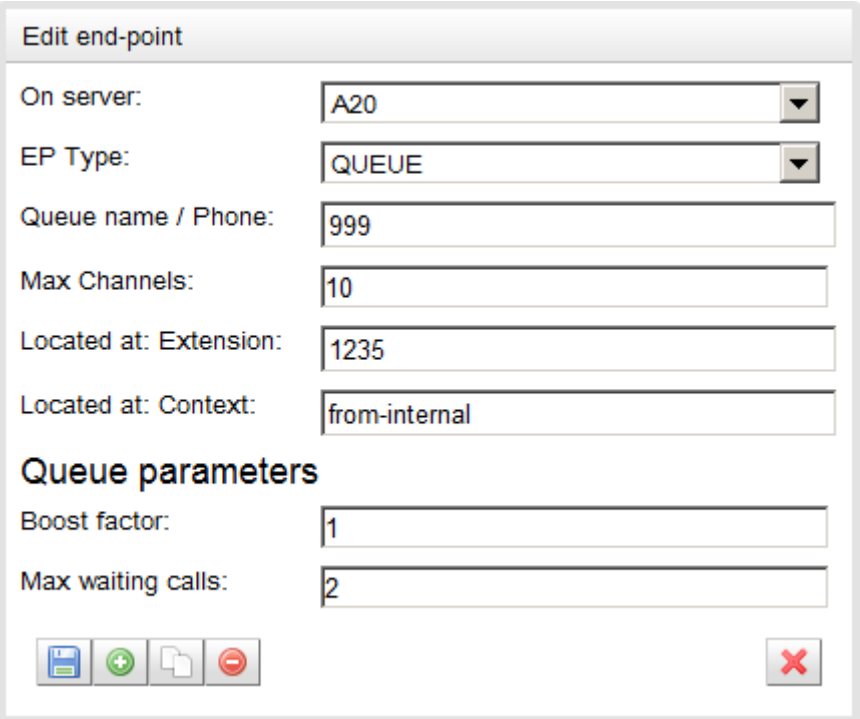
```
realtime.agent_autoopenurl=true
default.crmapp=
```

Then create a queue "999" and set its Queue URL to:

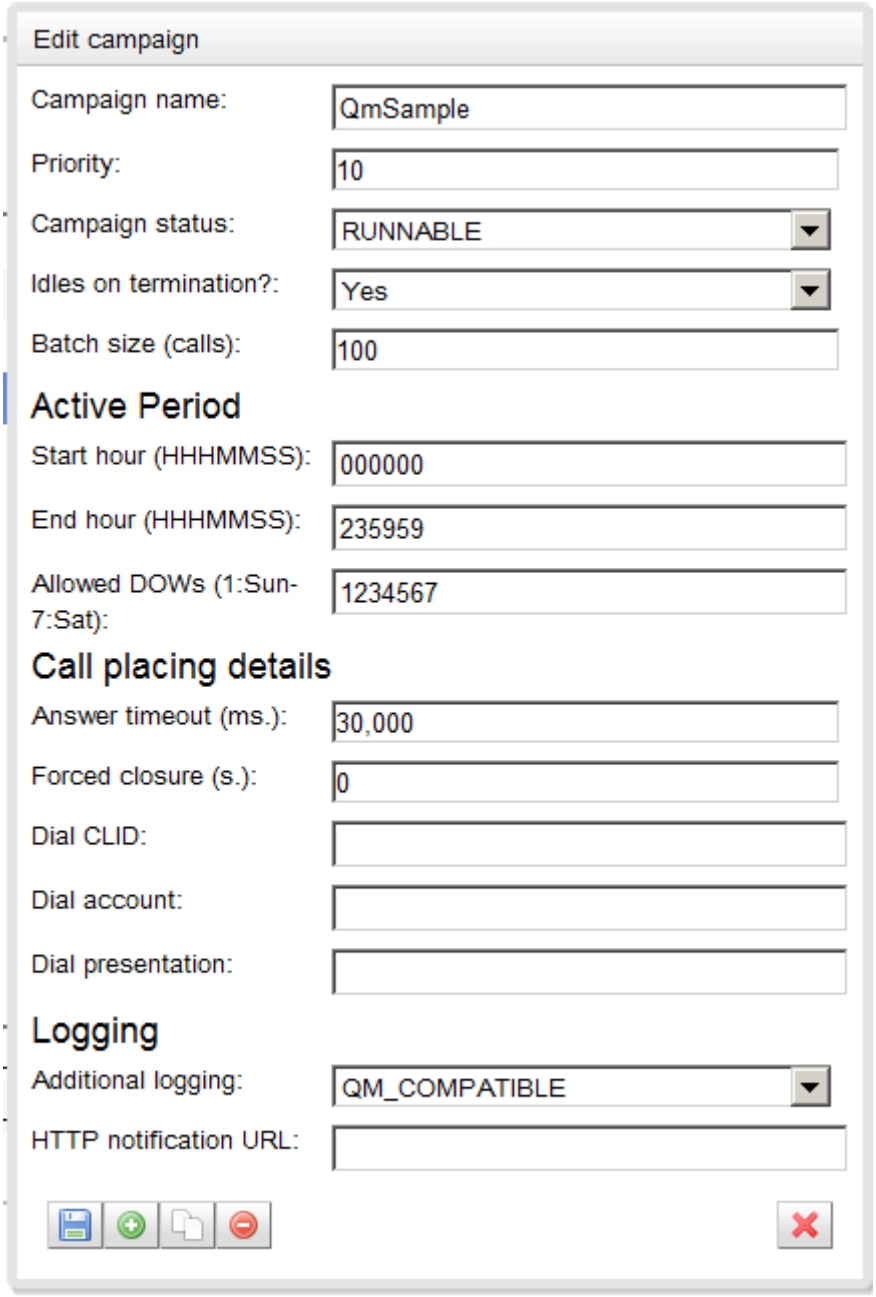
```
http://10.10.5.10/lenz/wombatPopup.php?caller=[C]&unique=[U]&agent=[A]
```

This tells QueueMetrics that it has to enable a screen-pop for calls coming in through that queue.

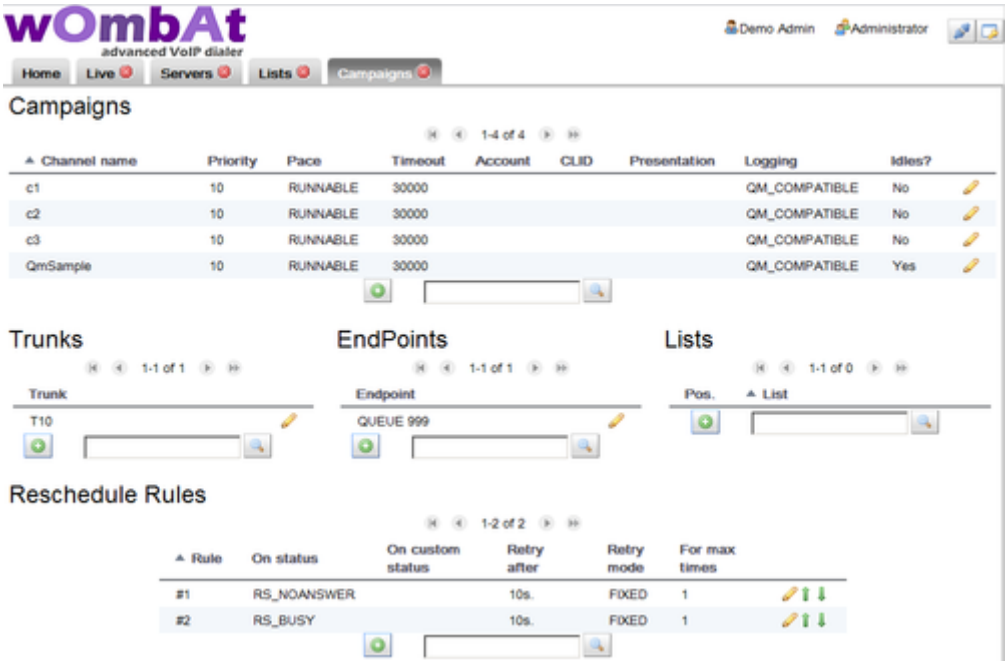
Now log on to WombatDialer; create an EndPoint of type Queue set for queue 999, and its entry point in the dialplan to `1235@from-internal-custom` . Do not forget to set the "Boost factor" to 1 and the "Maximum queue length" to 2.



Then create a campaign called "QmSample" that idles on termination.



Set the trunks, the EP we just created, add no lists and define any reschedule rules you need. The end result should look like the following:



Now start the campaign. It will go in status IDLE immediately, as it has no numbers to dial, and you will see it all yellow in the Live page of WombatDialer.

At this point, it is time for your agents to log in in QueueMetrics. Have them log-in to their Agent's page, and from there log them on to queue 999. At this point, make sure that each agent's browser allows QueueMetrics to open pop-up windows (this is usually disabled in modern browsers).

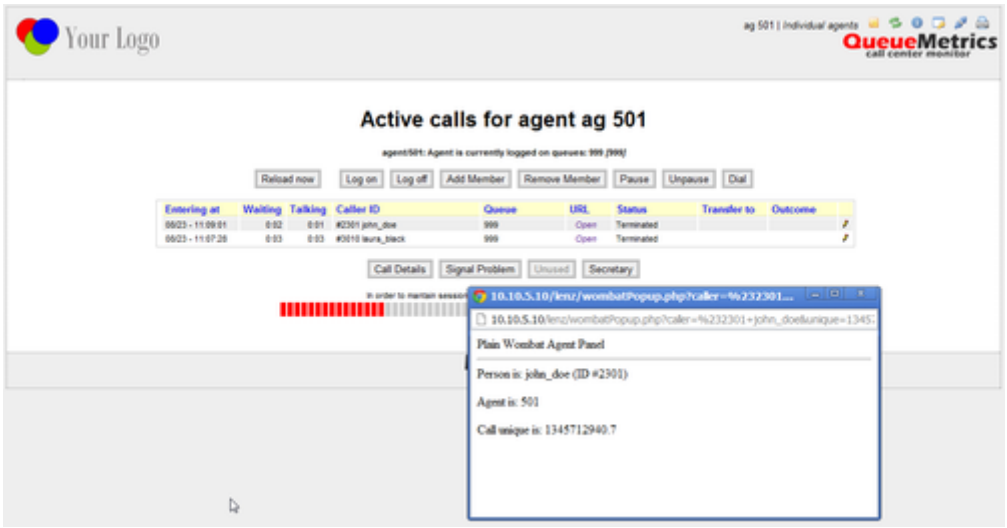
Now, for each call you'd like to be started, send an HTTP request to WombatDialer that looks like the following one (we do this through the Linux shell, but there is nothing preventing you to script this in some other way:

```
curl "http://10.10.5.18:8080/wombat/api/calls/index.jsp
   ?op=addcall
   &campaign=QmSample
   &number=2000136
   &attrs=ID:2301,PERS:John_Doe"
```

Note that you associate two parameters: ID that is a unique id used by your CRM and PERS that is the called person's name. You could also add additional parameters you may need, e.g. a person's class or their telephone number.

What happens now is that when the call goes through, its caller-id is rewritten as "#2301 John_Doe" as soon as it reaches Asterisk. When your agent receives the call, they will reload the agent's page

and the pop-up will be immediately opened.



In the pop-up, you parse the caller field and extract your CRM id, so you can use this for your external application to display the correct data.

From the agent's page you can also manually open other forms for recent calls. The agent will also see the person's name in QueueMetrics, so this makes their life easier. The agent may also set a manual status code for each call through QueueMetrics, e.g. if the sale was successful or not, and this lets you measure your agent's performance. You may also use QueueMetrics's extensive QA monitoring features to analyze calls and improve the process.

Analyzing outbound campaigns

If you set your campaign to have QM_COMPATIBLE logging, you will find data about two different queues on QueueMetrics:

- If you run a report for queue "QmSample", you will see all calls that WD attempted - the successful as well as the unsuccessful ones. This is basically the activity that WombatDialer performed and the actual telephone usage duration. In a real-life scenario, the vast majority of calls will be unanswered.
- If you run a report for queue "999", you will see activity about calls that were successfully connected and were actually queued to be answered by agents, You would expect to have a very low unanswered rate here - if it is high, it means something is not working as expected. You will of course have some lost calls, e.g. because the called person hung up before the agent was connected.
- The difference between the number of successful calls in "QmSample" and the total number of calls in "999" are calls that were hangup before reaching the queue, e.g. during an initial IVR phase.

Elastix queue call-backs

You are called in to a client site; they seem to have a problem. They run a small (10 agents) inbound call center, and when you join everybody else in the meeting room, there is a large and colorful graph in the middle of the table. The graph shows the call wait times during the week and boy, it's

not a good sight. Their main inbound activity is to offer client support for a company selling sport bikes, and everybody seems to be calling on Monday morning. It looks like people go riding on weekends and whatever problem they have, they call on Monday morning. Wait times peak, abandon rates spike, and nobody is happy. The call center manager is mostly concerned of having to hire and train some temp people in order to handle the load that only happens one day a week. They ask you if you have any better idea on what can be done. And yes, you have some.

You can program an Asterisk queue so that when people tire of waiting, they press a digit and get to a menu where they can leave their number. Then the system queues their call and attempts to call them at a convenient time. This way:

- your customer are happy; they don't have to wait in queue for so long
- your call center manager is twice happy: the first time because wait times and abandon rates go down, the second one because by placing calls at a convenient time they can smooth out the workload of their agents during the day

This scenario requires some additional "glue" to what is basically supported by Asterisk - exiting a queue and reading a number are easy, but then starts the pain. You'll have to create a database and write a script that reads back from it. You have to handle invalid numbers, busy numbers and the like (if we promised to call back the client, we cannot just try once and forget about it). You'll have to have a GUI of some kind for the manager to start and stop dialing. You'll have to adapt to the number of available agents. You'll have to report on this activities. You'll have to avoid flooding the trunks of your PBX with too many calls. In short, it's the kind of thing that gets more complex the more you think about it. That's what WombatDialer is for.

What we plan to do is to use WombatDialer as the call-back engine. It can be controlled by an external HTTP API, so you can do that from the Asterisk dial-plan. It has exact knowledge of the current set-up and call back rules, so you get the number of calls you expect on one or more Asterisk servers. It can work with an existing PBX and does not interfere with calls that are not its own. It keeps track of call completions and knows what to do in case of invalid and busy numbers. It has reports of its own and can work with QueueMetrics for powerful and detailed reports.

The client uses Elastix as PBX system, so we'll have to integrate it with WombatDialer. No problem!

So what we do is:

- First we create a normal queue, for inbound. We call it "400".
- Then we create a call-back queue. If our main queue is called "400", then let's call this second queue "401". The idea is that WD will monitor this queue - when you have members on this queue, then WD will start placing calls. This way an inbound call-center with multiple queues will find it very natural to have some agents join and leave a call-back queue. When you create this queue, make sure you set "Ring Strategy: rrmemory", "Event When Called: Yes", "Member Status: Yes", "Autofill: yes" so that WD can use it effectively.
- we create a piece of dialplan that will handle the exits from queue "400" and will gather the telephone number
- we create a new "custom extension" (399) that will jump in the dialplan at "Local/1@queue-leavenumber"

- In Elastix, we create an IVR menu and set it as a destination for queue "400". This menu has only one option (1) that basically jumps to the custom extension "399" that we just created, in order to call our script
- we go back to the queue "400" and set its "Fail Over Destination" as our IVR we just created

We start by editing the extensions_custom.conf file in our system, adding a new stanza like:

```
[queue-leavenumber]
exten => 1,1,NoOp
exten => 1,n(Start),agi(googletts.agi,"Please enter your telephone
           number and we will call you back.",en)
exten => 1,n,agi(googletts.agi,"The number must be composed of 7 digits.",en)
exten => 1,n,Read(CBNUM,beep,7,,2,5)
exten => 1,n,NoOp( Num ${CBNUM} )
exten => 1,n,GotoIf($["${LEN(${CBNUM})}"="7"]?lenOk)
exten => 1,n,agi(googletts.agi,"The number you entered has the wrong number
           of digits.",en)
exten => 1,n,GoTo(1)

exten => 1,n(lenOk),agi(googletts.agi,"You entered the following number",en)
exten => 1,n,SayDigits(${CBNUM})
exten => 1,n,Wait(1)
exten => 1,n,agi(googletts.agi,"Press 1 to confirm or any other digit to start
           again.",en)
exten => 1,n,Read(CONF,beep,1,,2,5)
exten => 1,n,GotoIf($["${CONF}"="1"]?Store:Start)

exten => 1,n(Store),NoOp
exten => 1,n,Set(WHEN=${STRFTIME(${EPOCH},,%y%m%d-%H%M%S )}
exten => 1,n,Set(PARM=number=${CBNUM}&attrs=orgQ:400%2Cwhen:${WHEN})
exten => 1,n,Set(foo=${CURL(http://10.10.5.18:8080/wombat/api/calls/?
           op=addcall&campaign=callback&${PARM})})
exten => 1,n,agi(googletts.agi,"Thank you! we will call you back as soon
           as possible.",en)
exten => 1,n,Hangup
```

We use Google TTS as a voice synthesizer - you could use a different one or you could have the messages custom-recorded for you. What our dialplan does is first to collect a 7-digit number, then read it back asking for confirmation and when confirmed, it sends it over to WombatDialer on a campaign called "callback". Together with the number, we also store the code of the queue that the call was on and the date and time this number was gathered. (Please note that in order to send multiple comma-separated parameters in the HTTP request, we have to use *%2C* instead of the plain comma ",").

