The sip:carrier Handbook mr3.8.13

Sipwise GmbH
<support@sipwise.com>

Contents

1 Introduction **1**

1.1 About this Document . . . 1

1.2 Getting Help . . . 1

1.2.1 Phone Support . . . 1

1.2.2 Ticket System . . . 1

1.3 What is the sip:carrier? . . . 2

1.4 What is inside the sip:carrier? . . . 2

1.5 Who should use the sip:carrier? . . . 2

2 System Architecture **3**

2.1 Hardware Architecture . . . 3

2.2 Component Architecture . . . 4

2.2.1 Provisioning . . . 5

API and Web Interface . . . 6

Provisioning Database . . . 7

Persistent MySQL Database . . . 7

Central Redis Cache . . . 8

2.2.2 Signaling and Media Relay . . . 8

Load Balancing of Signalling . . . 8

Message Routing and Media Relay . . . 9

2.2.3 Scaling beyond one Hardware Chassis . . . 10

Scaling the proxy cluster . . . 11

Scaling the proxy cluster . . . 11

Scaling the load balancers . . . 11

Scaling the web servers . . . 11

2.2.4 Architecture for central core and local satellites . . . 11

3 Administrative Configuration **13**

3.1 Creating a Customer . . . 13
3.2 Creating a Subscriber . . . 18
3.3 Domain Preferences . . . 23
3.4 Subscriber Preferences . . . 25
3.5 Creating Peerings . . . 26
3.5.1 Creating Peering Groups . . . 26
3.5.2 Creating Peering Servers . . . 28
3.5.3 Authenticating and Registering against Peering Servers . . . 33
Proxy-Authentication for outbound calls . . . 33
Registering at a Peering Server . . . 36
3.6 Configuring Rewrite Rule Sets . . . 36
3.6.1 Inbound Rewrite Rules for Caller . . . 39
3.6.2 Inbound Rewrite Rules for Callee . . . 41
3.6.3 Outbound Rewrite Rules for Caller . . . 42
3.6.4 Outbound Rewrite Rules for Callee . . . 43
3.6.5 Emergency Number Handling . . . 43
3.6.6 Assigning Rewrite Rule Sets to Domains and Subscribers . . . 44
3.6.7 Creating Dialplans for Peering Servers . . . 45

4 Advanced Subscriber Configuration 46

4.1 Access Control for SIP Calls . . . 46
4.1.1 Block Lists . . . 46
Block Modes . . . 47
Block Lists . . . 47
Block Anonymous Numbers . . . 48
4.1.2 NCOS Levels . . . 48
Creating NCOS Levels . . . 49
Creating Rules per NCOS Level . . . 50
Assigning NCOS Levels to Subscribers/Domains . . . 52

Assigning NCOS Level for Forwarded Calls to Subscribers/Domains . . . 53
4.1.3 IP Address Restriction . . . 53
4.2 Call Forwarding and Call Hunting . . . 54
4.2.1 Setting a simple Call Forward . . . 54
4.2.2 Advanced Call Hunting . . . 55
Configuring Destination Sets . . . 55
Configuring Time Sets . . . 57
4.3 Limiting Subscriber Preferences via Subscriber Profiles . . . 58
4.3.1 Subscriber Profile Sets . . . 58
4.4 Voicemail System . . . 60
4.4.1 Accessing the IVR Menu . . . 60
Mapping numbers and codes to IVR access . . . 60
External IVR access . . . 61
4.4.2 IVR Menu Structure . . . 61
4.5 Configuring Subscriber IVR Language . . . 62
4.6 Sound Sets . . . 63
4.6.1 Configuring Early Reject Sound Sets . . . 64
4.7 Conference System . . . 68
4.7.1 Configuring Call Forward to Conference . . . 68
4.7.2 Configuring Conference Sound Sets . . . 69
4.7.3 Entering the Conference with a PIN . . . 70

5 Customer Self-Care Interfaces 71

5.1 The Customer Self-Care Web Interface . . . 71
5.1.1 Login Procedure . . . 71
5.1.2 Site Customization . . . 71
5.2 The Vertical Service Code Interface . . . 71
5.3 The Voicemail Interface . . . 72

6 Billing Configuration 74

6.1 Billing Data Import . . . 74
6.1.1 Creating Billing Profiles . . . 74
6.1.2 Creating Billing Fees . . . 76
6.1.3 Creating Off-Peak Times . . . 78
6.1.4 Fraud Detection and Locking . . . 79
6.2 Billing Data Export . . . 80
6.2.1 File Name Format . . . 80
6.2.2 File Format . . . 81
File Header Format . . . 81
File Body Format for Call Detail Records (CDR) . . . 81
File Body Format for Event Detail Records (EDR) . . . 85
File Trailer Format . . . 87
6.2.3 File Transfer . . . 88

7 Invoices and invoice templates 89

7.1 Invoices management . . . 89
7.2 Invoice templates . . . 91
7.2.1 Invoice Templates management . . . 91
7.2.2 Invoice Template content . . . 92
Layers . . . 93
Edit SVG XML source . . . 95
Change logo image . . . 97
7.2.3 Save and preview invoice template content. . . . 98
7.3 Invoices generation . . . 100

8 Email templates 103

8.1 Email events . . . 103
8.2 Initial template values and template variables . . . 103
8.3 Password reset email template . . . 103
8.4 New subscriber notification email template . . . 104

8.5 Invoice email template . . . 104
8.6 Email templates management . . . 106

9 Provisioning interfaces 109

9.1 REST API . . . 109
9.2 SOAP and XMLRPC API . . . 109

10 Configuration Framework 111

10.1 Configuration templates . . . 111
10.1.1 .tt2 and .customtt.tt2 files . . . 111
10.1.2 .prebuild and .postbuild files . . . 112
10.1.3 .services files . . . 113
10.2 config.yml, constants.yml and network.yml files . . . 114
10.3 ngcpcfg and its command line options . . . 114
10.3.1 apply . . . 114
10.3.2 build . . . 114
10.3.3 commit . . . 114
10.3.4 decrypt . . . 115
10.3.5 diff . . . 115
10.3.6 encrypt . . . 115
10.3.7 help . . . 115
10.3.8 initialise . . . 115
10.3.9 pull . . . 115
10.3.10 push . . . 115
10.3.11 services . . . 115
10.3.12 status . . . 116