In order to configure WombatDialer:

- We create a trunk called "Trunk" with a dial-string of `Local/9${num}@from-internal` and a capacity of 10 lines. This basically replies all numbers as if they were entered on a local extension prefixed by 9.

- We create an End-Point of type Queue for monitoring queue 401; set extension to "401" and context to "from-internal"; max number of lines to 10; boost factor as 1 and max waiting calls to 2. This means that the number of calls placed will match the number of available agents on queue 401.
- We create a campaign called "callback"; set it to Idle on termination and turn on QM_COMPATILE logging. We add the trunk and the EP we just created. We create a set of reschedule rules in order to handle REJECTED, BUSY, INVALID and NOANSWER calls, e.g. by retrying up to 5 times each waiting 10 minutes between each attempt. Note that we create no lists for this campaign.
- We start the new campaign; having no numbers, it should immediately turn yellow on the Live page to tell you it's idling.

If we start sending calls to the queue and we try and leave any numbers, we will see that a new list will be created on WombatDialer under the name "callback/AUTO" and that will contain the numbers and attributes like:

```
Number:
            5551235
Attributes:
            orgQ:400
            when:121115-153402
```

Those numbers are NOT immediately called, but WD will wait for some agent to be present and active on queue "401" so that they can be called back. This way, the call-center manager can monitor the current call backlog and decide who and when it is to join the callback queue.

Further improvements

- If there is a caller-id on the call, you could ask the caller whether to use it as the number to be called back
- You could add time limits to the WD campaign so that you are sure that no calls are made outside acceptable periods

Automated recall of lost inbound calls

If you run a call center, serving clients in a timely way is often very complex, as it requires having enough people available to handle traffic spikes. The number of callers that disconnect because they have been waiting too long in a queue is then an important driver of the quality of your work, and these frustrated callers are the focus of much attention and scheduling/planning efforts in all call centers. This is because in a traditional setting doing inbound calling you basically had no other way of servicing the client but waiting for the person to call in.

With an Asterisk-based PBX and using digital lines, this scenario changes a bit, as:

- Your average caller has an associated caller-id that often matches a physical phone in their proximity

- Telephone traffic is very cheap compared to the cost of agent time for call handling
- You have ample means of programming the PBX to suit your exact needs

So it is now a conceivable scenario to improve the services you are offering by adding an automated call-back option, so that you search the logs of lost calls and you actively schedule recalls on them in order to get back to people who hung up in frustration.

The plan: automatic queue recalls

In this article, we explain how to implement a basic call-back scenario using QueueMetrics and WombatDialer. What we do is very easy, as in:

- We periodically run a script to gather the caller-ids of lost calls that were handled on a queue
- We check each caller-id as to be sure is a valid number
- We check that there is no subsequent successful call on the queue from the same caller-id (as to prevent recalling people who already retried themselves)
- We schedule those calls for dialing no more than once per number per day

As our dialing schedule happens on a WombatDialer campaign, we can control the flow of calls through it by adding and removing agents supposed to handle outbound traffic, or pausing it completely during periods of high inbound traffic.

Step 1. Configuring QueueMetrics

In order to gather information from QueueMetrics to an external script, we need to enable XML-RPC access credentials. This is usually very easy to do, as QueueMetrics ships with a (disabled) ROBOT login that allows external access.

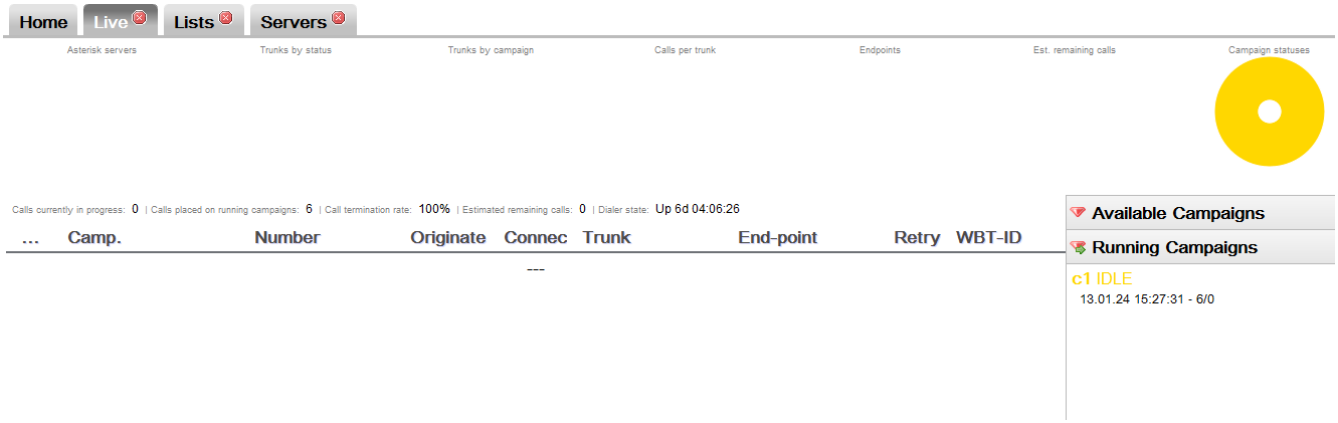
Enabling it is very easy: log in as an administrator, click on "Edit users", edit the "robot" user and set "Enabled" to yes. While you are at it, take a second to change the default password.

Step 2. Configuring WombatDialer

Set up WombatDialer with a queue end-point (as described for example in Elastix Queue call-backs with WombatDialer) and make sure everything is working.

Create a new campaign for calling back people - set its "Idles on termination" property to yes and make the logging QueueMetrics-compatible. This way the campaign can run until needed, waiting for more numbers to be added when idle. Do not add any call list as we will load numbers to be called through the WombatDialer APIs.

Before you start scheduling recalls, your campaign should look like the following one:



You might also want to pause it, so you can decide when to run it.

Step 3. The script

Scripting QueueMetrics and WombatDialer is really easy. It can be done in any language - we chose PHP as it is well known, has good XML-RPC support to query QueueMetrics and is very simple to edit and customize.

We created a sample script that can easily be downloaded from GitHub - as you will likely edit this to suit your needs, feel free to fork a new project and work on that. Our script is available from https://github.com/Loway/OpenWombatDialerAddOns in a folder named "AutoRecall".

The following parameters should be edited in the script:

```
$qm_server = "10.10.5.11";
$qm_port = 8080;
$qm_webapp = "queuemetrics";
$qm_login ="robot";
$qm_pass = "robot";
```

These parameters specify the XML-RPC connector of your QueueMetrics instance.

```
$wbt_url = "http://10.10.5.18:8080/wombat";
$wbt_cmp = "c1";
```

These parameters specify the URL of WombatDialer and the campaign that calls should be added to. The dialer must be running when the calls are added and the campaign should be active (usually IDLE). Note that the campaign you use for call-back might be paused so that call-backs are actually deferred during periods of high activity.

```
$queue = "300";
$lookback = 3600 * 8 ; // in seconds
$allowedPatterns = array(
    "/^555..../",
    "/^0041.+/"
```

```
);
```

These parameters decide which set of queue(s) should be scanned and how long is to look back for the current day. Multiple queues can be used, separated by the pipe character.

The last parameter is a set of regexps that will be used to check the numbers read from QueueMetrics. At least one regexp must match for the number to be queued. This is used to avoid queuing invalid numbers or - worse - malicious numbers.

Step 4. Putting it all together

In order to run the script periodically, you could create a cron-job that runs it every 20 minutes. As number are never recalled more than once and the script keeps an history files of numbers already dialed, you can safely run it over and over again.

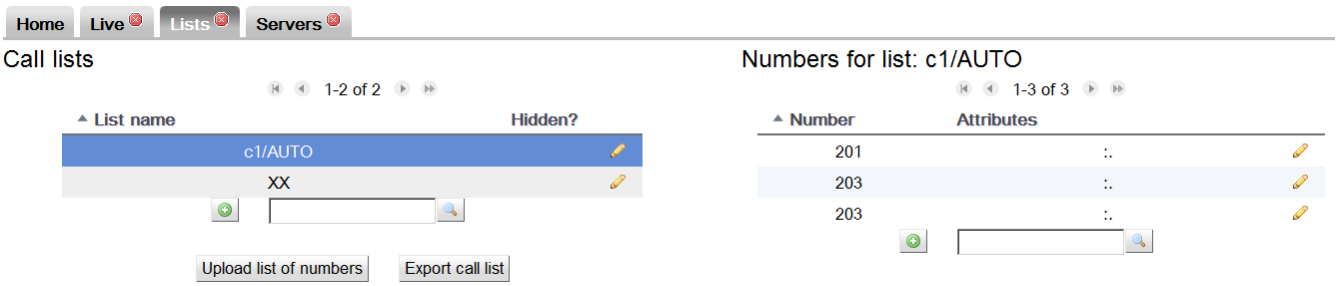
Once tested, a crontab entry like the following one will automate the running:

```
*/20 * * * * /usr/bin/php /root/WombatDialerExamples/AutoRecall/autoRecall.php
```

This is how a simple run looks like - the scripts logs its activity to STDOUT, so you may want to redirect it to some log file for the keeping.

```
$>php autoRecall.php
Finding applicable period
Loading call log file for 2013-01-24
Looking for data between 2013-01-24.07:54:33 and 2013-01-24.15:54:33
 on server '10.10.5.25' - queues '300'
Query took 0 seconds.
# 201 - Last call lost @ 2013-01-24.15:46:39 - Scheduling.
Adding 201 to campaign c1 on WombatDialer.
Saving call log
```

After running this, you should see that new numbers are added to an AUTO call list like the one shown in the following screenshot; and if the campaign is not paused and agents are available on the recall queue, calls will be dialed as needed.



Improving the solution

In order to run this solution in a real-life scenario, you should edit the campaign in order to:

- set up a time window that matches your agents' presence and when it is customarily allowed to recall. For example, even if a call is queued at 11 PM on a Saturday night, a recall might be acceptable only on Monday morning. This of course depends on what you are doing and the local customs.
- set up reschedule rules in order to handle calls unanswered and busy lines correctly. It would be too bad not to be able to recall just because the caller's phone was busy at the moment
- it could also be useful to connect the caller to a reverse-IVR first, so that they get a message like "Hello, we are calling you back because of your call made at 10.30AM. If you'd like to talk to one of our agents, please press 1 now" before being routed to an agent
- a simple addition that could be made to the script would be to set up a minimum wait time to qualify calls; that is, you would recall only people who waited in queue for more than 10 seconds.
- using a technique very similar to the one explained here, it would be trivial to set up campaigns for quality assessment or customer satisfaction, run as reverse IVRs.

Preview dialing with QueueMetrics

Preview dialing is a type of reverse dialing where the agent has a chance to "preview" the number that is to be called before actually having a call placed.

The way it works is:

- The agent logs on to a queue. WombatDialer uses the queue (as it usually does in Reverse dialing modes) to know which agents are available. This makes integration with QueueMetrics very easy.
- The agent asks WombatDialer for a number to call. WombatDialer gets the next number out - considering all its dialing and reschedule rules - and reserves it for the agent.
- The agent uses a GUI where they can see the number to be dialed and typically embeds, or links to, an external CRM app so that they can review the call
- When the agent is ready, they ask for the call to be either placed or skipped.
- If the call is to be placed, the agent is connected immediately and listens to Music on Hold until the callee is on line
- If the call is not to be placed, it is made with a special code
- Reschedule rules apply - so error states are handled correctly

Step 1: Set up your PBX:

In order to run this tutorial, we need to set up our PBX as follows:

- Three SIP extensions: 201, 202 and 203. We will use 201 as the agent extension while 202 and 203 will be used as sample end-points
- One queue, called "999", to hold our agent. Note that calls will NOT be processed through the queue but we will use the queue to keep track of which agents are available. When defining it, we make sure that we set: "Ring Strategy: rrmemory" - "Event when called: yes" - "Member

status: yes" so that WombatDialer can fully observe it.

Step 2: Installing WombatDialer

We install WombatDialer on the PBX system using *yum*. When we log in, we create:

- An Asterisk server that can talk to PBX. We can use the AMI user "admin" with the default password you gave during the system installation.
- A trunk named named "Trunk", which dial string will be `Local/${num}@from-internal`
- An end-point of type QUEUE, which name is "999" (so that it observes queue 999), located at extension "999" context "from-internal" (this is not really needed if you run only reverse campaigns), setting both "Reverse dialing: yes" and "Manual preview: yes"
- We create a list called "Numbers" and leave it blank for now
- We create a campaign called "Reverse", which has logging set as "QM_COMPATIBLE" and which logging code is "999". We add to it the trunk, the EP and the list we just created.

Now we go to the list manager and upload the following list:

```
200,NAME:John
202,NAME:Mike
```

This tells WD to dial numbers 202 and 203, and sets a NAME attribute for each call (that will be useful when previewing).

Step 3: Installing and configuring QueueMetrics

Install QueueMetrics on the same machine as the PBX system by typing:

```
yum install queuemetrics
```

Now log in into QM and configure:

- An agent called agent/201
- A user for agent 201 called "agent/201" password "201" class "AGENTS"
- A general monitoring queue called "999". Set agent/201 in the MAIN level of that queue.

Now edit the "System parameters" and edit the section `realtime.agent_web1` as follows:

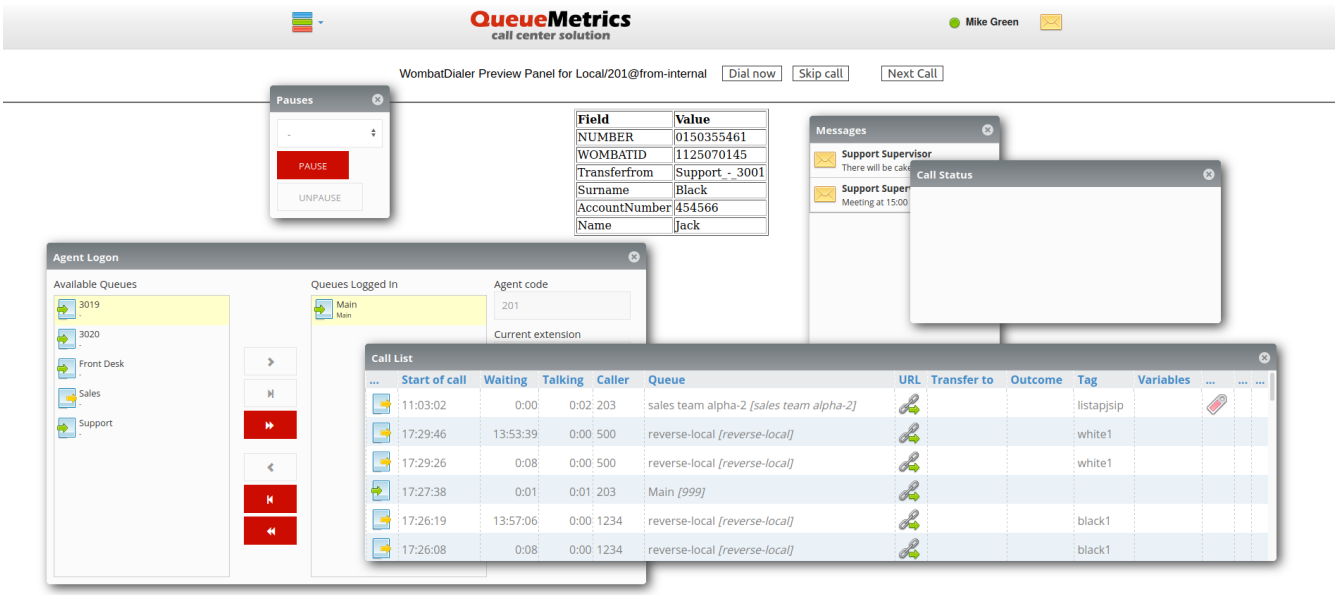
```
realtime.agent_web1_url=http://10.10.5.46:8080/wombat/agents/rd_pop.jsp?agent=Local/[A
]@from-internal
realtime.agent_web1_label=WombatPreview
```

Edit the public IP of your PBX server and replace the "10.10.5.46" address above.

Making it all work together

- Log "agent/201" into QueueMetrics and have him join queue 999 on extension "201". Note that we are using Hotdesking - the agent logs into Asterisk with their SIP extension.
- Start the dialer. From the Live page, start the campaign "Reverse". No calls will be placed.
- Check the dialer status. You should see WombatDialer observing queue 999 and finding "Local/201@from-internal" logged on. Try pausing and unpausing the agent and refreshing the dialer: the status of Local/201@from-internal should change accordingly.

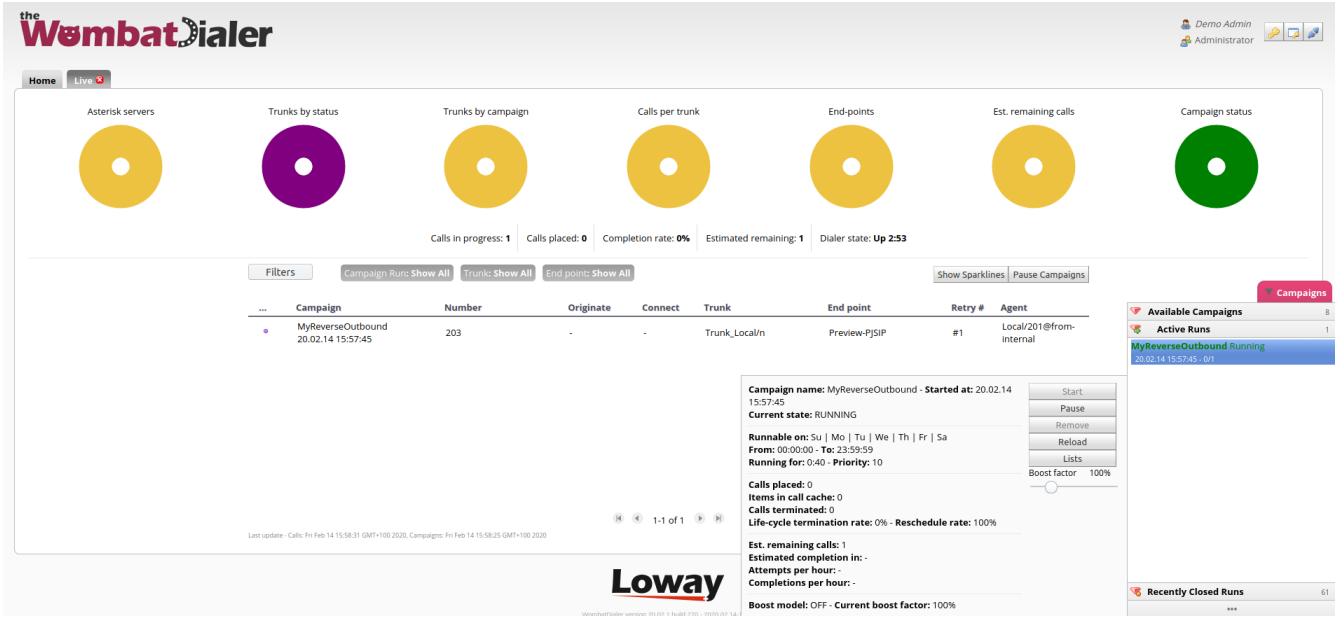
Now click on the button called "WombatPreview" from the Agent's page. You should see a preview panel like the one in the picture:



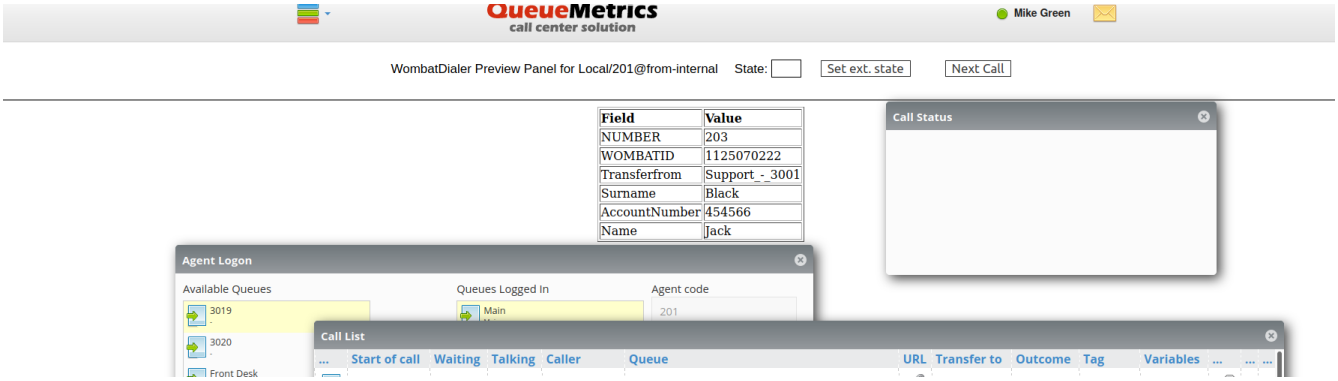
As you can see the Preview panel is displaying the right record.

If you look at the Live page on WombatDialer, the call should appear as "Reserved". When the agent clicks on Dial, the call should start. Once the agent answers, your dialer will try and connect the other extension.

When dialing, the Live page will show what is going on:



And the same thing will happen on the Agent's page:



Nice work, isn't it? :)

Improving the solution

- You can add PSTN numbers to your lists and the will be dialed as if the had been entered on a local extension.
- If you want WombatDialer to dial without waiting for an agent decision, just remove the "Manual preview" check and restart the dialer.
- If you want a customized preview panel, you can create one by using the WombatDialer API.

Effective answering-machine detection

We want to run a simple outbound campaign, in which we plan to play a file and wait for the receiver's acknowledgment by pressing *1*. We also want to make sure that in case the receiver has an Answering Machine, we play a specific audio message that will be recorded by the machine.

The first thing we need is to set up the Answering Machine Detector in Asterisk; this can be done using the embedded application AMD or a number of external tools. In this tutorial we cover the embedded AMD application.

So the first thing we do is to configure the file *amd.conf* so that it has the following contents:

```
initial_silence            = 2500
greeting                   = 1500
after_greeting_silence     = 300
total_analysis_time        = 5000
min_word_length            = 120
between_words_silence      = 50
maximum_number_of_words    = 4
silence_threshold          = 384
```

These are the default parameters used to detect a machine and they may need some tweaking for your local market and spoken language.

The we create a special dialplan extension:

```
[amddetect]
exten => 1,n,Answer
exten => 1,n,Background(beep)
exten => 1,n,AMD(${AMD_EXTRA})
exten => 1,n,NoOp("AMD: ${AMDSTATUS} - ${AMDCAUSE}")
exten => 1,n,GotoIf($["${AMDSTATUS}" = "MACHINE"]?machine)

exten => 1,n,Set(TIMEOUT(response)=5)
exten => 1,n,Playback(/var/lib/asterisk/sounds/custom/${HUMANMSG})
exten => 1,n,Read(ack,,1)
exten => 1,n,GotoIf($["${ack}" = "1"]?confirmed)
exten => 1,n,Hangup

exten => 1,n(confirmed),UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:OK)
exten => 1,n,Wait(1)
exten => 1,n,Hangup

exten => 1,n(machine),UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:AMD)
exten => 1,n,WaitForSilence(2500)
exten => 1,n,Playback(/var/lib/asterisk/sounds/custom/${MACHINEMSG})
exten => 1,n,UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:AMDALL)
exten => 1,n,Wait(1)
exten => 1,n,Hangup
```

And we define an extension of type PHONE that points to *1@amddetect*.

When we answer the call, we play a short beep to make sure that the channel is up, and then we start AMD detection. The call will be blocked until AMD detection is complete; so there is a definite trade-off between quick and accurate detection. To make your life easier during testing, we print the result of the detection and its cause on the Asterisk console.

On the campaign itself, we define two campaign variables that will hold the names of audio files to

be played back in either case:

- HUMANMSG: is the audio file to be played back to humans
- MACHINEMSG: is the audio file to be played to answering machines

Though this is not strictly needed, you can tweak the parameters of the AMD detector by editing the *Extra AMD settings* on the campaigns; these are passed as parameters to the AMD detector; if blank, defaults are used.

The final status for answered calls will be:

- TERMINATED: a call that went to a human that did not acknowledge it
- TERMINATED/OK: a call that went to a human that acknowledged it
- TERMINATED/AMD: a call that went to AMD but was hung up before the message was played completely
- TERMINATED/AMDALL: a call that went to AMD and the message was played completely.

You can use these call statuses to decide how to reschedule those calls, and to create new lists or blacklist some specific numbers.

Fax detection works in a very similar way, but it kicks in automatically when enabled on a channel.

Understanding blacklists

One of the most important features of a dialer, paradoxical as it sounds, is to avoid calling numbers that are not supposed to be contacted. But why should you, in the first place, try to call numbers that you are not supposed to call?

In practice, this is a common scenario - for example, let's say we work for a car dealership, and you prepare a list of customers to contact because their car is ready for pick up after its service ticket. While you are running this campaign, some especially eager customers decide to contact us first to inquire about the status of their vehicle, and you tell them that it's ready. At this point, calling them again would be a waste. It would also give the impression that we do not know about their previous interaction.

But **how** to do it? Once WombatDialer schedules a number, the right thing to do is to cancel the call on the campaign, if the campaign hasn't placed the call already. So, we say, let's delete it from the relevant list, but we may want to re-use the same list of numbers more than once, and we do not want to strike some numbers out of it "just because".

Also, if there callee ever asks "Why didn't anybody call me? I was waiting for your call!", it is important to understand that it was an explicit decision and not a failure. As you have no way of knowing about the cancellation when preparing the list of contacts to reach, it must be an "add-on" while the system is running. That's what blacklists do: we can keep a separate list of numbers we don't want to call and check it dynamically as we progress through our lists.

A second common reason why you could use a blacklist is because the customer directly **asks us**

not to be recalled. By adding them to a blacklist we can keep this information in a separate place from the lists we manage, and WombatDialer makes sure we never forget to remove those numbers from the ones we are supposed to dial.

This way, our employees tasked with extracting call lists need not worry about making sure that all canceled numbers are manually removed, and why. In many countries, you have a legal obligation to adhere to a customer's will never to be called again, so you need to be 100% sure you are doing this process right.

So there are two related but different scenarios: in one case you want to "edit" the current run of your campaign only; in the other, you want to create a long-term database of numbers that are not to be recalled. And, in practice, you likely want to do both things on actual campaigns you are running.

How do blacklists work?

WombatDialer checks a number against blacklists **every time it has to dial it**. If a number is present on any blacklist, then it is not dialed at all and it goes straight to a special state `BLACKLIST`.

As WombatDialer will try dialing an unsuccessful number multiple times and with different rules, a number might then live for a while in a state when it has already been called, but Wombat still wants to dial it in the future. At any time you add that number to a blacklist, all further redials are automatically blocked. This is handy because sometimes recalls may happen after hours or days from the first trial, so it makes sense that we can pull the handbrake on them at any time.

We can even blacklist a number for a specified period only; by setting the attribute `BLACKLISTED_UNTIL` on a call to a specific date, we make sure that the given number is blacklisted only up to that point in time. Similarly, as with plain call lists, the same number can be present multiple times on a blacklist; and WombatDialer makes sure that if it has a time limit, it is enforced. A number with no time limit is considered blacklisted forever.

As you do for lists, you can have multiple blacklists linked to the same campaign - the difference here is that while calls from plain lists are loaded in sequence, one at a time, every number to dial is checked at once against all blacklists linked to that campaign. Having multiple blacklists is handy because you can use them as different "rules" - for example, you could have a temporary blacklist that you use to avoid recalling people that just called in, and at the same time, a general blacklist (maybe shared between all of your campaigns) where you keep customers that are never to be recalled. This gives you a lot of flexibility in how you organize your work.

Plain lists vs blacklists

So, what is the difference between a normal call list and a blacklist? The answer, surprisingly, is "none". Being a blacklist or a normal list depends on the role that the list plays in the campaign. This means that all functions meant to update lists work the same way in both cases.

For example, you can ask WombatDialer to add the current number to a list as a disposition rule at the end of the call cycle. It will simply oblige, and then it is up to you to decide if you want to use that list to place or to avoid further recalls.

Blacklists and multiple numbers

WombatDialer can use a set of numbers to reach out to customers (see Using multiple numbers per call). For example, when calling a person, it can first try their home number, then their mobile, then their office.... as you see fit. In WombatDialer lingo, this is called a `MULTINUM` - and while we use it as the "main" number in the identity, on every attempt the number dialed is rotated in a round-robin fashion.

When blacklisting, you want to stop all calls going to a specific person - so you add the main number to a blacklist, and it will automatically block all further recalls. On the other hand, additional numbers cannot be blocked directly.

This is not usually an issue, because the main number, the one that must be blocked, will be tied to the person's identity on your database.

APIs and integrations

We have so far seen that blacklists, by their nature, tend to be very dynamic. Sometimes you decide which customers to call based on a fixed, given list of numbers, in other cases, you will use the API to "drip-feed" numbers to Wombat when there is a need to call them - for example, a visitor leaving their number on your website might be added to a current campaign. If all goes well, they will be recalled in a few seconds.

Note: all examples below are issued as HTTP GET calls. To maintain readability, we break them into multiple lines and may not quote all characters properly. You should.

This is how you would do it using Wombat APIs:

```
http://127.0.0.1:8080/wombat/api/calls/
  ?op=addcall
  &campaign=WEBSITE
  &number=1234
  &attrs=NAME:John,SURN:Doe
  &schedule=2024-03-18.14:00:00
```

This says "Call number 1234 on campaign WEBSITE, where we know that the person's name is John and their surname is Doe, after 2 PM today". Behind the scenes, their number is added to a list called `WEBSITE/AUTO` that Wombat manages for you, and will then do the recalls.

But what happens if they call us in the meantime, and we want to cancel the call? in this case, we have to know which blacklist to use - that is, one that is currently linked to the campaign WEBSITE. Let's say it was called DNC.

```
http://127.0.0.1:8080/wombat/api/lists/
  ?op=addToList
      &list=DNC
      &numbers=1234,REASON:recalled
```

This says "Add number 1234 to list DNC". As you know that list DNC is a blacklist that is set for the campaign WEBSITE, once you add the number, every number to be called on that campaign will be checked and skipped if necessary. While it's not mandatory, we suggest adding a `REASON` attribute to that number, so we know WHY this number was blacklisted and, possibly, by whom.

You could also do the opposite - what if there is a new case open for the same customer, and you need to remove this number from the list DNC? you could use the following API call:

```
http://127.0.0.1:8080/wombat/api/calls/
  ?op=cancelBlacklist
  &list=DNC
  &numbers=1234
  &reason=NEWCASE
```

This says "If there is number 1234 on the list DNC, mark it so that it is not used for blacklisting, and write the reason for de-blacklisting as NEWCASE". This is because it is important to keep track of why a number was added to a blacklist or removed from it.

When adding a new number to a blacklist, you could use any attribute to keep track of the reason - WombatDialer does not care, but it will make your life easier when auditing what went on. When removing, WombatDialer writes a special attribute `BLACKLISTED_CANCELLED` that contains the reason you gave it and the time the de-blacklisting happened.

Note that, to cancel a blacklisting, it must be canceled on **all** the blacklists linked to that campaign where the number might be present. If the number is present multiple times on the same list, all the instances where it is currently blacklisting (that is, where it was not already canceled or where its blacklisting period has already expired) will be changed to earmark them as canceled.

At this point, of course, you are free to add the number again if you need to cancel it again.

Administering WombatDialer

Installing

WombatDialer is easily installed using **yum** or it can be installed manually. It can be installed on the same server as Asterisk is (for small systems or testing) but on larger production systems we recommend that it should be installed on a dedicated system - a virtual machine will do perfectly.

WombatDialer can share a Tomcat instance with QueueMetrics for smaller systems, but nowadays it makes no sense to do so. Your life will be easier if you deploy two different and separate servers (that may be actually VMs on the same host), one for each product. If you install them within the same system, they will by default share the Java VM - so any issue with one (say, an unplanned restart) will bring down the other. Apart from test systems, just don't do it.

WombatDialer can also be easily deployed using Docker, as explained in Docker installation. This is perfect for installing a test system; but we have a number of sites running real-life workloads on it. If you use Docker, you can have QM and Wombat on the same server, as Docker will keep them separated.

WombatDialer offers a HTTP control API that is very simple, as to be used by the Asterisk dial plan, IoT and scripts; it is **not** meant to be exposed to unauthenticated access.

The best approach is to either protect access to the whole WombatDialer system via the firewall, or to put a proxy in front and filter all requests to `/api/...` URLs for whitelisted IPs.

It is also possible to add optional auth tokens to all API calls, as explained in Optional authentication. This can be combined with a firewall and a proxy for a thorough approach to system security.

Installing using yum

On a CentOS or derived Linux distribution, just run the following commands:

```
wget https://yum.loway.ch/loway.repo -O /etc/yum.repos.d/loway.repo

yum install wombat
```

This will in turn:

- install Java 8
- install Tomcat 8.5
- install MariaDB or MySQL server (if missing)
- install the dialer

After installation, you can access the filesystem for Wombat as `/usr/local/queuemetrics/wd-current`. To start the system and configure it further, see Starting and Stopping.

At the moment we support CentOS Stream 8, CentOS Stream 9, Rocky Linux 8 and Rocky Linux 9. If you run CentOS 5, 6 or 7, it is time to upgrade - if you really cannot, please see https://www.queuemetrics.com/faq.jsp?uid=faq-110-java8-centos5

Manual installation

You can install WombatDialer within any working servlet container version 2.5 or newer - we suggest Tomcat 8.5. You will also need a working MariaDB or MySQL server. OpenJDK 8 or 11 are fine.

- Download the latest version of WombatDialer as a .tar.gz package from the website
- Unpack the file in your servlet containers' webapps directory and rename the folder to "wombat"
- Restart your servlet container

Example: manual installation on Debian / Ubuntu

The following example was tested on an Ubuntu 22.04 system (LTS until 2027). All Debian-inspired systems should anyway be very similar.

First make sure you are running your shell as as `root`.

The main dependencies are OpenJDK and MariaDB. WombatDialer works with MySQL as well.

```
apt update
apt-get install tomcat9 mariadb-server
```

Then let's download WombatDialer as a TGZ archive:

```
cd /var/lib/tomcat9/webapps/
wget ..../WombatDialer-23.12.1-215.tar.gz
tar zxvf WombatDialer-23.12.1-215.tar.gz
mv wombatdialer-23.12.1/ wombat/
```

Now we install the initial database schema - we will create a database called "wombat" on the local DB, create a user to connect to it, and upload initial schema.

```
cd /var/lib/tomcat9/webapps/wombat/WEB-INF/DB
mysql < wombat_create.sql
mysql -uwombat -pdials wombat < wombat_schema.sql
```

Now we restart Tomcat so changes will be picked up:

```
service tomcat9 restart
```

Now connect to `http://127.0.0.1:8080/wombat` and follow the on-line wizard to create/update the database.