11 Network Configuration 117

11.1 General Structure . . . 117
11.2 Available Host Options . . . 117

12 Advanced Network Configuration **118**

12.1 Extra SIP Sockets . . . 119

12.2 Extra SIP and RTP Sockets . . . 119

13 Security and Maintenance **123**

13.1 Sipwise SSH access to sip:carrier . . . 123

13.2 Firewalling . . . 123

13.3 Password management . . . 124

13.4 SSL certificates. . . . 125

13.5 sip:carrier Backup . . . 126

13.5.1 What data to back up . . . 126

13.5.2 The built-in backup solution . . . 126

13.6 Recovery . . . 127

13.7 Reset Database . . . 127

13.8 System requirements and performance . . . 127

13.9 Troubleshooting . . . 130

14 Monitoring and Alerting **132**

14.1 Internal Monitoring . . . 132

14.2 Statistics Dashboard . . . 132

14.3 External Monitoring Using SNMP . . . 132

14.3.1 Overview and Initial Setup . . . 132

14.3.2 Details . . . 133

A Cloud PBX **138**

A.1 Configuring the Device Management . . . 138

A.1.1 Setting up Device Models . . . 139

A.1.2 Uploading Device Firmwares . . . 142

A.1.3 Creating Device Configurations . . . 143

A.1.4 Creating Device Profiles . . . 145

A.2 Preparing PBX Rewrite Rules . . . 146
A.2.1 Inbound Rewrite Rules for Caller . . . 147
A.2.2 Inbound Rewrite Rules for Callee . . . 147
A.2.3 Outbound Rewrite Rules for Caller . . . 149
A.3 Creating Customers and Pilot Subscribers . . . 150
A.3.1 Creating a PBX Customer . . . 150
A.3.2 Creating a PBX Pilot Subscriber . . . 154
A.4 Managing a Customer PBX . . . 160
A.4.1 Creating more Subscribers . . . 161
A.4.2 Assigning Subscribers to Devices . . . 166
Synchronizing a PBX Device for initial Usage . . . 169
A.4.3 Configuring Sound Sets for the Customer PBX . . . 172
Uploading a Music-on-Hold File . . . 173
Uploading Auto-Attendant Sound Files . . . 174
A.4.4 Configuring the Auto Attendant . . . 175
Preparing the Sound Set . . . 175
Configuring the Auto Attendant Slots . . . 176
Activating the Auto Attendant . . . 176
A.5 Device Auto-Provisioning Security . . . 177
A.5.1 Server Certificate Authentication . . . 177
A.5.2 Client Certificate Authentication . . . 179
A.6 Device Bootstrap and Resync Workflows . . . 179
A.6.1 Cisco SPA Device Bootstrap . . . 180
Initial Bootstrapping . . . 180
Subsequent Device Resyncs . . . 180
A.6.2 Panasonic Device Bootstrap . . . 182
Initial Bootstrapping . . . 182
Factory Reset . . . 182

Subsequent Device Resyncs . . . 183
A.6.3 Yealink Device Bootstrap . . . 183
Initial Bootstrapping . . . 183
Factory Enable Yealink Auto-Provisioning . . . 184
Subsequent Device Resyncs . . . 184

B Sipwise Clients and Apps **185**

B.1 sip:phone Mobile App . . . 185
B.1.1 Zero Config Launcher . . . 185
3rd Party Sign-Up Form . . . 186
3rd Party Launch Handler . . . 189
B.1.2 Mobile Push Notification . . . 189
Architecture . . . 190
Configuring the Push Daemon . . . 191

C NGCP configs overview **192**

C.1 config.yml overview . . . 192
C.1.1 asterisk . . . 192
C.1.2 backuptools . . . 193
C.1.3 bootenv . . . 194
C.1.4 cdrexport . . . 195
C.1.5 checktools . . . 195
C.1.6 cleanuptools . . . 197
C.1.7 database . . . 198
C.1.8 faxserver . . . 198
C.1.9 general . . . 199
C.1.10 haproxy . . . 199
C.1.11 heartbeat . . . 199
C.1.12 intercept . . . 200
C.1.13 kamailio . . . 200

C.1.14 mediator . . . 203
C.1.15 nginx . . . 204
C.1.16 ntp . . . 204
C.1.17 ossbss . . . 204
C.1.18 pbx (only with additional cloud PBX module installed) . . . 206
C.1.19 prosody . . . 206
C.1.20 pushd . . . 206
C.1.21 qos . . . 207
C.1.22 rate-o-mat . . . 207
C.1.23 redis . . . 208
C.1.24 reminder . . . 208
C.1.25 rsyslog . . . 209
C.1.26 rtpproxy . . . 209
C.1.27 security . . . 210
C.1.28 sems . . . 210
C.1.29 sshd . . . 211
C.1.30 voisniff . . . 212
C.1.31 www_admin . . . 212
C.1.32 www_csc . . . 214

1 Introduction

1.1 About this Document

This document describes the architecture and the operational steps to install, operate and modify the Sipwise sip:carrier.

In the various chapters, it describes the system architecture, the installation and upgrade procedures and the initial configuration steps to get your first users online. It then dives into advanced preference configurations like rewrite rules, call blockings, call forwards etc.

There is a description of the customer self-care interface, how to configure the billing system and how to provision the system via the provided APIs.

Finally it describes the internal configuration framework, the network configuration and gives hints about tweaking the system for security and performance.

1.2 Getting Help

1.2.1 Phone Support

Depending on your support contract, you are eligible to contact our Support Team by phone either in business hours or around the clock. Business hours refer to the UTC+1 time zone (Europe/Vienna). Please check your support contract to check the type of support you've purchased.

Before calling our Support Team, please also open a ticket in our Ticket System and provide as much detail as you can for us to understand the problems, fix them and investigate the root cause. Please provide the ticket number assigned to your newly created ticket when asked by our support personnel on the phone.

Phone numbers, Ticket System URL and account information can be found in your support contract. Please make this information available to the persons in your company maintaining the sip:carrier.

1.2.2 Ticket System

Depending on your support contract, you can create either a limited or an unlimited amount of support tickets on our Web based Ticket System. Please provide as much information as possible when opening a ticket, especially the following:

- **WHAT** is affected (e.g. the whole system is unreachable or customers can't register or place calls)
- **WHO** is affected (e.g. all customers, only parts of it, and **WHICH** parts - only customers in a specific domain or customers with specific devices etc)
- **WHEN** did the problem occur (time frames, or after the firmware of specific devices types have been updated etc)

Our Support Team will ask further questions via the Ticket System along the way of troubleshooting your issue. Please provide the information as soon as possible in order to solve your issue in a timely manner.

1.3 What is the sip:carrier?

The sip:carrier is a SIP based Open Source Class5 VoIP soft-switch platform providing rich telephony services. It offers a wide range of features to end users (call forwards, voicemail, conferencing, call blocking, click-to-dial, call-lists showing near-realtime accounting information etc.), which can be configured by them using the customer-self-care web interface. For operators, it offers a fully web-based administrative panel, allowing them to configure users, peerings, billing profiles etc., as well as viewing real-time statistics of the system. For tight integration into existing infrastructures, it provides a powerful REST API.

The sip:carrier comes pre-installed on six+ servers in one+ IBM Flex Chassis, see Section 2. Apart from your product specific configuration, there is no initial configuration or installation to be done to get started.

1.4 What is inside the sip:carrier?

Opposed to other free VoIP software, the sip:carrier is not a single application, but a whole software platform, the Sipwise NGCP (Sipwise Next Generation Communication Platform), which is based on Debian GNU/Linux.

Using a highly modular design approach, the NGCP leverages popular open-source software like MySQL, NGINX, Catalyst, Kamailio, SEMS, Asterisk etc. as its core building blocks. These blocks are glued together using optimized and proven configurations and work-flows and are complemented by building blocks developed by Sipwise to provide fully-featured and easy to operate VoIP services.

The installed applications are managed by the NGCP Configuration Framework, which allows to change system parameters in a single place, so administrators don't need to have any knowledge of the dozens of different configuration files of the different packages. This provides a very easy and bullet-proof way of operating, changing and tweaking the otherwise quite complex system.

Once configured, integrated web interfaces are provided for both end users and administrators to use the sip:carrier. By using the provided provisioning and billing APIs, it can be integrated tightly into existing OSS/BSS infrastructures to optimize work-flows.

1.5 Who should use the sip:carrier?

The sip:carrier is specifically tailored to companies who want to provide fully-featured SIP based VoIP service without having to go through the steep learning curve of SIP signalling, integrating the different building blocks to make them work together in a reasonable way. The sip:carrier is already deployed all around the world by all kinds of VoIP operators, using it as Class5 soft-switch, as Class4 termination platform or even as Session Border Controller with all kinds of access networks, like Cable, DSL, WiFi and Mobile networks.

2 System Architecture

2.1 Hardware Architecture

The sip:carrier starts with a minimum deployment of 50.000 subscribers, requiring one chassis with two web servers, two db servers, two loadbalancers and two proxies. A fully deployed sip:carrier for 250.000 subscribers fills the chassis up with 14 servers, containing two web servers, two db servers, two loadbalancers and 8 proxies.

Figure 1: Hardware setup for single chassis

The system is based on an IBM Flex Chassis taking up rack space of 10U with 14 computing nodes based on IBM x220 servers.