The default installers for both Tomcat 8/9 and MariaDB start their services automatically on boot.

On earlier versions (es 18.04 LTS, supported up to April 2023, EOL in 2028), you would use the package `tomcat8` instead of `tomcat9`, and its webapps are stored in `/var/lib/tomcat8` and not in `/var/lib/tomcat9`. For the rest, it just works.

On installation complete....

When installation is done, connect to `http://myserver.address:8080/wombat` - you will be asked for the MySQL root password to create a local database When done, login using as user **demoadmin** password **demo**.

WombatDialer ships with a free demo key that lets you use up to two lines - no limit on the number of calls you make. You may want to get a **free demo key** so you can run a full evaluation of the software in a real-life scenario.

License keys can be installed from the License page as soon as you log in.

Using an outbound proxy

WombatDialer requires outbound HTTPS internet connectivity to our licensing servers and cannot run without. It can use a direct connection or can go through a standard Java SOCKS proxy.

If a direct connection is not possible, then you have to set up a SOCKS 5 proxy (that we imagine being on address `1.2.3.4` at port `8888`). You can then tell WombatDialer to use it by adding the following items to the `JAVA_OPTS` start-up parameters:

```
-Dhttps.proxyHost=1.2.3.4 -Dhttps.proxyPort=8888
```

For more information on proxies supported by the JVM and how to configure them, refer to https://docs.oracle.com/javase/8/docs/technotes/guides/net/proxies.html

Checking connectivity

To check whether connectivity is working correctly, and to test the different Java parameters that you can set in JAVA_OPTS, you can use a script called `checkLicenseConnectivity.sh` that can be found under `WEB-INF/DB/`.

For example, to check connectivity with no proxy, you would simply run:

```
# ./checkLicenseConnectivity.sh
============================================================
```

```
  Attempting connection to the Loway licensing servers
============================================================

No proxy configuration settings
============================================================
   All went well.
============================================================
```

While to attempt using a SOCKS proxy, you could for example try:

```
# ./checkLicenseConnectivity.sh -DsocksProxyHost=127.0.0.1 -DsocksProxyPort=63080
============================================================
  Attempting connection to the Loway licensing servers
============================================================

Proxy configuration:

 |Property      |Value    |
-+--------------+---------|
0|socksProxyHost|127.0.0.1|
1|socksProxyPort|63080    |


Feb 04, 2022 9:21:44 AM it.loway.tpf.web.LowayHttpClient connect
SEVERE: Error accessing license server
java.net.SocketException: Connection refused (Connection refused)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:435)

============================================================
  Connection failed:
Error accessing HTTP URL
============================================================
```

Configuring e-mail

If you plan to use e-mail notifications, you should edit the `tpf.properties` file to tell WD about your SMTP servers and the e-mail address it is supposed to use.

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_AUTH=yes
SMTP_USER=your-gmail-account@gmail.com
SMTP_PASSWORD=wombat
SMTP_USE_SSL=no
SMTP_FROM="WombatDialer" <your-gmail-account@gmail.com>
SMTP_DEBUG=yes
```

As e-mail sending happens on the notification thread, a very slow SMTP server might delay HTTP

notifications. In this case, we suggest setting up a local SMTP relay that will immediately accept e-mail and will forward it to the "true" SMTP host.

We suggest leaving SMTP_DEBUG on during the initial setup as it writes a verbose log of all SMTP activity on the logs and lets you spot errors and issues quickly.

Installing on Docker

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system-level virtualization on Linux.

When you have Docker installed, starting a temporary instance of WombatDialer is as simple as typing:

```
docker run -p 8080:8080 -d loway/wombatdialer
```

This will download the current WombatDialer (already pre-configured for you) and will run it by exposing it on the local port 8080.

The problem with this approach is that when the image is terminated, all data is lost.

A better approach would be to create a data-only container to store the data. At this point, every time you start WombatDialer, it looks for its own data into the data-only container.

You start by creating a named data-only container:

```
docker run --name=MYWBT loway/data true
```

At this point, you can run WombatDialer using that container, as in:

```
docker run --volumes-from MYWBT -p 8080:8080  -d loway/wombatdialer
```

WombatDialer will recognize whether the data container is blank and will initialize it properly. The container also monitors the health of the running WombatDialer process and will restart it as needed.

If you prefer, you may use a local folder instead of a data-only container, as in:

```
docker run -v /root/WBT1DATA:/data -p 9090:8080 -d loway/wombatdialer
```

This command mounts the local folder `/root/WBT1DATA` for WombatDialer to store its data in, and - for a change - will expose WombatDialer on port 9090.

You can also control the local time-zone and the amount of memory used, by passing them as parameters when starting up the image:

```
docker run -e CFG='{"memory":400,"timezone":"GMT"}' -p 8080:8080 -d loway/wombatdialer
```

When you need to upgrade WombatDialer - you simply stop the current image and start a new one using the same data container. Your dialer is updated!

It is also possible to have a deeper control the JVM execution environment by overriding the default JVM variables; for example:

```
docker run --name WOMBAT
           -e CATALINA_OPTS="-Xms256M -Xmx512M
-Duser.timezone=GMT+01:00"
           -e TZ=Europe/Lisbon
           -p 8088:8080
           -P -d loway/wombatdialer
```

What this means is that:

- the `CATALINA_OPTS` setting will affect the Java machine memory and timezone, so you can fine-tune the JVM as you best see fit
- the `TZ` setting will affect the container's own timezone

Upgrading

Upgrading WombatDialer is meant to be very easy - it is in any case worthwhile to make a complete backup (see below) before attempting an upgrade. If you run WombatDialer on a virtual machine, the easiest way might be taking a snapshot of the whole box before attempting the upgrade.

Before attempting an upgrade, always make sure that no calls are in progress and that the dialer is turned OFF. If you have running campaigns, you should pause them and wait for relevant calls to terminate.

If you update from a version of WombatDialer before 18 to version 18+, the database driver was changed. Look for your `tpf.properties` file and replace the entry:

```
JDBC_DRIVER=com.mysql.jdbc.Driver
JDBC_URL=jdbc:mysql://127.0.0.1/wombat?user=wombat&password=dials&useUnicode=true&char
acterEncoding=UTF-8
```

With:

```
JDBC_DRIVER=org.mariadb.jdbc.Driver
JDBC_URL=jdbc:mariadb://127.0.0.1/wombat?user=wombat&password=dials&autoReconnect=true
```

And restart.

Upgrading via yum

If you originally installed via yum, this should be enough:

```
yum update wombat
```

System settings should be automatically transferred.

When the update is complete, on the first connection you might be asked to upgrade the database - the dialer will not start until the database is up to date. Database upgrades will be applied automatically.

If after an upgrade you get an error like `java.util.MissingResourceException: Can't find bundle for base name org.eclipse.jdt.internal.compiler.problem.messages`, this means that there are cached classes created with a different version of the JDK.

You can have them flushed by issuing the commands:

```
service qm-tomcat6 cleancache
service qm-tomcat6 start
```

Manual upgrade

- Stop the servlet container
- Download the latest version of WombatDialer as a .tar.gz package from the website
- Move the current "wombat" webapp to a storage place (do not delete it!)
- Unpack the file in your servlet containers' webapps directory and rename the folder to "wombat"
- Copy the `tpf.properties` file from the old wombat webapp to wombat/WEB-INF
- Restart your servlet container

When the update is complete, on the first connection you might be asked to upgrade the database - the dialer will not start until the database is up to date. Database upgrades will be applied automatically.

Starting and stopping

If you installed using `yum`, you should be able to run:

```
/etc/init.d/qm-tomcat6 stop
/etc/init.d/qm-tomcat6 start
```

to stop and start the Tomcat servlet container and thence WombatDialer.

If you installed manually, see the user manual for your servlet container.

Before stopping, always make sure that no calls are in progress and that the dialer is turned OFF. If you have running campaigns, you should pause them and wait for all relevant calls to terminate. If not, those calls will end up in status LOST.

Additional environment configuration

It is possible to add additional Java environment configuration options to specify system-wide behavior. Those are generally added to the `JAVA_OPTS` environment variable - eg to specify an HTTP proxy, to force a specific time-zone, to enable JMX monitoring, a garbage collection policy or a number of other features.

WombatDialer has also a few parameters of its own:

- `wd.auth.tokens`: sets a securty token to protect API calls from unauthorized access, see Optional authentication. By default it is unset.
- `wd.core.pause`: after each iteration, Wombat may need a small pause to let the database commit. On some systems, a value too low may cause errors with the message "Batch update returned unexpected row count". Default is 200ms; you may want to tweak it higher or lower.

On a default RPM installation of WombatDialer, you can set these properties by editing the file `/etc/sysconfig/qm-tomcat6`, by adding them to the end of the `JAVA_MEM` property so that instead of being e.g. `JAVA_MEM="-Xms256M -Xmx768M"` it becomes `JAVA_MEM="-Xms256M -Xmx768M -Dwd.auth.tokens=3db6e6b81bfe8a,e51979ce575e949dc"`.

The dialer status at startup

The dialer will usually start automatically when the webapp is loaded. Exceptions to this behavior happen if:

- WombatDialer was updated and the database required updating. In this case the dialer will be turned off until manually activated.
- There is no valid license installed.

If you prefer to avoid the dialer starting automatically, you can do this by creating a flag file called *'donotstart'* to be placed under WEB-INF, like e.g.

```
touch ./WEB-INF/donotstart
```

In order to revert to the previous behavior, simply remove the flag file.

Backing up

All the information WombatDialer requires to run is contained within the `wombat` MySQL database.

Backing it up should suffice in most cases.

If you use a different database that the default MySQL on localhost, then you need to back-up the file `wombat/WEB-INF/tpf.properties` that contains the database connection information.

For ease of recovery, you may also want to back up the whole "wombat" webapp.

Logs and logging

User-serviceable logs are available on the syslog viewer page. This tracks errors, agent logins, campaign activity and many common user-facing conditions.

A log of severe errors is also held in the Tomcat logs directory, usually with file names of the format *wombat.2013-01-05.txt* or similar.

This behavior can be customized by editing the file `WEB-INF/classes/logging.properties` in the WombatDialer webapp.

```
org.apache.juli.FileHandler.level = SEVERE
org.apache.juli.FileHandler.directory = ${catalina.base}/logs
org.apache.juli.FileHandler.prefix = wombat.

java.util.logging.ConsoleHandler.level = SEVERE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

You can turn off logging by setting the logging level to NONE, or create extra-detailed and huge logs by setting the log level to FINE.

An extensive tutorial on Tomcat logging can be found at http://wiki.apache.org/tomcat/Logging_Tutorial

Please note that running very detailed logs on a production system may create huge files and may render the system unusable by generating too many I/O requests.

Monitoring a running instance

WombatDialer exposes a large set of metrics through the JMX interface of Java; such metrics can be inspected, downloaded and monitored by remote processes, or sent to a database for further analysis.

A detailed reference description of each counter is available in JMX statistics API, as well as a webservice to download all these data at once as a JSON object.

Connecting via VisualVM

One of the nicest things of the Java environment, the one that sits behind QueueMetrics and

WombatDialer, is that it offers a very accurate and detailed view of what is actually going on. You do not need to use a special build of the software that is meant to gather statistics, and you do not have to pay a high performance price for understanding what the system is doing, so that it is perfectly acceptable to gather live statistics on a production environment. Even better, JVMs are built for being monitored, so that there is a standardized ecosystem of tools to help you understand what is actually going on. This approach is called JMX.

VisualVM is a free graphic tool that can be used to connect to a running Java VM in order to understand what it is doing. Apart for checking memory, CPU usage and garbage collections, it can be used to gather accurate information on what WD is doing internally, as WD exposes its own telemetry through JMX.

To monitor WD running on a server from a workstation, the easiest thing is to set up a SSH tunnel. You will need a recent Tomcat RPM (8.5.64 or newer).

So, once it is installed, you edit `/etc/sysconfig/qm-tomcat6`, and where it says:

```
# Remove the comments below to activate JMX monitoring
# JAVA_RMI_ADDRESS=localhost
JAVA_JMX_PORT="30000"
```

You uncomment the line with `JAVA_RMI_ADDRESS`; save the file and restart. You just need to do this once, and this can be done safely even on production systems. You can change the JMX port if you need to - does not matter which port you choose, as long as it is the same on your PC and on the server.

Now, you go to your workstation and start a new SSH session with the following incantation, to create a local tunnel for port 30000 and the following one too:

```
ssh root@my.wombat.server \
    -L 30000:localhost:30000 \
    -L 30001:localhost:30001
```

You can now run VisualVM on your workstation - most likely it will already be installed if you have a Java SDK; if not, you can download it from: https://visualvm.github.io

Note that while the example above logs in over SSH as *root*, any user will do.

When you launch VisualVM, you should now click on the button *Add JMX connection*, and create a new connection to `localhost:30000`, with no password and no encryption (as we already use the SSH tunnel to guarantee security and encryption).

Peeking around: CPU and memory

Apart from the welcome page, the first things you should check are memory and CPU.

For example, in the graphs above, we can see that:

- CPU usage is around 30%, and is stable (graph on the top left). Very little garbage collections (the blue lines) - they are usually the main culprits for Java performance problems.
- We have no memory pressure. The system hovers around 2G of usage (graph on the top right) and has more memory than it actually needs.
- The system was kind of idle in the last few minutes - it did not allocate much memory (look at the memory graph), but it is computing something, as the CPU is still running.
- We can know **what** it is doing by getting a thread dump.

WombatDialer is not especially memory-hungry, but might create a lot of intermediate objects that may cause memory pressure; of course, if you do not have enough memory, you will have a lot of garbage collections that will freeze the system and will severely impact performance.

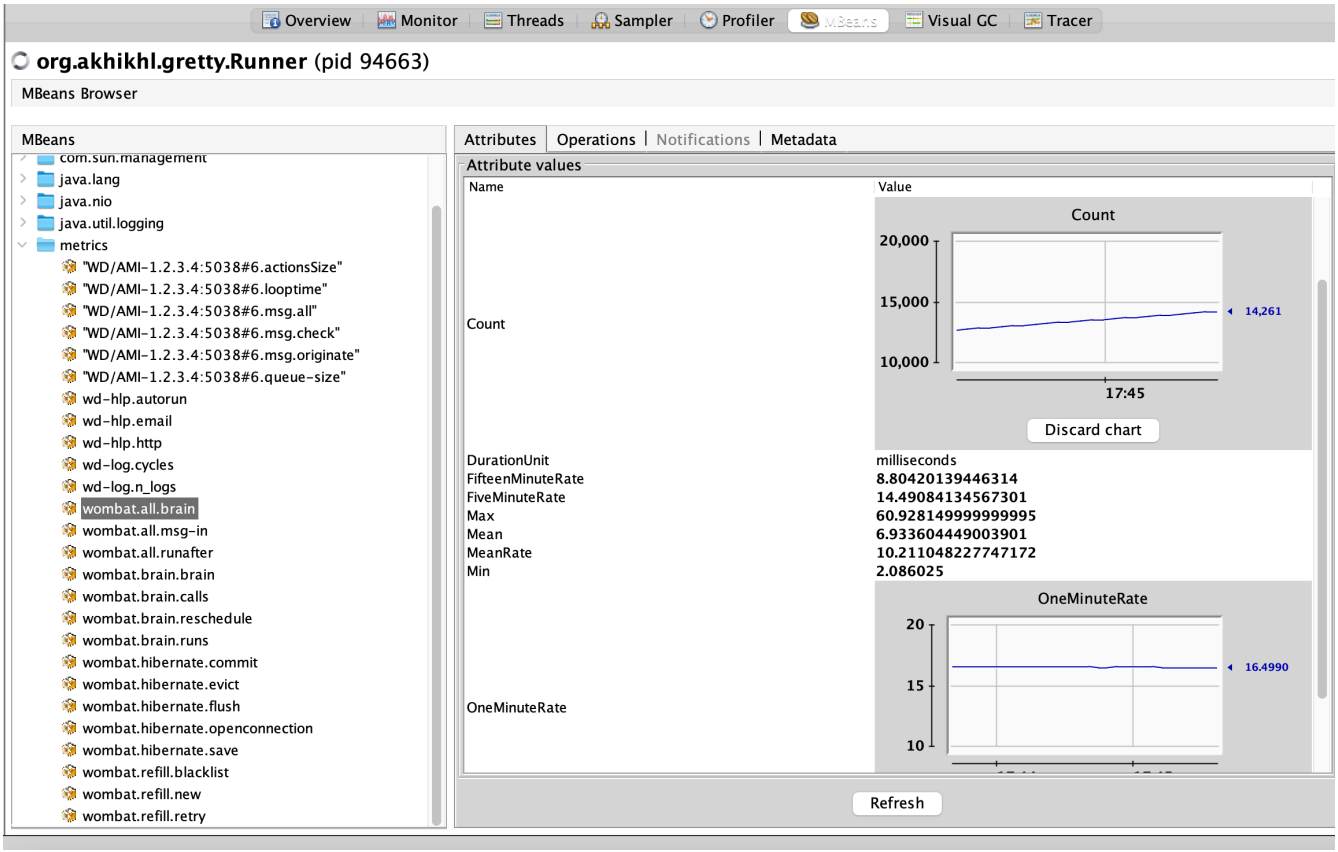
A number of interesting things can be done with VisualVM:

- *Know your JVM*: you can see the JVM settings from *Overview* | *JVM arguments*.
- *Memory monitoring*: you can see the current CPU, memory and thread usage from the *Monitor* page. Note that with most settings, it is normal that all memory be used up before a garbage collection is performed; so you would expect to see spikes and falls in the graph. You can also force a garbage collection if you want to see the "true" memory usage, but this may be unwise on heavily loaded production servers.
- *Thread monitoring*: you can get a textual thread dump like the one discussed above by selecting *Threads* | *Thread dump*
- You can use the *Sampler* to acquire a breakdown of memory and CPU usage per class (you first need to install the plugin *VisualVM-Sampler* from the Plugins menu, if not already available)
- You can keep a server open with multiple instances of VisualVM in order to monitor multiple servers at once and compare what is happening on each of them.