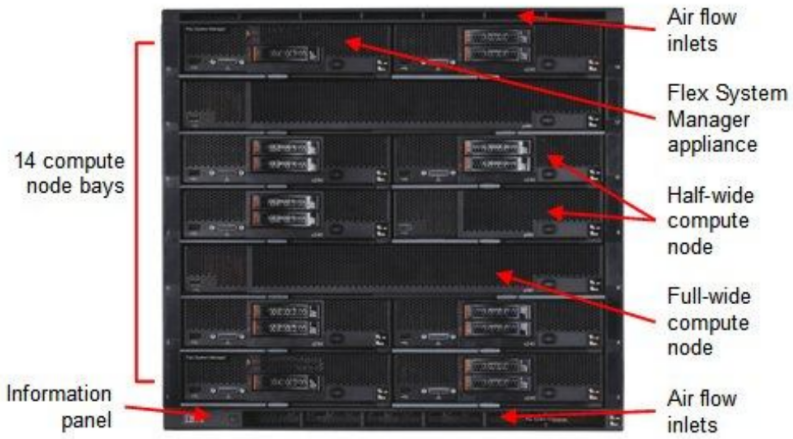


Figure 2: Chassis front view

All nodes are equipped equally with two hard disks in Raid-1 mode.

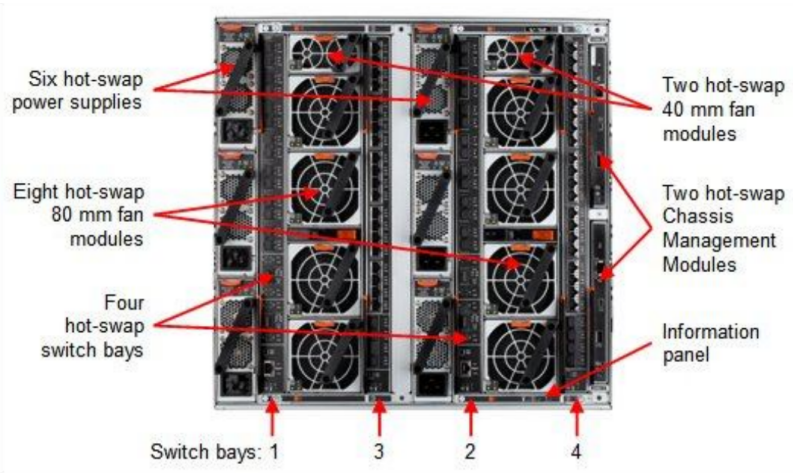


Figure 3: Chassis back view

The power supply is designed fully redundant in an N+N fashion with N=3, for example to feed 3 PSUs with normal power and 3 PSUs with UPS power.

Figure 4: Chassis switch module

Each chassis is equipped with two EN2092 Gigabit Ethernet switches providing 10 GbE uplinks each. Four 10GbE uplinks are optional and need to be licensed separately if needed.

2.2 Component Architecture

The sip:carrier is composed by a cluster of four different node types, which are all deployed in active/standby pairs:

- **Web-Servers** (web1a/web1b): Provide northbound interfaces (CSC, API) via HTTPS for provisioning
- **DB-Servers** (db1a/db1b): Provide the central persistent SQL data store for customer data, peering configuration, billing data etc.
- **Proxy-Servers** (proxy1a/proxy1b .. proxy4a/proxy4b): Provide the SIP and XMPP signalling engines, application servers and media relays to route Calls and IM/Presence and serve media to the endpoints.

- **Load-Balancers** (lb1a/lb1b): Provide a perimeter for SIP and XMPP signalling.

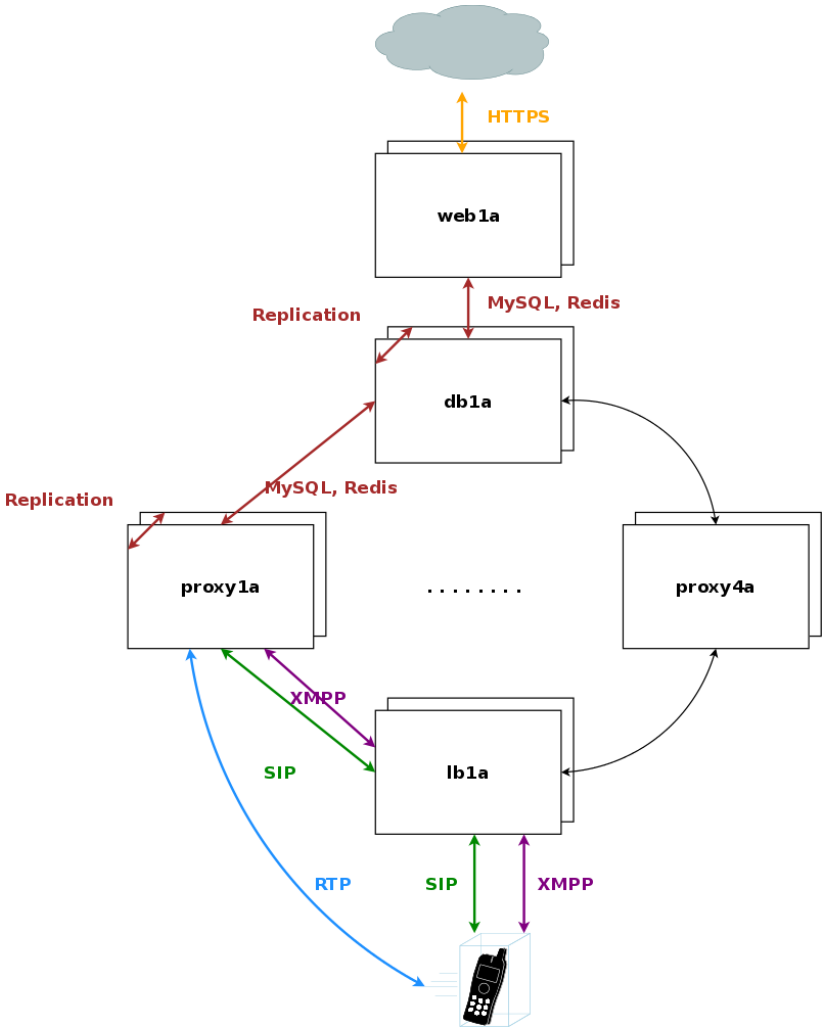


Figure 5: Architecture Overview

The system is provisioned via the web servers on a central pair of db servers. Signalling is entering the system via the lb servers to a cluster of proxies, which in turn communicate directly (caching and shared data) and indirectly (static provisioning data replicated via master/slave) with the db servers. Each pair of proxy is capable of handling any subscriber, so subscribers are not bound to specific "home proxies". Once a call starts on a proxy pair, it is ensured that the full range of services is provided on that pair (voicemail, media, billing, . . .) until call-teardown. Failures on an active proxy node cause a fail-over to the corresponding stand-by node within the proxy pair, taking over the full signalling and media without interruptions.

2.2.1 Provisioning

Any HTTPS traffic for provisioning (web interfaces, northbound APIs) but also for phone auto-provisioning enters the platform on the active web server. The web server runs an nginx instance acting as a reverse proxy for the ngcp-panel process, which in turn provides the provisioning functionality.

The web server is connected to the db server pair, which provides a persistent relational data store via MySQL and a high-performance system cache using Redis key-value store.

API and Web Interface

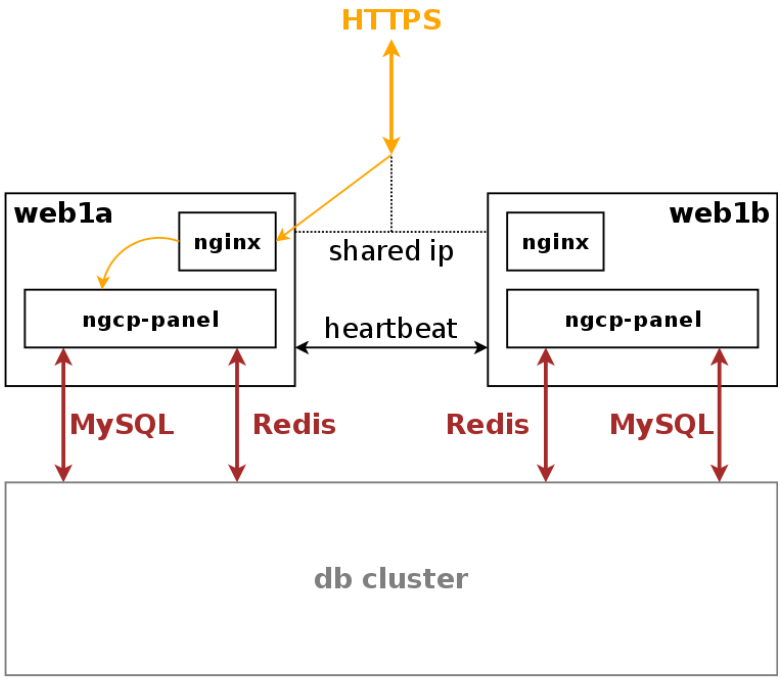


Figure 6: Web Server Overview

The web server pair is an active/standby pair of nodes connected via heartbeat. If one of the servers fail (by losing connection to the outside while the standby server is still connected, or caused by a hardware failure, or if it's down due to maintenance), the standby server takes over the shared IP address of the active node and continues serving the provisioning interface.

Provisioning Database

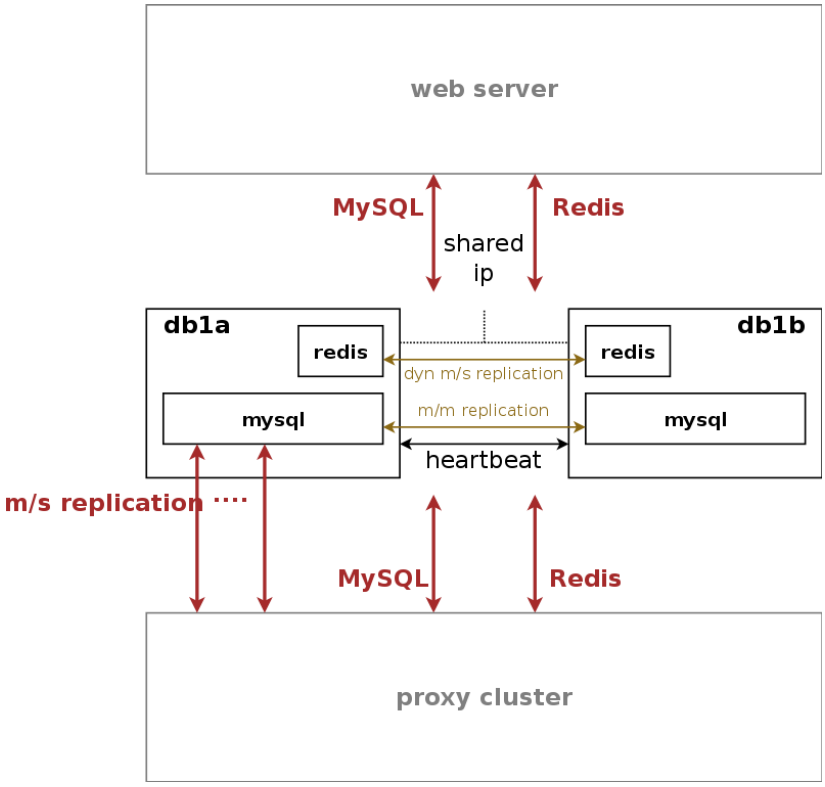


Figure 7: DB Server Overview

The db server pair is another active/standby pair with automatic fail-over. Nodes in the pair are running a MySQL master/master replication with replication integrity checks to ensure data redundancy and safety. Any changes via provisioning interfaces are stored in the MySQL cluster. The second service is a redis master/slave replication with automatic master propagation on fail-over. This redis cluster is used as a high-performance volatile system cache for various components which need to share state information across nodes.

Persistent MySQL Database

The MySQL instances on the db nodes syncronize via row-based master/master replication. In theory, any of the two servers in the pair can be used to write data to the database, however in practice a shared IP is used towards clients accessing the service, so only one node will receive the write requests. This is done to ensure transparent and instant convergence of the db cluster on fail-over for the clients.

On top of that, the first node of the db pair also acts as a master in a master/slave replication towards all proxy nodes in the system. That way, proxies can access read-only provisioning data directly from their local databases, resulting in reduced latency and significant off-loading of read queries on the central db cluster.

Central Redis Cache

A redis master/slave setup is used to provide a high-perfomance key/value storage for global system data shared across proxies. This includes concurrent call counters for customers and subscribers, as a subscriber could place two simultaneous calls via two different proxy pairs.

2.2.2 Signaling and Media Relay

Any signalling traffic enters and leaves the system via load balancers, which act as a perimeter towards the customer devices and performs NAT handling, DoS and DDoS mitigation. New connections are routed to a random pair of proxy servers, which do the actual routing for SIP and XMPP. The proxy servers also engage media relays for voice and video streams, which bypass the load balancers and communicate directly with the customer devices for performance reasons.

Load Balancing of Signalling

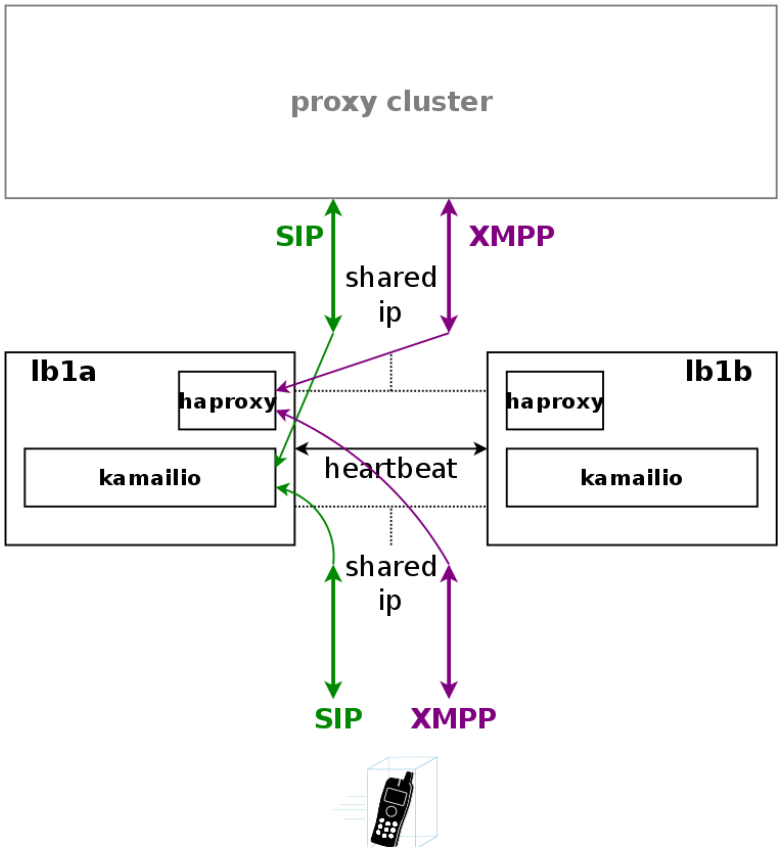


Figure 8: Load Balancer Overview

A node in a load balancer pair runs two services besides the usual heartbeat.

One is a state-less instance of kamailio, providing an extremely fast relay of SIP messages. Kamailio takes care of converting

TCP and TLS connections from the customer devices to UDP for internal communication towards proxies, and it performs far-end NAT traversal by inspecting the SIP messages and comparing it to the actual source address where packets have been received from, then modifying the SIP messages accordingly. If a SIP message is received by the load balancer, it distinguishes between new and ongoing SIP transactions by inspecting the To-Tags of a message, and it determines whether the message is part of an established dialog by inspecting the Route header. Sanity checks are performed on the headers to make sure the call flows adhere to certain rules for not being able to bypass any required element in the routing path. In-dialog messages are routed to the corresponding proxy servers according to the Route defined in the message. Messages initiating a new transaction and/or dialog (registrations, calls etc) are routed to a randomly selected proxy. The selection algorithm is based on a hash over the Call-ID of the message, so the same proxy sending a authentication challenge to an endpoint will receive the authenticated message again.