Understanding WombatDialer's telemetry

To access telemetry, you click on MBeans, and open the items under "Metrics", as displayed below. Each metric has a name and offers one or more attributes. When you find an interesting attribute, you just click on any number to turn it into a graph that follows the value across time, as shown

below.



Most things that WD exposes in its telemetry are loops; so they usually offer:

- a **Count**, that shows how many times the loop was traversed. Often, a loop runs when there is an incoming message to be processed
- a value, that is usually shown in terms of **Min**, **Mean** and **Max**, and also in terms of **percentiles** (median, 95%, 99%).
- a **Rate**, that is how many times per second the loop was triggered, in the last 1, 5 and 15 minutes

When inspecting a system, you should start from the core metrics, that is the brain, and from metrics of connected Asterisk servers. The brain should be running smoothly, and at least 4/5 times per second, and the command queue on each Asterisk server should be empty or nearly-empty.

Core Metrics

On each iteration, WD receives a set of messages from the GUI, your API and from any connected PBX servers. It uses those messages to update its state, then decides what to do, and every once in a while runs clean-up tasks. At the end of each loop, the database is synchronized (with a commit and/or an eviction and/or a refresh).

- `wombat.all.msg-in` is the time spent processing the incoming mailbox. On an healthy system, each loop should take just a few milliseconds. When processing API requests, it might spike up. If a loop is taking more than 1 second, messages are processed in a further iteration; this should never happen, or very rarely. Look at the high percentiles to see if this happens to you.
- `wombat.all.brain` tracks how long each brain iteration takes. They should be taking a few milliseconds each. Detail on what is happening within is shown below.

- `wombat.all.runafter` is the task queue. Most times it should complete in zero time, i.e., the queue is empty.

Brain time is further divided into:

- `wombat.brain.brain` is the actual decision time
- `wombat.brain.runs` is the time spent updating the campaign (e.g. refilling new calls)
- `wombat.brain.calls` is the time spent scheduling calls (included the second check against blacklists)
- `wombat.brain.reschedule` is the time to close terminated calls, and apply reschedule/disposition rules

After that, timed commits happen and logs are shipped for writing to a separate helper process.

When starting new campaign runs, or just running normally, numbers to be dialed have to be loaded from the database:

- `wombat.refill.new` is the time spent fetching new numbers to dial
- `wombat.refill.retry` is the time spent fetching retries
- `wombat.refill.blacklist` is the time spent checking calls against blacklists. Calls are checked against black-lists twice: when loaded and just before being attempted.

Database metrics

- `wombat.hibernate.openconnection` is the time and number of connections open by the brain. As connections are persistent, it should happen just once, or a few times - otherwise it means the dialer keeps restarting or the database disconnects. This is a major signal of distress.
- `wombat.hibernate.save`, `wombat.hibernate.flush` and `wombat.hibernate.commit` happen when Wombat tries to write to the database. Theey might take a while - check high percentiles of duration.
- `wombat.hibernate.evict` is called to remove old objects
- `wombat.hibernate.refresh` does a deep reload of objects

Look out for very high numbers, and then check what is happening on your database. Does it have enough CPU and IOPS to run smoothly? Is it very busy, or is it sitting idle?

Asterisk metrics

Each Asterisk head has separate statistics and is displayed with its own name.

- `WD/AMI-xxx.commandQueueSize` is the number of messages that are ready to be sent to the PBX, but cannot be delivered because the messages-per-second limit is hit, or because the PBX is processing them too slowly. Ideally should always be zero - a long queue prevents WD from working correctly. If this is consistently high, you need to increase the AMI message limit - but beware that a PBX flooded with messages is prone to sudden crashes, so run your experiments before changing it in production.

- `WD/AMI-xxx.isConnected` is 1 when the server is connected and 0 otherwise.
- `WD/AMI-xxx.fromAsterisk` is the time taken processing messages from Asterisk.
- `WD/AMI-xxx.fromMaster` is the time spent converting actions from WD to Asterisk.
- `WD/AMI-xxx.sendSocket` is the time spent sending messages on the socket.

Other metrics

The Helper offloads some tasks from the main dialer:

- `wd-hlp.autorun` is the time spent running tasks (HTTP, SMTP, database updates)
- `wd-hlp.email` is the time spent sending email
- `wd-hlp.http` is the time spent in HTTP notifications

Logger:

- `wd-log.cycles` is the time spent writing call logs. Logs are written in batches, where possible.
- `wd-log.n_logs` is the number of log entries written

Troubleshooting a running instance

Problems with a WD instance are usually either on the Java side or on the database side. We present a few useful techniques to gather important data to be sent over for inspection.

Creating a thread dump

In order to diagnose problems with WombatDialer, it may be needed to take thread dumps by running:

```
/etc/init.d/qm-tomcat6 threaddump > dump_2013-01-18.txt
```

These dumps prints what the threads are doing at a point in time. It may be worthwhile to take multiple ones, a few minutes apart from each other.

Creating a TCP dump

In order to diagnose problems with WombatDialer, it may be needed to take TCP dumps:

First off you can check your network interfaces by typing:

```
tcpdump -D
```

the result will be something like

```
1.eth0
```

```
2.any (Pseudo-device that captures on all interfaces)
3.lo
```

Issue the dump on port 5038 (remember to add the -s option to avoid capturing only packet's headers)

```
tcpdump -i lo -s 65535 -w dump.pcap port 5038
```

This will generate a file called **dump.pcap** which can be analyzed using Wireshark.

Creating a heap dump

On request, you may need to create a heap dump. This is a large file containing the full contents of the live JVM.

```
[root@wombat1 ~]# /usr/local/queuemetrics/java/bin/jps
1900 Jps
30808 Bootstrap
```

The line containing `Bootstrap` contains the PID of the Java VM running Tomcat.

```
[root@wombat1 ~]# /usr/local/queuemetrics/java/bin/jmap -F
-dump:live,format=b,file=heap.bin 30808
Attaching to process ID 30808, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 17.1-b03
Dumping heap to heap.bin ...
Finding object size using Printezis bits and skipping over...
Finding object size using Printezis bits and skipping over...
Finding object size using Printezis bits and skipping over...
Finding object size using Printezis bits and skipping over...
Finding object size using Printezis bits and skipping over...
Heap dump file created
```

The resulting file (called `heap.bin` as specified on the command line) might be very large and it might take a while to create on busy systems with large Java heaps.

Before sending the file over it is better to compress it:

```
[root@wombat1 ~]# ls -l heap.bin
-rw-r--r-- 1 root root 40779016 Jan 10 08:45 heap.bin
[root@wombat1 ~]# bzip2 heap.bin
[root@wombat1 ~]# ls -l heap.bin.bz2
-rw-r--r-- 1 root root 8055474 Jan 10 08:45 heap.bin.bz2
```

Logging all MySQL queries

Edit the `my.cnf` file (usually in `/etc/mysql/my.cnf`). Look for the section called "Logging and Replication".

```
#
# * Logging and Replication
#
# Both location gets rotated by the cron-job.
# Be aware that this log type is a performance killer.

log = /var/log/mysql-all-queries.log
```

Just uncomment the "log" entry to turn on logging. Restart MySQL with this command:

```
service mysql restart
```

Logging of "slow queries" in MySQL

Edit file `my.cnf` and add the following parameters:

```
[mysqld]
log-slow-queries=/var/log/mysql-slow-queries.log
long_query_time = 2
log-queries-not-using-indexes
```

This will log all queries taking more than two seconds or running without a defined index to the slow queries log.

Then restart MySQL.

Database Clean-up Script

In order to manage WombatDialer's database size, a clean-up script is included in WombatDialer's installation package. The script file can be found in the application folder at the following path:

```
WEB-INF/DB/cleanup_wbt.sh
```

Before you run this script make sure you give it the appropriate permissions it needs to be executed.

This script, when executed, searches for old data that is no longer useful or has been hidden by the user. To decide how old must this data be in order for it to be deleted, the user must input a date that will be considered its clean-up threshold.

There are two areas where data can be deleted:

- **Campaign runs that are hidden, or happened before the given threshold date.** This means that the script will search for all runs of campaigns that are currently hidden that happened before the input date. The runs will be removed; if the campaign has no other runs, then it will be removed as well. If a campaign is deleted, all logs associated with the campaign will be deleted as well. All System logs records before the given date are erased as well.
- **Unused lists.** If a list is hidden, and is used by no campaign or runs, then the list and all numbers and attributes are removed. If you run this after cleaning old campaigns, lists used by those campaigns will be removed.

When all the data has been erased, the script will initiate a procedure to rebuild all tables and indexes, thus optimizing and improving database access times.

Please note that in order to successfully run this script, WombatDialer must be turned off before running it. On a multi-gigabyte database, plan a few hours of possible downtime.

Running this script will cause some data to be **deleted permanently**. It is possible for the script to leave the database in an inconsistent state. *It is mandatory that you make a backup copy* of your database before running this script.

Running the clean-up script

After navigating to the correct folder and setting the appropriate permissions to the script file, you can execute it by typing:

```
./cleanup_wbt.sh -u wombat -p dials -d wombat -t 2017-12-31 -m all
```

As you can see, we need to input five parameters to execute the script. These parameters are all mandatory except for one:

- *-u*: MySQL user parameter (Mandatory)
- *-p*: MySQL password parameter (Mandatory)
- *-d*: MySQL WombatDialer database name (Mandatory)
- *-t*: Date threshold to identify old data (yyyy-mm-dd). Data more recent than this date will not be considered for elimination. (Mandatory)
- *-m*: Mode selection. Available modes are: `list`, `delete`, `optimize`, `preview`, `all`. The default mode is `all`.

There are five different modes for the script:

- *list*: Does not delete any data, only shows hopper elements that would be removed in a normal execution.
- *preview*: Does not delete any data, only shows hopper elements and call lists that would be removed in a normal execution.
- *delete*: Deletes all old data, without table optimization.

- *optimize*: Optimizes all the tables.
- *all*: Default mode. Executes all the passages in order.

We suggest manually running the script as `preview` first; then `delete` and then `optimize`.

API reference

This chapter documents the different WombatDialer APIs available. As a reference point, we have a public repository with some examples and integration code that leverages APIs at https://github.com/Loway/OpenWombatDialerAddOns where you can find scripts and examples. We welcome contributions!

Controlling Asterisk from WD

When a call is originated on Asterisk, a set of channel variables are set:

- `WOMBAT_HOPPER_ID` is a unique identifier of the call being processed (on old versions, it used to appear as `WOMBAT_ID`)
- `WOMBAT_RETRY_NO` is the number of the current recall
- `WOMBAT_DIALING_NUMBER` is the main number being called (that may be different from the actual one dialed when using multinumbers)
- `WOMBAT_DIALING_LIST` is the name of the list that this number comes from
- all **input** attributes are passed to the call

This way, all attributes you set for the call are immediately available within the Asterisk dialplan.

For example, if you define input attributes called **HH** and **MM** in WD, you can use them in the dialplan to synthesize an audio message as in the following example:

```
exten => s,n(start),agi(googletts.agi,"Your appointment with the dentist
                is for tomorrow at ${HH} ${MM}")
```

If you do a `DumpChan()` of the channel that Wombat triggered, you will see something similar to the dump below:

```
Dumping Info For Channel: Local/5551234@wdtk-00000006;1:
============================================================
Info:
Name=               Local/5551234@wdtk-00000006;1
Type=               Local
UniqueID=           1700210157.12
LinkedID=           1700210157.12
...

Variables:
WOMBAT_HOPPER_ID=1555774267
WOMBAT_DIALING_LIST=CUSTOMERS2307A
WOMBAT_RETRY_NO=0
WOMBAT_DIALING_NUMBER=5551234
NAME=John
SURNAME=Doe
```

```
HH=10
MM=30
MULTINUM1=5553456
============================================================
```

Controlling WD from Asterisk

WD lets you pass information about a call that is currently processed in Asterisk in the form of **User Events**. This lets you set the completion code for the call and any additional parameters, e.g. IVR responses, right from the Asterisk dialplan.

UserEvents can only be set on calls that are connected; if sent before the call is up, they will be ignored.

Format of User Events

User events are triggered through the dialplan through the **UserEvent** Asterisk application.

They are used to:

- Set the completion code for a given call
- Store additional outbound attributes on the call in WD

Triggering a user event

You can easily trigger User Events by calling the Asterisk dialplan like:

```
exten => s,n,UserEvent(name,Uniqueid:${UNIQUEID},P0:parm1)
```

or

```
exten => s,n,UserEvent(name,Uniqueid:${UNIQUEID},P0:parm1|P1:parm1)
```

In case you want to pass two parameters.

The event name is not case sensitive.

The `Uniqueid` field must match the ID that is used for the main leg of the call; so in case you are dialing in reverse mode, you should use:

```
exten => s,n,UserEvent(name,Uniqueid:${CHANNEL(LINKEDID)},P0:parm1)
```

To make sure you are using the right identifier.

You can see the "canonical" Uniqueid on a WombatDialer call by going to the "Call

> Log Record" of your call and checking under "Technical details". You can also see it on live Asterisk calls by using the CLI command `core show channel ...` when the call is up.

Setting call status codes

In order to set the extended status (completion code) of a call, you should trigger an UserEvent like:

```
exten => s,n,UserEvent(CALLSTATUS,Uniqueid:${UNIQUEID},V:XXX)
```

This sets the completion code for the current call to XXX. If multiple completion codes are received for the same call, the last one wins. Completion codes are then used by WD's Reschedule Rules in order to decide what to do with the call.

Output call variables

You can set an output attribute for the current call by running:

```
exten => s,n,UserEvent(ATTRIBUTE,Uniqueid:${UNIQUEID},name:value)
```

This will create an attribute called "name" with value "value". If the same attribute already exists, it is replaced.

As an alternative, you can append values to an existing attribute, like:

```
exten => s,n,UserEvent(ATTR_APPEND,Uniqueid:${UNIQUEID},name:value)
```

This will append the value "value" to the current value of attribute "name"; if the attribute is not present, it will be created with value "value". This makes it easy to to track the traversal of IVR trees.

You may also want to clear all output attributes (but of course not input ones) of a call by issuing:

```
exten => s,n,UserEvent(ATTR_CLEAR,Uniqueid:${UNIQUEID})
```

Please note that:

- Attributes are *per number*; so if you call the same number multiple times, its attributes will be the last ones used.
- All attributes are text strings
- A number may have an unlimited number of attributes

It is possible to set status and attributes from the HTTP interface as well.

Controlling WD over HTTP

WD is built so that it can be controlled externally by an external script or dial-plan function. The protocol is very simple and - by default - not authenticated (but see below), so you must make sure that it is not reachable from unsafe parties. See Installing Wombat.

In the following examples, we assume that WD is running at address `http://127.0.0.1:8080/wombat`

It is important to notice that the /' before the question mark in the following examples is mandatory - if not present you will get an HTTP `error 404`.

Though the examples below display HTTP GET requests, the APIs work as well with HTTP POST calls.

If you use Tomcat 8+ as a servlet container then it will return an error 400 if you do not accurately URL-encode all characters according to RFC 7230 and RFC 3986. This often happens when passing a set of elements that are separated by the pipe symbol - so for example `a|b` will not work, but `a%7Cb` will. For readability's sake, the examples below still show the pipe character, and it's up to you encoding your inputs correctly.

Optional authentication

While it is not so by default, you can add an optional layer of security that will prevent access to basic API calls unless you hold one of the approved tokens. These tokens are specified system-wide through a JVM configuration property (see Additional environment configuration), and you can set one or more of them by separating them with a comma, as shown below:

```
wd.auth.tokens=3db6e6b81bfe8a,e51979ce575e949dc
```

These tokens are case-sensitive, and we suggest using random text sequences, as you best see fit in terms of length.

When the JVM property is set, all API calls below require an extra parameter `auth_token_id` that will be checked to see if it matches any of the tokens set. If none of them matches, then the request is denied.

For simplicity's sake, all examples in this chapter are displayed without the extra auth parameter; but if you have a call like:

```
http://127.0.0.1:8080/wombat/api/calls/?op=addcall&campaign=ABC&number=45678
```

And you enabled auth tokens, you need to add your token to the request, like:

```
http://127.0.0.1:8080/wombat/api/calls/?op=addcall&campaign=ABC&number=45678&auth_toke
```

```
n_id=3db6e6b81bfe8a
```

Engine control

The dialer can be started and stopped by the API, and its current state can be retrieved.

Retrieving current engine state

```
   http://127.0.0.1:8080/wombat/api/engine/?op=STATE
```

Will produce a JSON output like:

```
{
  "state" : "READY",
  "uptimeMs" : 5809,
  "msgSent" : 23,
  "msgReceived" : 108,
  "maxLicensedChannels" : 20
}
```

This shows:

- The current state.
- The current up-time for the dialer
- The number of messages sent and received
- The maximum number of licensed channels

Valid states are:

- `READY`: Wombat is up and running.
- `DOWN`: The dialer is currently switched off.
- `TERMINATION_REQD`: The dialer was up and a message was queued to ask it to shut down. This may take a few seconds; you can check its state by sending a new STATE call.
- `TIMEOUT_NO_REPLY`: The dialer did not answer within 10 seconds; it is either extremely overloaded, or it was shut down in the meantime.