The second service running on a load balancer is haproxy, which is acting as load balancing instance for XMPP messages. The same way the SIP load balancer routes SIP messages to the corresponding proxy, the haproxy passes XMPP traffic on to the proxy maintaining a session with a subscriber, or randomly selects a proxy in case of a new connection while automatically failing over on timeouts.

Message Routing and Media Relay

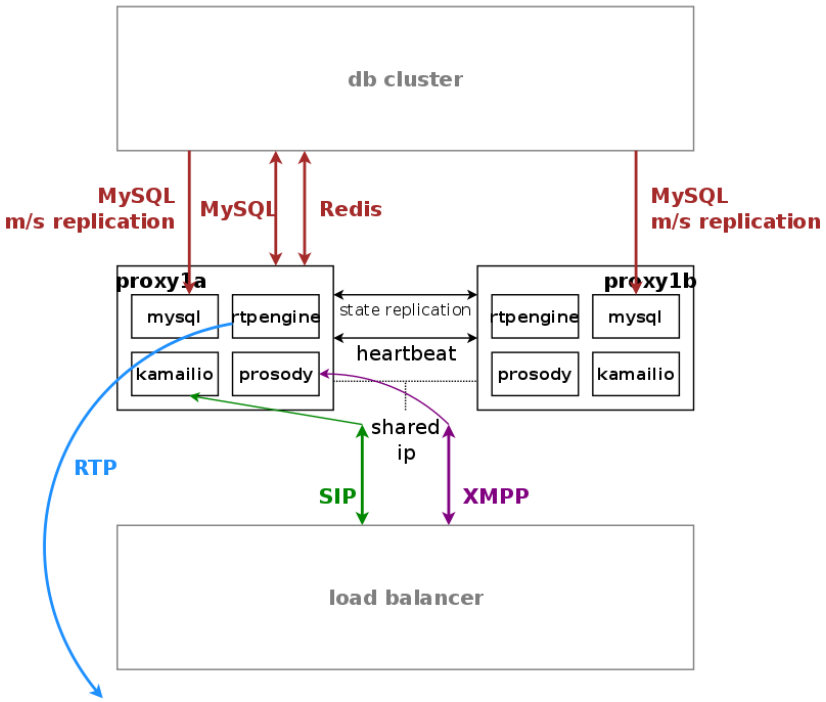


Figure 9: Proxy Server Overview

Proxy servers also come in pairs, and by default there are four pairs of proxies in a standard sip:carrier setup.

The proxies are responsible for doing the actual SIP routing and media handling and the XMPP presence and chat message deliveries. Each proxy pair can handle any subscriber on the overall system, compared to the concept of "home proxies" in other architectures. The advantage of this approach is that the overall system can be scaled extremely easily by adding more proxy pairs without having to redistribute subscribers.

Once a load balancer sends a new message to a proxy, the SIP transaction and/or dialog gets anchored to this proxy. That way it is ensured that a call starting on a proxy is also ended on a proxy, and the full range of feature handling like media relay, voicemail, fax, billing and rating is performed on this proxy to not have a central point for various tasks, potentially leading to a non-scalable bottle-neck. Due to the anchoring, proxies come in pairs and replicate all internal state information via redis to the standby node. In case of fail-over, the full signalling and media is moved to the standby node without interruption.

The complete static subscriber information like authentication credentials, number mappings, feature settings etc. are replicated from the db cluster down to the local MySQL instance of the proxies. The ratio of db read requests of static subscriber data versus reading and writing volatile and shared data is around 15:1, and this approach moves the majority of the static read operations from the central db cluster to the local proxy db.

Volatile and shared information needed by all proxies in the cluster read from and written to the db cluster. This mainly includes SIP registration information and XMPP connection information.

Billing and rating is also performed locally on the proxies, and only completed CDRs (rated or unrated depending on whether rating is enabled) are transferred to the central db cluster for consumption via the northbound interfaces.

For SIP, the relevant instances on a proxy are kamailio acting as a stateful proxy for SIP registration and call routing, sems acting as a back-to-back user-agent for prepaid billing and application server, rtpengine as media relay and RTP/SRTP transcoder, and asterisk as voicemail server. XMPP is handled by an instance of prosody, and several billing processes mediate start and stop records into CDRs and rate them according to the relevant billing profiles.

2.2.3 Scaling beyond one Hardware Chassis

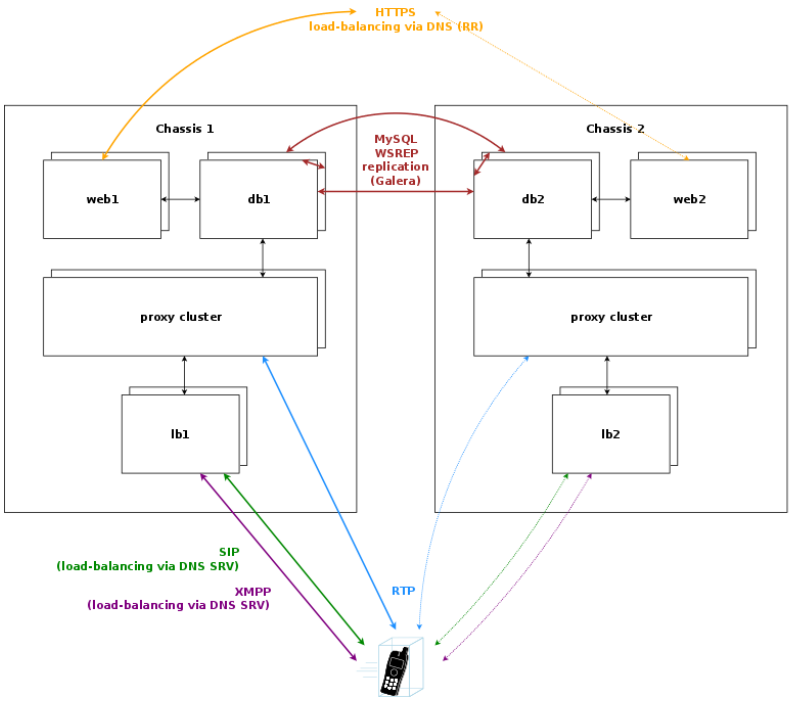


Figure 10: Scaling beyond one chassis

If the sip:carrier is scaled beyond 250.000 subscribers and therefore exceeds one chassis, a second chassis is put into place. This chassis provides another two web servers, two db servers, two load balancers and 8 proxies, doubling the capacity of the system.

Scaling the proxy cluster

The DB cluster is the only node type which requires a notable change on the architecture. Once more than one db pair is deployed, the replication mechanism between db nodes changes from master/master between the nodes of the db1 pair to a synchronous multi-master replication over all db nodes on the system using Galera. This change allows to scale both read and write requests over multiple nodes, while being transparent to all other nodes.

Scaling the proxy cluster

New proxy nodes replicate via master/slave from the first db node in the chassis as usual. Since the db cluster holds all provisioning information of all subscribers, the proxy nodes join the cluster transparently and will start serving subscribers as soon as all services on a new proxy are reachable from the load balancers.

Scaling the load balancers

Loadbalancers are completely stateless, so they start serving subscribers as soon as they are made visible to the subscribers. This could either be done via DNS round-robin, but the better approach is to configure a DNS SRV record, which allows for more fine-grained control like weighting load-balancer pairs and allowing fail-over from one pair to another on the client side.

The loadbalancers use the Path extension of SIP to make sure during SIP registration that calls targeted to a subscriber are routed via the same loadbalancer pair which the subscriber used during registration for proper traversal of symmetric NAT at the customer premise.

A SIP or XMPP request reaching a loadbalancer can be routed to any available proxy in the whole system, or only to proxies belonging to the same chassis as the loadbalancer, depending on the system configuration.

Scaling the web servers

New web server pairs are made available to web clients via DNS round-robin. Any pair of web servers can be used to read or write provisioning information via the web interfaces or the API.

2.2.4 Architecture for central core and local satellites

Tip

this architecture is not part of the standard deployment and is to be defined in the project plan!

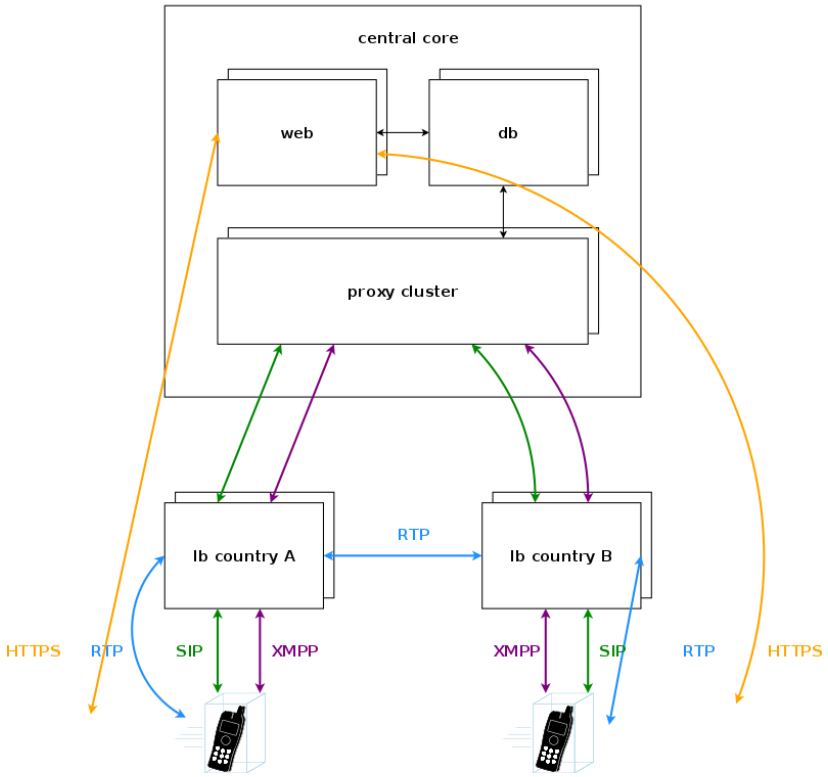


Figure 11: Central core with local breakouts

In case of a geographically distributed system spanning across multiple countries, different regulatory requirements have to be met for signalling and media, especially when it comes to if, where and how subscriber traffic can be intercepted. Countries might have the requirement to intercept traffic in the country, so the signalling and media must be anchored to an element in the country. Also if a media stream stays within a country, it is preferred to keep the media as close to the subscribers as possible to reduce latency, so relaying streams via a central core has to be avoided.

For this scenario, the sip:carrier allows to move the loadbalancers directly into the countries. DNS settings for subscribers within the country ensure that they will always contact those loadbalancers, either using separate DNS settings per country for a SIP domain, or using GeoIP mechanisms in DNS to return the closest loadbalancer based on the location of the subscriber. To anchor media to the countries, the rtpengine instances are moved from the proxies to the loadbalancers and are controlled via the stateless kamailio instances on the loadbalancers instead of the kamailio instances on the proxies.

3 Administrative Configuration

To be able to configure your first test clients, you will need a Customer, a SIP domain and some subscribers in this domain. Throughout this steps, let's assume you're running the NGCP on the IP address *1.2.3.4*, and you want this IP to be used as SIP domain. This means that your subscribers will have an URI like *user1@1.2.3.4*.

Tip

You can of course set up a DNS name for your IP address (e.g. letting sip.yourdomain.com point to 1.2.3.4) and use this DNS name throughout the next steps, but we'll keep it simple and stick directly with the IP as a SIP domain for now.

Warning

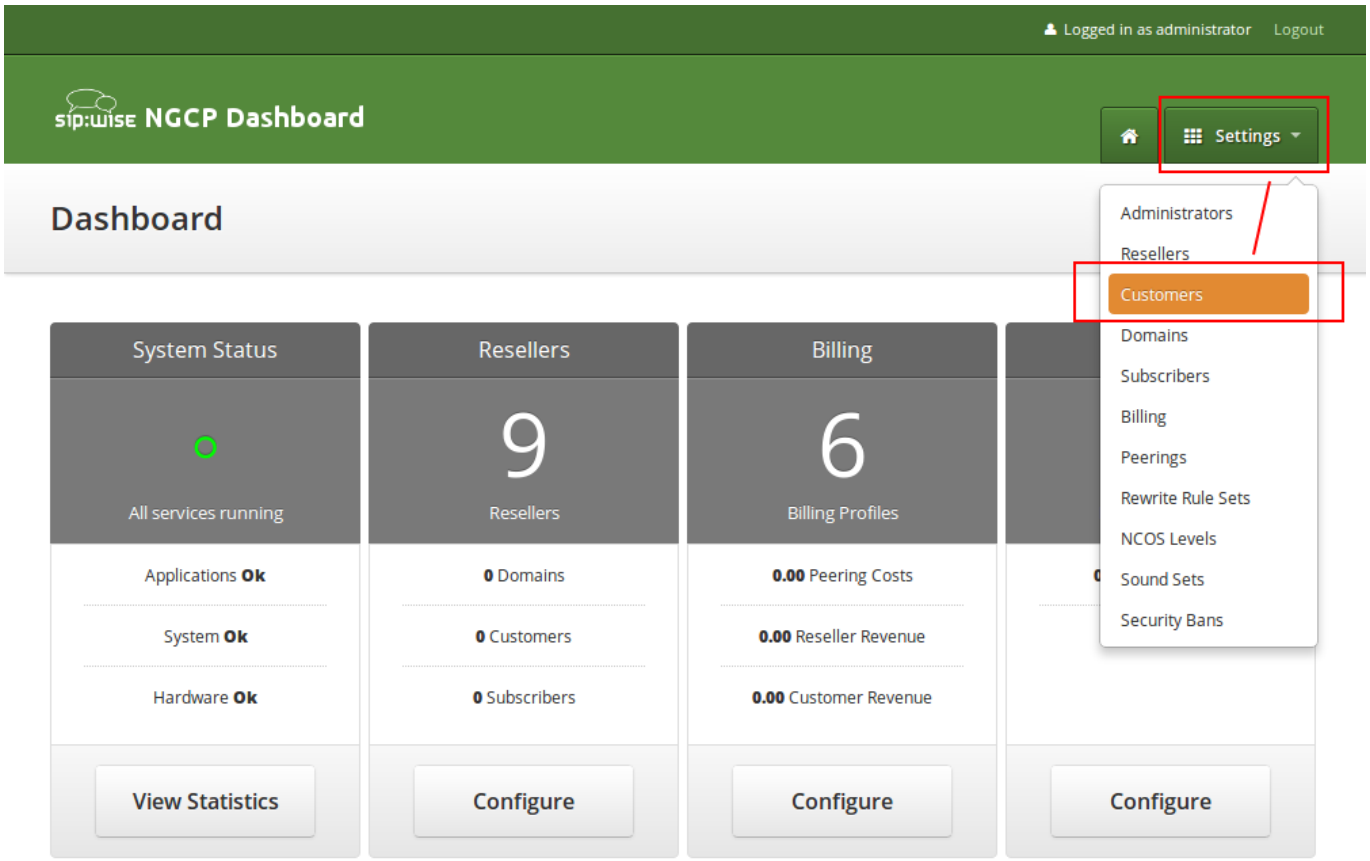
Once you started adding subscribers to a SIP domain, and later decide to change the domain, e.g. from 1.2.3.4 to sip.yourdomain.com, you'll need to recreate all your subscribers in this new domain. It's currently not possible to easily change the domain part of a subscriber.

Go to the *Administrative Web Panel* (*Admin Panel*) running on *https://<ce-ip>:1443/login/admin* and follow the steps below. The default user on the system is *administrator* with the password *administrator*, if you haven't changed it already.