Uptime, number of messages and licensing information will be zeroed out if the state is not `READY`.

Start and stop the engine

To **start** the dialer's engine you call:

```
http://127.0.0.1:8080/wombat/api/engine/?op=START
```

This call returns the current state. If you call this on an engine that is already started, nothing happens.

To **stop** the dialer, you call:

```
http://127.0.0.1:8080/wombat/api/engine/?op=STOP
```

The reply is always a message of type `TERMINATION_REQD`, and you will have to check the state again in a few seconds. Note that this is a graceful termination; the dialer is requested to stop, so it will do it once the current set of messages is processed.

Once the engine is off, any pending call will remain pending, and it will not be tracked any longer, no matter what happens on the PBX. So, before terminating the engine, you should make sure that all runs are paused or terminated, and that no calls are ongoing. See below for the relevant APIs.

Calls

Add a call to a campaign

This methods lets you add a call to a campaign that has a current run (so it may be active, idling, waiting for the right time, etc).

If the campaign is idling, it will be "woken up" automatically when new numbers are added through this call. Numbers will be added to its Automatic list that is created ion demand and has a different behavior from normal lists.

Numbers added this way have a higher priority that numbers coming from current lists that the campaign is going through (if any). So they will be handled without waiting for any other previous lists to complete first.

```
http://127.0.0.1:8080/wombat/api/calls/?op=addcall&campaign=ABC&number=45678
```

You can queue calls for the future by adding a reschedule attribute, either as an offset in seconds (positive integer) or an absolute date having the format `YYYY-MM-DD.HH:MM:SS`.

```
schedule=15
schedule=2012-03-18.00:00:00
```

You can also set multiple attributes, as in:

```
attrs=ID:2301,PERS:John_Doe
```

If you add the same number multiple times, then it will be dialed multiple times.

Current versions of WombatDialer by default apply backpressure, that is, will not just enqueue a number for being added but will expect a confirmation that the operation was performed. As the number is actually handled by the main WombatDialer process, this means that it is scheduled together with the rest of current activities, and therefore the call might require tens or hundreds of milliseconds to be completed. On the other hand, this prevents you from uploading numbers faster than WD can handle them.

If you are sending numbers from a time-sensitive process (e.g. the Asterisk dial-plan) you may want to add the parameter `fireforget=1` so that the call will return immediately. This was the default behavior up to WD 20.04. The downside is that if you push hundreds or thousands of numbers at once, they might require a lot of time to be fully processed, and this risks disrupting current calls or making the dialer unresponsive.

It is important to understand that all calls added this way are technically reschedules, even if no `schedule` attribute is set. Still, the first attempt will be numbered *0*, in order to keep a consistent approach when using multi-numbers.

This method is meant to drip-feed numbers to an existing run, as needed. If you need to upload hundreds or thousands of numbers at once, what you want to do is to upload a list of numbers at once and then add it to a campaign - this can be achieved though the Configuration API and/or the `addToList` method and has better performance.

Set extStatus and attributes on a live call

While a call is being dialed, it is possible to set a user event externally in a manner that exactly matches normal UserEvents. The important difference is that you do not need the UniqueID but the WombatID.

Set a call's extStatus to XY:

```
http://127.0.0.1:8080/wombat/api/calls/?op=extstatus&wombatid=123456&status=XY
```

Create or set attribute AB to value CD:

```
http://127.0.0.1:8080/wombat/api/calls/?op=attr&wombatid=123456&attr=AB&val=CD
```

Append string EF to the current value of attribute AB:

```
http://127.0.0.1:8080/wombat/api/calls/?op=attrAppend&wombatid=123456&attr=AB&val=EF
```

Remove all outbound attributes for this call:

```
http://127.0.0.1:8080/wombat/api/calls/?op=attrClear&wombatid=123456
```

This API works as well for closed calls; so you can set extStatus and variables on calls that are already terminated. The most important limitation when doing so is that any Reschedule or Disposition Rules that would have been triggered by if the call had had the correct extStatus on termination are NOT triggered.

Campaigns and runs

Start a new run with an existing campaign

This call starts a new run an existing campaign; if the campaign is not startable (e.g. because it is already running) the call is ignored.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=start&campaign=ABC
```

Pause a running campaign

This call pauses a campaign that is running or idling; if no run is found in a suitable state the call is ignored.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=pause&campaign=ABC
```

Unpause a running campaign

This call unpauses a run that is currently paused, when unpaused it may start dialing or keep on idling or start waiting for a correct time to place calls.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=unpause&campaign=ABC
```

Remove a run

This call removes an existing run; the run is supposed to be paused and you should either wait an appropriate amount of time or you should make sure no calls are active (if not, they will be forcibly closed in Wombat and will end up in state `ERROR`, while they will not be tracked anymore if they are active on the PBX).

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=remove&campaign=ABC
```

Reload a campaign

This call asks for a campaign to be reloaded, i.e. its definition to be read from the one that is currently saved in the database and not the one that was used when the campaign started.

This lets you pick up any changes you made to the campaign configuration on a running campaign.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=reload&campaign=ABC
```

Reload does not pick up an updated definition for Asterisk servers, Trunks and EPs, as these objects are possibly shared across all campaigns. Still, it is possible to "change" an existing object, say an EP, by the following rules:

- Create a new EP through the API
- Edit a campaign; remove the old EP and add the one you just created
- Reload the campaign

At this point, the new EP will start being used, while calls made on the old EP will still run until they terminate.

Clone a campaign

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=clone&campaign=xx&newcampaign=yy
```

This will clone existing campaign named ABC to a new one named DEF. It will clone only the campaign definition, so no activity statistics - just like if you were to create a new one from the GUI. After cloning, you'll have to start the campaign run.

Add an existing list to a campaign

This call lets you add an existing list to an existing campaign (usually on a cloned one). It will not work on running campaigns as they will not pick-up the new configuration until restarted.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=addList&campaign=QMS&list=ZEB
```

Add an existing list to a campaign as a black-list

This call lets you add an existing list as a black-list to an existing campaign (usually on a cloned one). It will not work on running campaigns as they will not pick-up the new configuration until restarted.

```
http://127.0.0.1:8080/wombat/api/campaigns/?op=addBlackList&campaign=QMS&list=ZEB
```

Notify the state for an API-driven end-point

This feature is experimental and subject to possible changes.

This call lets you feed information to WD for an API-driven end-point.

```
http://127.0.0.1:8080/wombat/api/endpoint/?op=update&queue=z1&n_idle=20
```

Parameters:

- `queue` is the name of the EndPoint - it must be of type API, otherwise it will not be updated. On the other hand, multiple EPs of type API can share the same `queue` and will be updated at once.
- `idle`, `ringing`, `talking`, `paused`: a comma-separated list of agents that are in each state. If no parameter passed, or left empty, each set of agents is considered to be empty.
- `calls`: a list of calls that are present on the queue waiting to be connected. Calls that are already connected should not appear here (but their agents should appear under `talking`). You can use any Ids you like.
- `n_idle`, `n_ringing`, `n_talking`, `n_paused`, `n_calls`: when the exact agent/call codes are not important (e.g. for direct dialling) you can use one of these parameters to have a list of entries created on the fly.

Agent codes should be "real" channel definitions for reverse or reverse-preview modes, at least for `idle` agents. When called on the Asterisk server, they should be able to ring the relevant phone on your destination system.

In all other cases (and for call lists), it is up to you whether you want to specify the exact agent and call codes, or you just want to update the system by sending in an aggregate. Internally, the aggregate will be expanded into a pro-forma list of items, for example `n_idle=2` will expand to `idle=idle/1,idle/2`.

This call is synchronous, it will return only after the update has been picked up by the dialer, as to help enforce backpressure.

This service should be called every time there is an update, at least with the list of free agents. It is okay to batch multiple calls together if they happen within 100-300ms from each other. You can experiment with lower update rates (e.g. every second, or every few seconds) but the update rate should be significantly shorter than the average call set-up time.

When sending messages very quickly, do monitor that incoming messages are processed as they come - you don't want to build up a backlog. See Monitoring a running instance.

Query current calls and recalls for runs

This call lets you query the state of ready and deferred calls for a given run or set of runs. Calls scheduled for the future are split into "reschedulingSoon", that is calls that are either immediately reschedulable or will become so within a short time limit (by default 5 minutes) and "reschedulingLater". The limit on which they are to be separated can be set by the user.

```
http://127.0.0.1:8080/wombat/api/recallinfo/
```

The method accepts the following parameters:

- *campaign* is a substring to be searched in a campaign's name. If missing, all runs will be shown.
- *late_reschedule* is the time limit - in seconds - that separates "soon" to be rescheduled calls from "late" reschedules. Default is 300.

The results shows all calls ready to be run (as they are in the hopper) and all live calls, split by status; plus all recalls scheduled for the run.

The output looks like the following one:

```
{
  "runs" : [ {
    "campaignName" : "ABC-3.15-fixed",
    "runName" : "15.03.25 16:18:36",
    "runId" : 123,
    "callsInHopper" : 10,
    "callsByState" : {
      "IN_HOPPER" : 10
    },
    "reschedules" : 0,
    "reschedulingSoon" : 0,
    "reschedulingLater" : 0
  }, {
    "campaignName" : "ABC Mexico",
    "runName" : "15.04.02 14:39:49",
    "runId" : 124,
    "callsInHopper" : 134,
    "callsByState" : {
      "IN_HOPPER" : 134
    },
    "reschedules" : 257,
    "reschedulingSoon" : 254,
    "reschedulingLater" : 3
  } ],
  "reschedule_late_limit_sec" : 300
}
```

Add (or restart) list to a running campaign

It is possible to add a list that the campaign was not originally started with to a running campaign. Lists are processed in sequence, so it will generally be read only after the original lists are fully scanned.

```
http://127.0.0.1:8080/wombat/api/runlists/?op=start&campaign=Test&list=L1
```

If the run already has that list and the list is paused, then it is unpaused.

If you are looking at the current lists from the GUI, then you will have to refresh them to see the changes you made.

Pause list in a running campaign

In order to pause a list on a running campaign, you can use the following call.

```
http://127.0.0.1:8080/wombat/api/runlists/?op=pause&campaign=Test&list=L1
```

If the list does not exist on your run, the service completes successfully but nothing happens.

In order to unpause a paused list, you call the *start* operation again.

Please note that pausing a list stops Wombat from fetching further numbers from the paused list, but does not prevent Wombat to call numbers it has already fetched that are ready to be called. Using a smaller "batch size" prevents Wombat to fetch too many numbers in advance, and so might be of interest if you need to pause/unpause lists often (at the price of a higher database load).

A campaign that has paused lists will not terminate naturally, but will remain active until either you unpause those lists or you forcibly remove the campaign.

Show lists on a running campaign

You can query the association of lists to a specific running campaign.

```
http://127.0.0.1:8080/wombat/api/runlists/?op=list&campaign=Test
```

The output shows all lists, specifying whether they are currently paused and whether they are already finished.

```
[ {
  "listId" : 2,
  "listName" : "TestList2",
  "highWater" : 131,
  "isRunning" : false,
  "isFinished" : false
}, {
  "listId" : 3,
  "listName" : "L1",
  "highWater" : 1203,
  "isRunning" : true,
  "isFinished" : true
} ]
```

Set boosting on a running campaign

You can set the boost factor on a running campaign:

```
http://127.0.0.1:8080/WombatDialer/api/campaigns/?op=boost&campaign=TestCampaign&facto
```

```
r=78
```

The boost factor is expressed as an integer percentage and must be between zero and 500 (5x).

Call lists

Add numbers to a list

This call adds the numbers passed in *numbers* to the call list ZEB_2 - if it does not exist, it will create it. Numbers are a pipe-separated list of numbers and optional attributes.

```
http://127.0.0.1:8080/wombat/api/lists/?op=addToList
       &list=ZEB_2&numbers=1234,AA:bb,CC:dd|5678,XX:yy
```

If you set the parameter *dedupe=1* then numbers will be checked for existence on this this list before being added. This requires a significant database load, so use with caution on a running system where lists are over 10,000 elements each. Numbers are considered to be existent if they are present on the list no matter what their attributes might be.

It is very common to use HTTP POST to add numbers, so that you can upload a large set at once. The maximum request size depends on your servlet container, but is usually in the order of 1-2 megabytes.

Cancel numbers from a blacklist

This call lets you cancel a number from being blacklisted on a specific list:

```
http://127.0.0.1:8080/wombat/api/calls/
  ?op=cancelBlacklist
  &list=DNC
  &numbers=5551000
  &reason=Incall1
```

This means that if there is number 5551234 on the list DNC that can be actively blacklist, mark it so that it is not used for black-listing, and write the reason for de-blacklisting it as "Incall1" (it's just a string where you may store anything that makes sense to you). If the number is present multiple times, then all of them will have their attributes `BLACKLISTED_CANCEL` set to "Incall1 @2023-11-10.18:31:32".

If you specify no reason, then the current date will be used to set `BLACKLISTED_CANCEL`.

You can cancel blacklisting of one or more numbers at once - if you wish to cancel multiple numbers, they must be separated by a pipe symbol, and it must be properly URL-encoded.

The webservice returns a synchronous reply like `OK: Numbers updated: 6 (N Lists 1 - Numbers found: 9)` - that means that the list you asked was found, that the number was present 9 times, bus at three of them were already cancelled or expired, changes were made on just 6 entries.

Stow away: rename and hide lists in one go

Sometimes it is useful to work with lists that always have the same names, and to remove them when they are not used. As lists cannot be deleted from a Wombat system, it is useful to "stow away" them by renaming them and hiding them in one go.

```
http://127.0.0.1:8080/wombat/api/lists/?op=stowAway&list=LEADS&as=LEADS_170112
```

This call renames your current list LEADS to LEADS_170112 and hides it; at this point you can simply create a new list called LEADS and use it.

You will have to manually remove the list from any campaigns that might be using it; otherwise they will keep using the old (hidden) list.

Preview dialing

You can control Preview Reverse dialing through the API. A somewhat simpler alternative may be using the Preview page.

Reserve a call

In order to reserve a call, you should call WombatDialer by issuing a request like the following:

```
http://127.0.0.1:8080/wombat/api/reserve/?op=reserve&agent=SIP/302
```

It is of paramount importance that the **agent** parameter matches an agent code as present in Asterisk.

If all goes well, the reply is like:

```
OK: OK
WOMBATID: 366253628
NUMBER: 301
VAR: A=1
VAR: B=2
```

You should make a note of the WOMBATID as it will be used in subsequent calls. If you call this API multiple times, the same call will be returned as long as the agent has one reserved call.

Error codes include:

- AGENT_NOT_AVAILABLE: The agent is not present, or is busy o paused
- AGENT_CANNOT_RESERVE: No available call for the agent to make - there is no running campaign that points to that agent where there is free space on the trunk and there is a call to be made, or the agent is busy elsewhere

Place a reserved call

In order to place a reserved call, you need to feed Wombat its WOMBATID:

```
http://127.0.0.1:8080/wombat/api/reserve/?op=dial&id=123456
```

Returns

- DIALING: the call is being placed
- ID_NOT_FOUND: No such ID
- ID_FOUND_WRONG_STATE: The ID exists but the call is in a wrong state

Skip a reserved call

In order to skip a reserved call, you need to feed Wombat its WOMBATID:

```
http://127.0.0.1:8080/wombat/api/reserve/?op=skip&id=123456&withStatus=XXX
```

The parameter "withStatus" is optional and that will be the extStatus for the call being skipped. You can use this to get fine control on whether it is to be rescheduled and how.

Returns:

- SKIPPED: the call was skipped
- ID_NOT_FOUND: No such ID
- ID_FOUND_WRONG_STATE: The ID exists but the call is in a wrong state

Live call details

You can get the details of a currently live or scheduled call.

```
http://127.0.0.1:8080/wombat/api/callinfo/?wid=WBT.1234
```

The parameter "wid" contains the WombatId. It can contain a prefix "WBT" as it is logged on QueueMetrics logs or it can be without it. As an alternative, you can pass the parameter "unqid" that contains an Asterik's UniqueID for the call.

The method will:

- look for the call on the Live calls table (returns "isLive" as true)
- look for the call in the logs (returns "isLive" as false)

Wombat will return a JSON data structure of the format:

```
{
```

```
  "callFound" : true,
  "isLive": true,
  "callRecord" : {
    "callId" : 418760,
    "listId" : {
      "listId" : 95,
      "name" : "Z7_04142015",
      "isHidden" : false,
      "securityKey" : ""
    },
    "number" : "9377893386",
    "attributes" : [ {
      "isInput" : true,
      "attrName" : "FIRST NAME",
      "attrValue" : "KATRINA"
    } ]
  },
  "call" : {
    "hopperCallId" : 212783745,
    "number" : "9377893386",
    "numberId" : 418760,
    "state" : "IN_HOPPER",
    "trunk" : "ABC Trunk",
    "astServer" : "ABC",
    "trunk_size" : 40,
    "endpoint" : "QUEUE Zebra Seven",
    "campaignRun" : "Z7_1210 15.04.16 12:22:57",
    "tst_originate" : 0,
    "tst_connect" : 0,
    "tst_terminate" : 0,
    "astUnique" : "",
    "astChannel" : "",
    "nRetry" : 0,
    "statusExt" : "",
    "subEp" : "SIP/21071",
    "bridgedChan" : ""
  }
}
```


The most important items are:

- callFound: whether a call with the ID was found or not
- isLive: whether the returned result comes from a live call or from the logs (completed call)
- call: the current details of the live call
- callRecord: the details of the number being called - its current attributes and the list it belongs to.

System health

It is possible to receive a JSON structure with general information about the system. It is meant to be used by external monitoring systems that need to make sure WombatDialer is working.