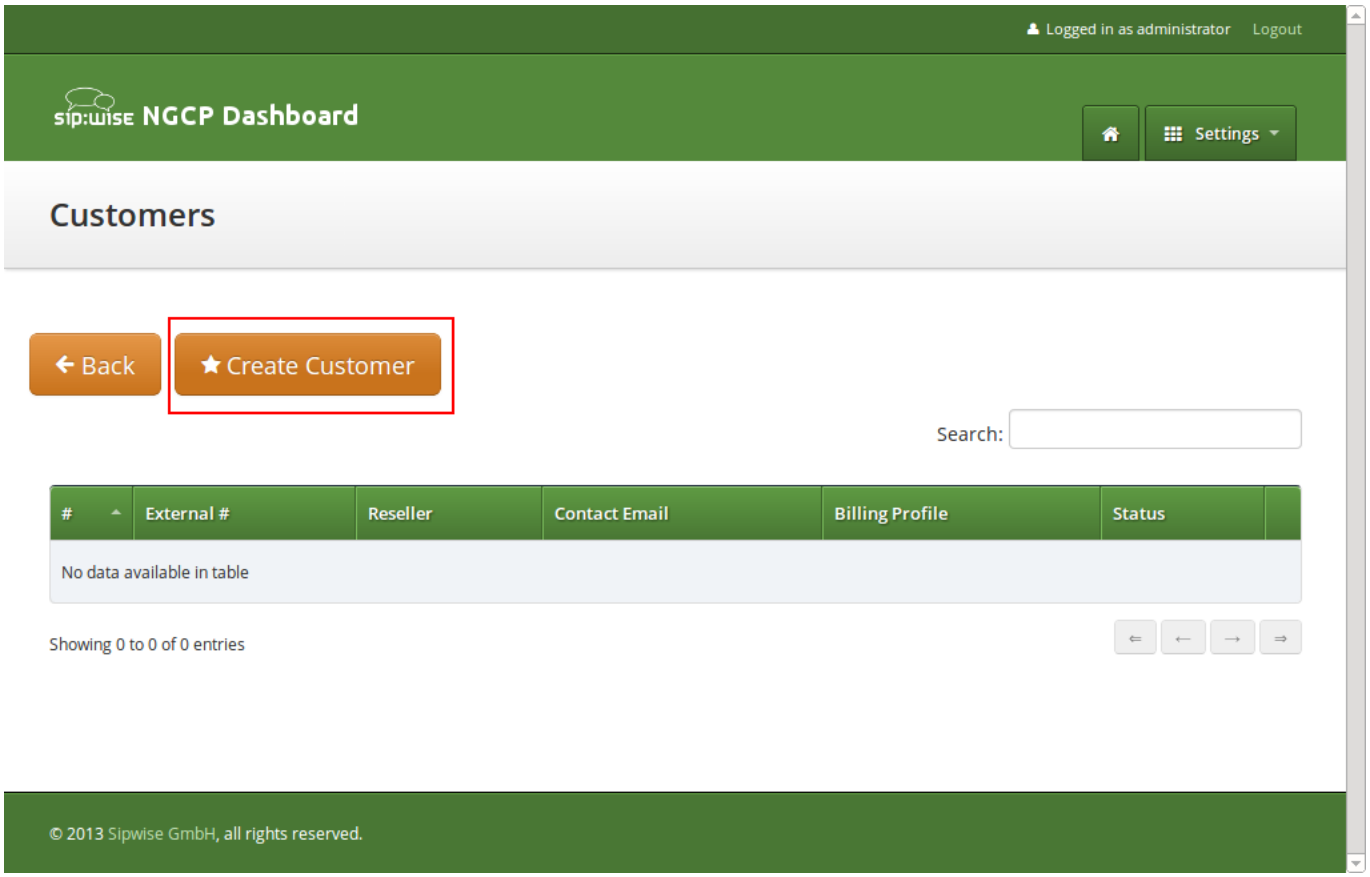
3.1 Creating a Customer

A Customer is a special type of contract on the system acting as billing container for SIP subscribers. You can create as many SIP subscribers within a Customer as you want.

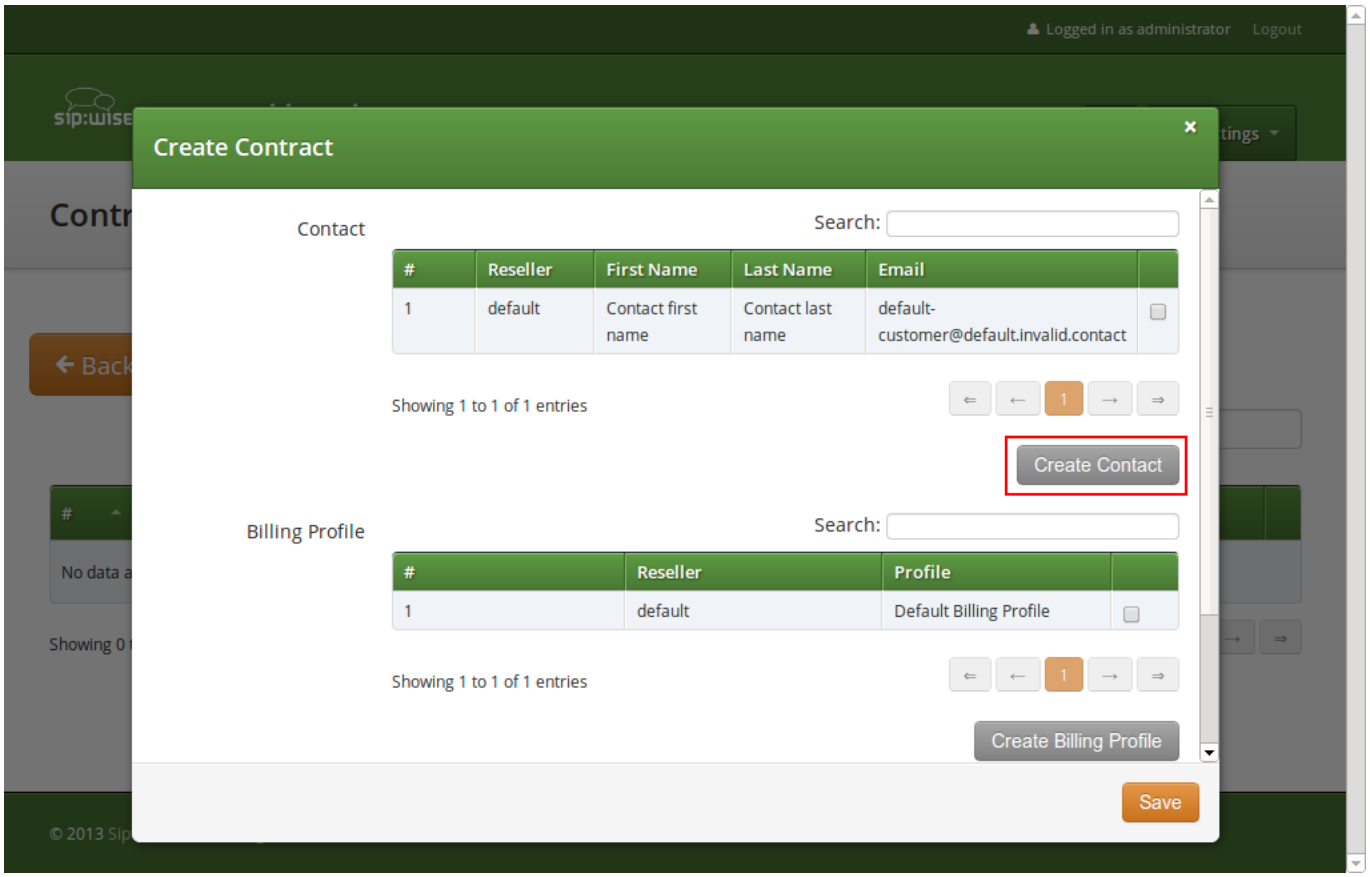
To create a Customer, got to *Settings→Customers*.