```
http://127.0.0.1:8080/wombat/api/sysup/
```

The response will be similar to the one below:

```
{
  "state" : "WBTUP",
  "ramFreeMb" : 73,
  "ramTotalMb" : 139,
  "ramMaxMb" : 1818,
  "db_access_ms" : 4,
  "generatedOn" : "Mon Sep 29 16:04:00 CEST 2014",
  "version" : "0.8.0/#1234"
}
```

When this transaction is run, the database is accessed to make sure it is available. If the response block does not contain the string "WBTUP" the application is supposed to be down and in need of a restart.

Even when you get the state `WBTUP`, you know that the web application is available, but the dialer itself might be stopped, like when you start and stop it from the GUI. See Engine control.

JMX statistics

WombatDialer traces a wealth of information about its runtime performance as JMX beans that are available to JMX pollers. As JMX is notoriously hard to set up correctly, it is also possible to read the majority of that data using a webservice:

```
http://127.0.0.1:8080/wombat/api/sysup/jmx/
```

The response will be similar to the one below:

```
{
  "\"WD/AMI-127.0.0.1:15038#6.actionsSize\":value" : "0",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":50th" : "11.229",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":95th" : "12.64",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":count" : "166",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":max" : "17.977",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":min" : "10.04",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":rate_sec" : "15.484",
  "\"WD/AMI-127.0.0.1:15038#6.looptime\":rate_sec_oneminute" : "15.416",
```

```
  "\"WD/AMI-127.0.0.1:15038#6.msg.all\":value" : "ERROR",
  "\"WD/AMI-127.0.0.1:15038#6.msg.check\":value" : "ERROR",
  "\"WD/AMI-127.0.0.1:15038#6.msg.originate\":value" : "ERROR",
  "\"WD/AMI-127.0.0.1:15038#6.queue-size\":value" : "ERROR",
  "generatedOn" : "Fri Nov 15 14:56:14 CET 2019",
  "jmx_query_ms" : "28",
  "jvm_code_cache" : "18897472",
  "jvm_cpu_load" : "0",
  "jvm_daemon_threads" : "24",
  "jvm_heap_max" : "3817865216",
  "jvm_heap_used" : "141595248",
  "jvm_loaded_classes" : "6145",
  "jvm_nonheap_max" : "-1",
  "jvm_nonheap_used" : "61834768",
  "jvm_thread_count" : "25",
  "jvm_total_compilation_time" : "12576",
  "jvm_unloaded_classes" : "0",
  "wd-hlp.autorun:50th" : "0",
  "wd-hlp.autorun:95th" : "0",
  "wd-hlp.autorun:count" : "0",
  "wd-hlp.autorun:max" : "0",
  "wd-hlp.autorun:min" : "0",
  "wd-hlp.autorun:rate_sec" : "0",
  "wd-hlp.autorun:rate_sec_oneminute" : "0",
  "wd-hlp.email:50th" : "0",
  "wd-hlp.email:95th" : "0",
  "wd-hlp.email:count" : "0",
  "wd-hlp.email:max" : "0",
  "wd-hlp.email:min" : "0",
  "wd-hlp.email:rate_sec" : "0",
  "wd-hlp.email:rate_sec_oneminute" : "0",
  "wd-hlp.http:50th" : "0",
  "wd-hlp.http:95th" : "0",
  "wd-hlp.http:count" : "0",
  "wd-hlp.http:max" : "0",
  "wd-hlp.http:min" : "0",
  "wd-hlp.http:rate_sec" : "0",
  "wd-hlp.http:rate_sec_oneminute" : "0",
  "wd-log.cycles:50th" : ".003",
  "wd-log.cycles:95th" : ".022",
  "wd-log.cycles:count" : "115",
  "wd-log.cycles:max" : ".067",
  "wd-log.cycles:min" : ".001",
  "wd-log.cycles:rate_sec" : "9.752",
  "wd-log.cycles:rate_sec_oneminute" : "9.784",
  "wd-log.n_logs:value" : "ERROR",
  "wombat.all.brain:50th" : "4.662",
  "wombat.all.brain:95th" : "11.947",
  "wombat.all.brain:count" : "199",
  "wombat.all.brain:max" : "18.31",
  "wombat.all.brain:min" : ".004",
```

```
  "wombat.all.brain:rate_sec" : "16.871",
  "wombat.all.brain:rate_sec_oneminute" : "16.112",
  "wombat.all.msg-in:50th" : ".013",
  "wombat.all.msg-in:95th" : "3.632",
  "wombat.all.msg-in:count" : "199",
  "wombat.all.msg-in:max" : "159.502",
  "wombat.all.msg-in:min" : ".004",
  "wombat.all.msg-in:rate_sec" : "16.862",
  "wombat.all.msg-in:rate_sec_oneminute" : "16.112",
  "wombat.all.runafter:50th" : ".003",
  "wombat.all.runafter:95th" : ".185",
  "wombat.all.runafter:count" : "199",
  "wombat.all.runafter:max" : "7.832",
  "wombat.all.runafter:min" : ".002",
  "wombat.all.runafter:rate_sec" : "16.878",
  "wombat.all.runafter:rate_sec_oneminute" : "16.112",
  "wombat.brain.calls:50th" : "2.82",
  "wombat.brain.calls:95th" : "5.177",
  "wombat.brain.calls:count" : "198",
  "wombat.brain.calls:max" : "6.772",
  "wombat.brain.calls:min" : "1.183",
  "wombat.brain.calls:rate_sec" : "16.787",
  "wombat.brain.calls:rate_sec_oneminute" : "15.928",
  "wombat.brain.reschedule:50th" : "0",
  "wombat.brain.reschedule:95th" : ".013",
  "wombat.brain.reschedule:count" : "198",
  "wombat.brain.reschedule:max" : ".019",
  "wombat.brain.reschedule:min" : "0",
  "wombat.brain.reschedule:rate_sec" : "16.786",
  "wombat.brain.reschedule:rate_sec_oneminute" : "15.928",
  "wombat.brain.runs:50th" : ".004",
  "wombat.brain.runs:95th" : ".02",
  "wombat.brain.runs:count" : "198",
  "wombat.brain.runs:max" : ".051",
  "wombat.brain.runs:min" : ".002",
  "wombat.brain.runs:rate_sec" : "16.786",
  "wombat.brain.runs:rate_sec_oneminute" : "15.928",
  "wombat.hibernate.commit:50th" : ".576",
  "wombat.hibernate.commit:95th" : "3.366",
  "wombat.hibernate.commit:count" : "201",
  "wombat.hibernate.commit:max" : "12.056",
  "wombat.hibernate.commit:min" : ".242",
  "wombat.hibernate.commit:rate_sec" : "17.044",
  "wombat.hibernate.commit:rate_sec_oneminute" : "16.48",
  "wombat.hibernate.evict:50th" : ".001",
  "wombat.hibernate.evict:95th" : ".001",
  "wombat.hibernate.evict:count" : "201",
  "wombat.hibernate.evict:max" : ".003",
  "wombat.hibernate.evict:min" : "0",
  "wombat.hibernate.evict:rate_sec" : "17.052",
  "wombat.hibernate.evict:rate_sec_oneminute" : "16.48",
```

```
  "wombat.hibernate.flush:50th" : ".462",
  "wombat.hibernate.flush:95th" : "1.45",
  "wombat.hibernate.flush:count" : "198",
  "wombat.hibernate.flush:max" : "6.107",
  "wombat.hibernate.flush:min" : ".199",
  "wombat.hibernate.flush:rate_sec" : "16.806",
  "wombat.hibernate.flush:rate_sec_oneminute" : "15.928",
  "wombat.hibernate.openconnection:50th" : "155.41",
  "wombat.hibernate.openconnection:95th" : "155.41",
  "wombat.hibernate.openconnection:count" : "1",
  "wombat.hibernate.openconnection:max" : "155.41",
  "wombat.hibernate.openconnection:min" : "155.41",
  "wombat.hibernate.openconnection:rate_sec" : ".085",
  "wombat.hibernate.openconnection:rate_sec_oneminute" : ".184",
  "wombat.hibernate.save:50th" : "0",
  "wombat.hibernate.save:95th" : "0",
  "wombat.hibernate.save:count" : "0",
  "wombat.hibernate.save:max" : "0",
  "wombat.hibernate.save:min" : "0",
  "wombat.hibernate.save:rate_sec" : "0",
  "wombat.hibernate.save:rate_sec_oneminute" : "0",
  "wombat.refill.new:50th" : "0",
  "wombat.refill.new:95th" : "0",
  "wombat.refill.new:count" : "0",
  "wombat.refill.new:max" : "0",
  "wombat.refill.new:min" : "0",
  "wombat.refill.new:rate_sec" : "0",
  "wombat.refill.new:rate_sec_oneminute" : "0",
  "wombat.refill.retry:50th" : "0",
  "wombat.refill.retry:95th" : "0",
  "wombat.refill.retry:count" : "0",
  "wombat.refill.retry:max" : "0",
  "wombat.refill.retry:min" : "0",
  "wombat.refill.retry:rate_sec" : "0",
  "wombat.refill.retry:rate_sec_oneminute" : "0",
  "wombat_version" : "0.0.0/#1234"
}
```


Data is measured (depending on what it is) as:

- `...:50th`: the median value of a timer.
- `...:95th`: the 95th percentile of a run time
- `...:count`: the number of times a time was measured
- `...:max`: maximum duration
- `...:min`: minimum duration
- `...:rate_sec`: number of times per second the timer is triggered
- `...:rate_sec_oneminute`: number of times per second the timer was triggered in the last minute

The fields have the following meanings (as better explained in Monitoring a running instance):

- General Wombat statistics:
  - `wombat.all.brain`: how long Wombat spent making decisions
  - `wombat.all.msg-in`: how long Wombat spent processing incoming messages
  - `wombat.all.runafter`: how long was spent running periodic jobs
- Decision maker:
  - `wombat.brain.brain`: how long was spent making decisions on calls to dial
  - `wombat.brain.calls`: how long was spent placing calls
  - `wombat.brain.reschedule`: time spent evicting or rescheduling calls
  - `wombat.brain.runs`: how long was spent starting and stopping runs
- Database layer:
  - `wombat.hibernate.commit`: time committing changes
  - `wombat.hibernate.evict`: time evicting stale objects from cache
  - `wombat.hibernate.flush`: time flushing database changes
  - `wombat.hibernate.save`: time saving new objects
  - `wombat.hibernate.refresh`: time doing a deep refresh of campaigns
  - `wombat.hibernate.openconnection`: how many connections were open. Usually only increments only on restarts.
- Fetching of calls from lists:
  - `wombat.refill.new`: time spent reading new calls from lists
  - `wombat.refill.retry`: time spent reading calls from retry cache
  - `wombat.refill.blacklist`: time spent checking blacklists
- For each PBX Head:
  - `(AMI head name).actionsSize`: actions to process
  - `(AMI head name).all`: all messages
  - `(AMI head name).check`: number of liveness checks
  - `(AMI head name).originate`: number of originates
  - `(AMI head name).queue-size`: current dialing queue size
  - `(AMI head name).fromAsterisk`: number of events received from Asterisk
  - `(AMI head name).fromMaster`: number of commands received from Wombat
  - `(AMI head name).connected`: 1 if connected, 0 if disconnected
- Helpers
  - `wd-hlp.autorun`: deferred actions
  - `wd-hlp.email`: time spent sending emails
  - `wd-hlp.http`: time spent doing HTTP notifications

- Logger
  - `wd-log.cycles`: commit timer
  - `wd-log.n_logs`: number of logs received to be written
- JVM stats
  - All stats beginning with `jvm_` are internal JVM stats
- Misc
  - `generatedOn`: when the request was generated
  - `jmx_query_ms`: total JMX query time
  - `wombat_version`: the version of Wombat

All times are in milliseconds.

Adding a license key

You may add a license key to a working WombatDIaler system or query its current licensing. This is meant for dynamic, short-term key assignment.

The following call adds key "ABCD.1234" and activates it:

```
http://127.0.0.1:8080/wbt/api/addkey/?op=add&key=ABCD.1234
```

The response will display the state of all installed keys on the system:


```
{
  "requestId" : "",
  "lockUntilComplete" : true,
  "responseStatus" : "OK",
  "responseMessage" : "",
  "toPanel" : "nopanel",
  "responseLog" : null,
  "keyText" : "",
  "isAdd" : true,
  "lLic" : [ {
    "licenseId" : "DEMO070",
    "licensedTo" : "DEMO / WombatDialer 070",
    "startsOn" : "2012-12-18",
    "endsOn" : "2013-12-18",
    "currState" : "REVOKED",
    "nChannels" : 2
  }, {
    "licenseId" : "LLX1234",
    "licensedTo" : "LLX1234 / Camp3x",
    "startsOn" : "2015-07-02",
    "endsOn" : "2016-07-10",
    "currState" : "ACTIVE",
```

```
    "nChannels" : 8
  } ],
  "l_px" : "257377543787",
  "l_ch" : 8,
  "processor" : "ch.loway.app.wombat.ui.svr.LicensingProcessor"
}
```

The value "l_ch" is the total number of licensed channels currently active, while details of existing licenses can be seen from "lLic".

The JSON configuration API

WombatDialer exposes a JSON configuration API that lets you fully configure WombatDialer - as you would through the GUI - using a JSON API. This makes it easy to deploy multiple instances of Wombat for very large systems handing thousands of channels at once. It also lets you configure campaigns to be run automatically, each with the correct configuration.

We have some tutorial scripts that let you see the concepts explained below in action - they are located at https://github.com/Loway/OpenWombatDialerAddOns under the folder `JSON_Configuration_API`. Get a copy and make sure you understand how they work before writing your own.

General concepts

JSON API requests can be of four kinds:

- LIST: returns a paged list of objects. You can tell Wombat the page you want to read by specifying a start offset and an expect result size. Plus, you can optionally send along a filter parameter that will be used to further refine the results.
- EDIT: you push a JSON object to be added to the Wombat database. If you define a primary key for the object, the object will be updated; if the key is empty (as set to JSON *null*) then the object will be added. A basic sanity check will be performed on inserts and updates. The result of an update is typically the first page of results.
- DELETE: The JSON object will be deleted. The object must include a primary key to tell Wombat which object is to be deleted.
- MOVE UP/DOWN: on some lists (e.g. reschedule rules) it is possible to move an object up or down in the list.

You specify the operation that is to be performed by passing the following parameters:

- *mode*: shall be *L* for list, *E* for editing, *D* for delete, *MU* and *MD* to move the object up and down. If not specified, listing is assumed.
- *data*: the JSON object itself. This is not needed when listing.
- *parent*: some objects depend on other objects being selected first (see the description of dependent objects below). This is the primary key of the main object that holds those sub-objects.

- *from*: the beginning of the list to be returned. If missing, zero is assumed.
- *items*: the maximum number of items to be returned. If missing, 100 is used.
- *query*: a string that is to be searched in text fields. Depends on the object - this behaves like the search boxes you see on the GUI.
- For any JSON query, you need to send Wombat an active user who holds the grants to perform the required operation, passing it as a Basic HTTP authentication.

For example, the following query lists available trunks through the popular *curl* command-line HTTP client:

```
curl  --user demoadmin:demo -i -X POST
"http://127.0.0.1:8080/wombat/api/edit/trunk/?mode=L"
```

The result will be something like:

```
{
  "status" : "OK",
  "error" : "",
  "results" : [ {
    "sys_dt_creazione" : "2015-03-18 17:02:46",
    "sys_user_creazione" : "Demo Admin",
    "sys_dt_modifica" : "",
    "sys_user_modifica" : "-",
    "trunkId" : 1,
    "astId" : {
      "sys_dt_creazione" : "2015-03-18 17:02:29",
      "sys_user_creazione" : "Demo Admin",
      "sys_dt_modifica" : "2015-03-23 15:49:45",
      "sys_user_modifica" : "Demo Admin",
      "id" : 1,
      "description" : "Asterisk Server",
      "ipAddress" : "127.0.0.1",
      "login" : "hello",
      "password" : "kitty",
      "port" : 5038,
      "unit_len" : 50,
      "msg_per_unit" : 5,
      "securityKey" : "",
      "state" : "DOWN"
    },
    "name" : "My Trunk",
    "dialstring" : "Local/${num}@from-internal/n",
    "capacity" : 10,
    "securityKey" : ""
  } ],
  "allResults" : 1,
  "startFrom" : 0,
  "nRecords" : 1,
```

```
  "timeMs" : 12
}
```

The JSON response always includes:

- a *status* that should be *OK* if all went well. If something went wrong, you will have a description in the *error* field (usually a Java stack trace).
- a list named *results* that may include zero or more elements. These are the objects you are querying.
- an estimated size for all possible results to this query passed as *allResults*.
- the point in the list of results from where results are sent back (*startFrom*) and the amount of records on the current page (*nRecords*).
- how long the operation took on the server, expressed in milliseconds, as *timeMs*.

If you want to insert or upgrade an object, you first need to save it as a JSON entity. All objects specify a primary key that you must use to make sure that the object stays the same during updates. Any access information - who created and updated the object, and when - is updated automatically.

So to change the name of the previous trunk, you could create a file called *newtrunk.json* that contains:

```
{
    "trunkId" : 1,
    "astId" : {
      "id" : 1
    },
    "name" : "NewName",
    "dialstring" : "Local/${num}@from-internal/n",
    "capacity" : 10,
    "securityKey" : ""
}
```

Note that:

- We use the *trunkId* we were supplied by reading, as it is the primary key for this object.
- All user and time-stamp information were removed. There is no harm in leaving it in, but it will not be used when updating.
- Any inner objects (like, in our example, the Asterisk server) will NOT be updated on the same query. It is safe to remove everything but the primary key (in our case *id*), as the object will be loaded by primary key in order to be correctly associated. You cannot associate to objects that are not existing at the moment the request is issued.

We can then push this new object by issuing:

```
curl  --user demoadmin:demo -i -X POST --data-urlencode data@newtrunk.json
```

```
"http://127.0.0.1:8080/wombat/api/edit/trunk/?mode=E"
```

Please note that you can mix-and-match GET and POST parameters as you best see fit.

Working with dependent objects

Dependent object are objects that have a parent and that make no sense without specifying their parent. For example, reschedule rules depend on a campaign and have no meaning outside of it. To work with a dependent object, you must specify the primary key of its parent, like in:

```
curl  --user demoadmin:demo -i -X POST
"http://127.0.0.1:8080/wbt/api/edit/campaign/reschedule/?parent=1"
```

The response will be like:

```
{
  "status" : "OK",
  "error" : "",
  "results" : [ {
    "rescheduleRuleId" : 1,
    "status" : "RS_NOANSWER",
    "statusExt" : "",
    "maxAttempts" : 1,
    "retryAfterS" : 120,
    "mode" : "FIXED",
    "idx" : "1",
    "campaignId" : 1
  } ],
  "allResults" : 1,
  "startFrom" : 0,
  "nRecords" : 1,
  "timeMs" : 18
}
```

Dependent objects always appear in the API "below" their main object, as in "/edit/campaign/reschedule/".

In order to insert a new rule for campaign #1, you would create a new entry like:

```
{
    "rescheduleRuleId" : null,
    "status" : "RS_NOANSWER",
    "statusExt" : "",
    "maxAttempts" : 1,
    "retryAfterS" : 120,
    "mode" : "FIXED",
    "campaignId" : 1
```

```
}
```

and push it like:

```
curl  --user demoadmin:demo -i -X POST --data-urlencode data@rrnew.json
"http://127.0.0.1:8080/wbt/api/edit/campaign/reschedule/?mode=E&parent=1"
```

Note that the primary key of the parent appears in both the object - as *campaignId* - and as *parent* in the JSON query.

If we want to move the object up (MU) or down (MD), we can edit the *rrnew.json* file to set the new *rescheduleRuleId* and:

```
curl  --user demoadmin:demo -i -X POST --data-urlencode data@rrnew.json
"http://127.0.0.1:8080/wbt/api/edit/campaign/reschedule/?mode=MU&parent=1"
```

And to delete it we would:

```
curl  --user demoadmin:demo -i -X POST --data-urlencode data@rrnew.json
"http://127.0.0.1:8080/wbt/api/edit/campaign/reschedule/?mode=D&parent=1"
```

Using the JSON API in practice

- Issue a list query using the shell to see available objects and how they are represented.
- To create a new object, create it manually from the GUI and read it through JSON to see how it "looks like".
- The primary key for any object appears in the object definition
- When listing objects, **always** take advantage of paged queries if you think that there might be more than a hundred objects returned. Reading and serializing thousands or tens of thousands of elements at once is not a good idea! On the other hand, returning hundreds of thousands of objects, 500/1000 at a time, is usually safe. Remember that it is important to avoid running extremely expensive operations that might impact the dialer itself by using too much CPU, contending the database or - worse - crashing the JVM by exhausting all of its available RAM! So while it is hard to put general rules that depend on the kind of hardware you are running and on the current load, it is always better to err on the side of caution.
- Objects are usually returned in descending order of insertion, so your shiny new object is typically the first one returned.

Available JSON APIs

The URL expressed in the table below are only the URL fragments that have to be appended to your WombatDialer API main URL - for example, if the table lists *asterisk* as the fragment, the URL will be *http://127.0.0.1:8080/wbt/api/edit/asterisk/* (note the mandatory trailing slash).

Table 4. JSON API

URL fragment	Description	Primary key name	Depends on?	Moves Up/Down	Security key
asterisk	Asterisk servers	id	-	No	ADMIN
ep	End-points	epId	-	No	ADMIN
trunk	Trunks	trunkId	-	No	ADMIN
oh	Opening Hours	openingHourId	-	No	OPHR
campaign	Campaigns	campaignId	-	No	CAMP
campaign/ep	EPs on campaigns	-	Campaign	No	CAMP
campaign/trunk	Tks on campaigns	-	Campaign	No	CAMP
campaign/list	Lists on campaigns	cclId	Campaign	Yes	CAMP
campaign/reschedule	Reschedule Rules	rescheduleRuleId	Campaign	Yes	CAMP
campaign/disposition	Cmp. Dispositions	dispositionId	Campaign	Yes	CAMP
campaign/oh	Cmp. Opening Hours	cohId	Campaign	Yes	CAMP
list	A call list	listId	-	No	LIST
list/record	Numbers on a list	callId	List	No	LIST
list/logs	Called numbers	id	List	No	LIST

Notes:

- OpeningHours are composite objects, so any inner rule is handled as a private JSON structure.
- Campaign EPs and Trunks are checked for uniqueness, as they do not have an inner order. Wombat will make sure that they cannot be added more than once.
- A field named *idx* is used to track the position of items in a sorted list.

The JSON Live and Reporting API

The JSON live API lets you query the state of the system as seen from the Live and Reports page.

The following table shows a succinct description of what is available.

URL fragment	Description	Result key	Depends on?	Security key
live/calls	Live calls	-	-	RT
live/runs	Active campaigns	-	-	RT
reports/runs	Runs for period	hoppercampId	from / to	REPORT
reports/stats	Stats for runs	-	id	REPORT
reports/logs	Logs for runs	-	id/from/items	REPORT

Live Calls

You can get a list of live calls and available campaigns by issuing:

```
curl  --user demoadmin:demo -i -X POST "http://127.0.0.1:8080/wombat/api/live/calls/"
```

The results includes:

- A list of possible campaigns, their IDs and whether they are running
- A list of live calls, under "hopperState"

For example:


```
{
  "status" : "OK",
  "error" : "",
  "timeMs" : 9,
  "result" : {
    "requestId" : "",
    "lockUntilComplete" : false,
    "responseStatus" : "OK",
    "responseMessage" : "",
    "toPanel" : "API_CALL",
    "responseLog" : null,
    "campaigns" : [ {
      "campaignId" : 1,
      "name" : "My Campaign",
      "isRunning" : true,
      "secKey" : ""
    } ],
    "asterisks" : [ ],
    "hopperState" : [ {
      "hopperCallId" : 554418377,
      "number" : "12345678991",
      "state" : "DIALLING",
      "trunk" : "MyTrunk",
      "astServer" : "A44",
      "trunk_size" : 10,
      "endpoint" : "ep8",
```

```
      "campaignRun" : "test 15.03.18 17:04:32",
      "tst_originate" : 1427475059650,
      "tst_connect" : 0,
      "tst_terminate" : 0,
      "astUnique" : "1427475059.7189",
      "astChannel" : "Local/12345678991@wdtrunk-00000dcf;1",
      "nRetry" : 0,
      "statusExt" : "",
      "subEp" : "",
      "bridgedChan" : ""
    }]
  }
}
```

Live Campaigns

You can get information on Live campaigns by issuing:

```
curl  --user demoadmin:demo -i -X POST "http://127.0.0.1:8080/wombat/api/live/runs/"
```

The result includes an item called "campaigns" where you can see the live status:

```
{
  "status" : "OK",
  "error" : "",
  "timeMs" : 2404,
  "result" : {
    "requestId" : "",
    "lockUntilComplete" : false,
    "responseStatus" : "OK",
    "responseMessage" : "",
    "toPanel" : "API_CALL",
    "responseLog" : null,
    "action" : "NONE",
    "hopcampId" : 0,
    "campaignId" : 0,
    "campaigns" : [ {
      "name" : "cccd",
      "startedAt" : "15.03.18 17:04:32",
      "campaignId" : 1,
      "hoppercampId" : 2,
      "priority" : 10,
      "state" : "IDLE",
      "n_calls_completed" : 0,
      "n_open_retries" : 0,
      "n_calls_attempted" : 0,
      "n_est_remaining_calls" : 0,
      "secondsSinceStarted" : 771835,
```

```
      "listName" : "#0",
      "listPos" : 0,
      "timeDow" : "1234567",
      "timeStartHr" : "000000",
      "timeEndHr" : "235959",
      "predictiveModel" : "OFF",
      "boostFactor" : 1.0,
      "dowAsString" : "Su | Mo | Tu | We | Th | Fr | Sa"
    } ]
  }
}
```

Reports: runs in period

This call gets you a list of runs (identified by their "hoppercampId") that were made in a specific period:

```
curl  --user demoadmin:demo -i -X POST
"http://127.0.0.1:8080/wbt/api/reports/runs/?from=2000-01-01.00:00:00&to=2020-01-
01.00:00:00"
```

Both the "from" and "to" parameters are mandatory.

The result is a list of runs:

```
{
  "status" : "OK",
  "error" : "",
  "timeMs" : 69,
  "result" : {
    "requestId" : "",
    "lockUntilComplete" : true,
    "responseStatus" : "OK",
    "responseMessage" : "",
    "toPanel" : "API_CALL",
    "responseLog" : null,
    "dateFrom" : 946681200000,
    "dateTo" : 1577833200000,
    "campaignsAndRuns" : [ {
      "name" : "test",
      "startedAt" : "15.03.18 17:04:32",
      "campaignId" : 1,
      "hoppercampId" : 2,
      "priority" : 10,
      "state" : "RUNNING",
      "n_calls_completed" : 91,
      "n_open_retries" : 109,
      "n_calls_attempted" : 91,
```

```
      "n_est_remaining_calls" : 9911,
      "secondsSinceStarted" : 782167,
      "listName" : "appero",
      "listPos" : 0,
      "timeDow" : "1234567",
      "timeStartHr" : "000000",
      "timeEndHr" : "235959",
      "predictiveModel" : "OFF",
      "boostFactor" : 1.0,
      "dowAsString" : "Su | Mo | Tu | We | Th | Fr | Sa"
    } ]
  }
}
```


Report: stats for runs

This call runs statistics for a set of runs, identified by the parameter "id" that can be present multiple times.

```
curl  --user demoadmin:demo -i -X POST
"http://127.0.0.1:8080/wbt/api/reports/stats/?id=1&id=7&id=123"
```

The result is a set of stats by call type:


```
{
  "status" : "OK",
  "error" : "",
  "timeMs" : 8,
  "result" : {
    "requestId" : "",
    "lockUntilComplete" : true,
    "responseStatus" : "OK",
    "responseMessage" : "",
    "toPanel" : "API_CALL",
    "responseLog" : null,
    "campaignsToConsider" : [ 1, 7, 123 ],
    "statsOut" : [ {
      "gbCampaignId" : 1,
      "gbState" : "RS_BUSY",
      "gbTrunk" : "A44 NewName",
      "gbStateExt" : "",
      "nCalls" : 90,
      "totWaitPre" : 10,
      "totWaitAfter" : 271,
      "totTalk" : 0
    }, {
      "gbCampaignId" : 1,
      "gbState" : "RS_NUMBER",
```

```
      "gbTrunk" : "A44 NewName",
      "gbStateExt" : "",
      "nCalls" : 1,
      "totWaitPre" : 0,
      "totWaitAfter" : 0,
      "totTalk" : 0
    } ]
  }
}
```

Reports: logs by runs

Given a set of runs, gets the details of calls placed.

Parameters:

- *from*: the start of results in the log set
- *items*: how many logs you want back (page length)
- *id*: the run ids - can be repeated multiple times

```
curl  --user demoadmin:demo -i -X POST
"http://127.0.0.1:8080/wbt/api/reports/logs/?from=0&items=3&id=1&id=2&id=123"
```

Do not forget to set "sane" paging criteria, to avoid getting a result set that's too large for available memory.

The results look like:

```
{
  "status" : "OK",
  "error" : "",
  "timeMs" : 12,
  "results" : [ {
    "dim_id" : 1,
    "listId" : 1,
    "listName" : "GoodList",
    "trunkId" : 1,
    "trunkName" : "NewName",
    "astId" : 1,
    "astName" : "",
    "epId" : 4,
    "epName" : "ep7",
    "campId" : 1,
    "campName" : "test",
    "runId" : 2,
    "runName" : "15.03.18 17:04:32",
    "runStarted" : 1426694673000,
```

```
    "id" : 1,
    "callId" : 1,
    "callNumber" : "1",
    "numberDialed" : "1",
    "attributes" : "IN:puts ^OUT:put",
    "attempted" : 1427474910000,
    "waitPre" : 1,
    "waitAfter" : 111,
    "talk" : 0,
    "statusCode" : "RS_NUMBER",
    "astChannel" : "",
    "astUnique" : "",
    "nRetry" : 0,
    "statusExt" : "",
    "nextRetry" : 0,
    "wombatId" : "515144647",
    "subEp" : "",
    "bridgedChannel" : ""
  } ],
  "allResults" : 91,
  "startFrom" : 0,
  "nRecords" : 3
}
```

The result set is the same you'd get for the *edit/list/logs* call, but with different search criteria.

HTTP/S life-cycle notifications of calls

In order to enable HTTP life-cycle notifications, you must set a URL in your campaign definition that points to an HTTP script. This causes WD to POSTs the result of each call to your HTTP server.

Both "http" and "https" URLs are allowed.

On the server, you'll have a script that accepts notification, like the simple PHP script that follows:

```
<?
$out = "";
foreach($_POST as $name => $value) {
   $out .= "$name:$value ";
}
print($out);
error_log("RQ: $out",0, "", "");
?>
```

The script above basically logs all activity on the HTTP error log. What you get is a sequence of calls like:

```
RQ: num:5551234 reschedule:0 I_MM:30 extstate:
     state:RS_REJECTED I_HH:10 retry:0
RQ: num:5556785 reschedule:0 I_MM:00 extstate:
     state:RS_REJECTED I_HH:11 retry:0
RQ: num:5552012 reschedule:0 I_MM:30 extstate:
     state:RS_REJECTED I_HH:11 retry:0
RQ: num:5551234 reschedule:0 I_MM:30 extstate:1
     state:TERMINATED I_HH:10 O_APPT:APPE retry:0
RQ: num:5556785 reschedule:300 I_MM:00 extstate:
     state:RS_NOANSWER I_HH:11 retry:0
RQ: num:5552012 reschedule:0 I_MM:30 extstate:2
     state:TERMINATED I_HH:11 O_APPT:PSYC retry:0
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:1
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:2
RQ: num:5556785 reschedule:300 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:3
RQ: num:5556785 reschedule:0 I_MM:00 extstate:0
     state:TERMINATED I_HH:11 retry:4
```

You can see that for each call:

- `num` is set to the main number dialed and `currentnum` the number actually dialed (that might be different when using multinumbers).
- `listname` is the list that the number was taken from
- `state` is the call state at its completion
- `extstate` is the call's extended state, if present
- `retry` is the retry counter
- `reschedule` is set to the time in seconds to be waited before a reschedule is attempted; if no reschedule is required, it will be set to zero
- `asteriskid` is the current Asterisk-id of the call
- `wombatid` is the WombatId as it appears on the `queue_log`
- all **input** attributes (the ones you set with the telephone number) are passed along prepended by **I_**
- all **output** attributes (the ones you set in the Asterisk dialplan), if any, are passed along prepended by **O_**

This way you can easily create an integration script that stores the results of the call on a database or passes them along for further processing.

The HTTP notification server works asynchronously on a dedicated thread, so it is possible that there is some latency if your script has long completion times. The notification server does not retry on errors.

System configuration

Security keys

The following security keys are reserved for use by WombatDialer and have a pre-defined meaning.

Table 5. System Keys

Key	Meaning
USER	All interactive users must hold this key
RT	User can access the Live page
REPORT	User can run reports
ADMIN	User can edit configuration and users
SYSLOG	User can view the system log
LIST	User can view lists
CAMP	User can view campaigns
COPY	User can copy campaigns
LICKEY	User can add/edit licenses
V_WBT	User can view the dialer status
CTRL_WBT	User can start/stop the dialer
OPHR	User can edit Opening Hours
DEBUG_AMI	User can see the extended AMI status. Do not turn on in production unless explicitly told so.

All other strings can be freely used as "feature" security keys.

Default users

The following users ship in the default database installation.

They are meant as a template - you should create your own or at least change the default passwords. Failure to do so is a major security risk.

Table 6. Default users

Login	Password	Class	Notes
demoadmin	demo	ADMIN	
demouser	demo	USER	

For more information...

To know more about WombatDialer in your specific setting or inquire about commercial licenses, please feel free to contact Loway.

The latest version of WombatDialer can be found on the home page located at the address http://wombatdialer.com

There is a WombatDialer section in the QueueMetrics users forum for mutual support, troubleshooting and ideas at http://forum.queuemetrics.com

To stay updated on the latest developments of WombatDialer, you may:

- read our blog at https://www.wombatdialer.com/blog
- find some examples at https://github.com/Loway/OpenWombatDialerAddOns
- join the Facebook group at http://www.facebook.com/WombatDialer
- read our Twitter feed at http://twitter.com/WombatDialer