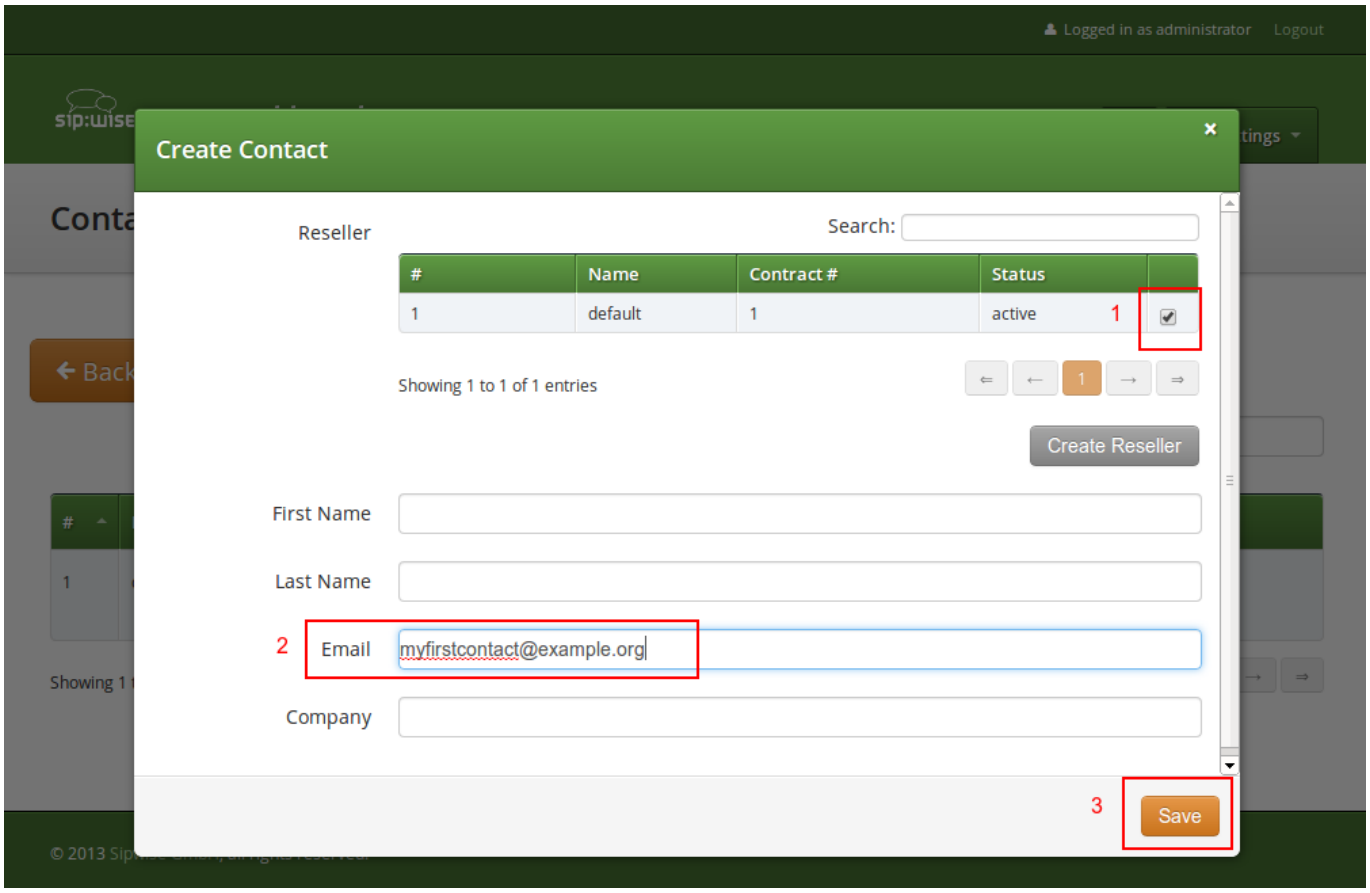
Click on *Create Customer*.



Each *Customer* needs a *Contact*. We can either reuse the default one, but for a clean setup, we create a new *Contact* for each *Customer* to be able to identify the *Customer*. Click on *Create Contact* to create a new *Contact*.

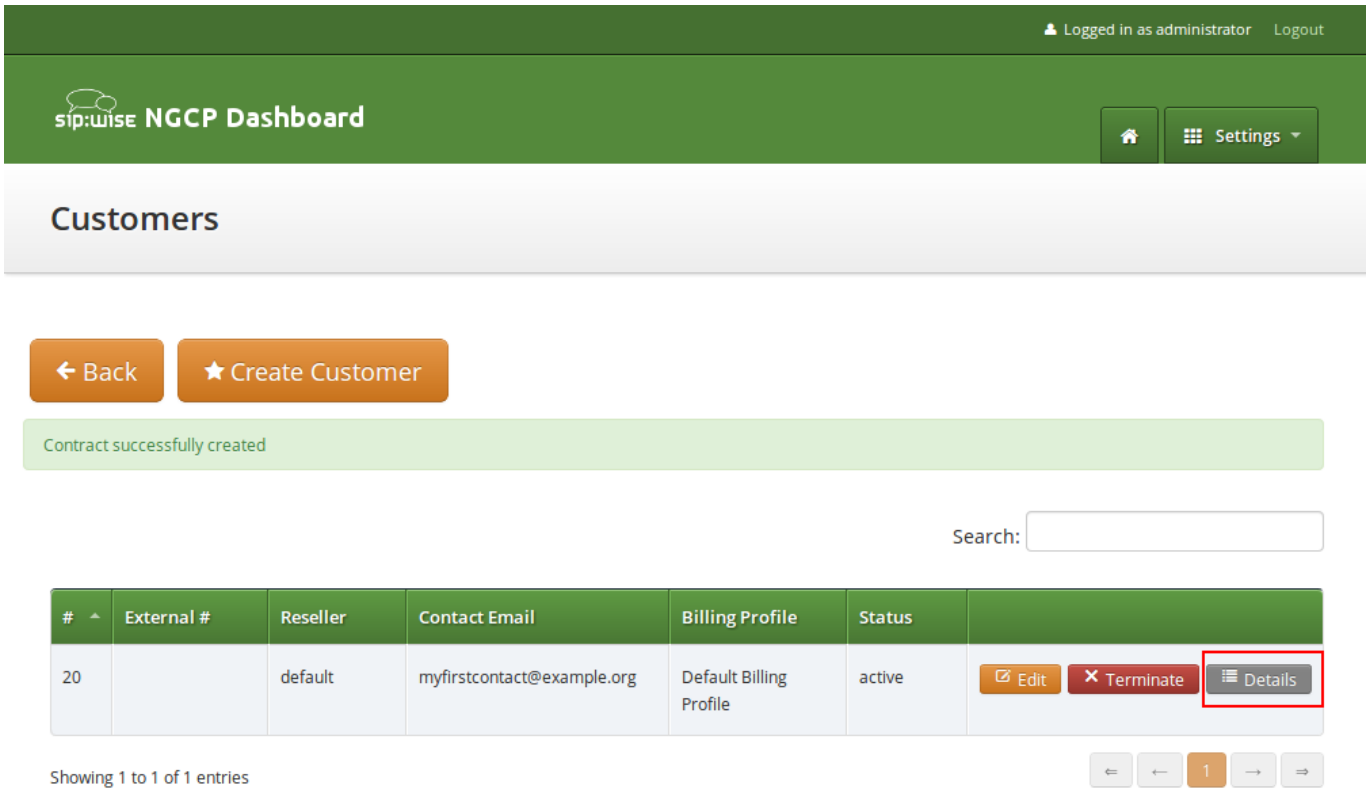


We assign the Contact to the default *Reseller*. You can create a new one if you want, but for a simple setup the default *Reseller* is sufficient. Select the *Reseller* and enter the contact details (at least an *Email* is required), then press *Save*.



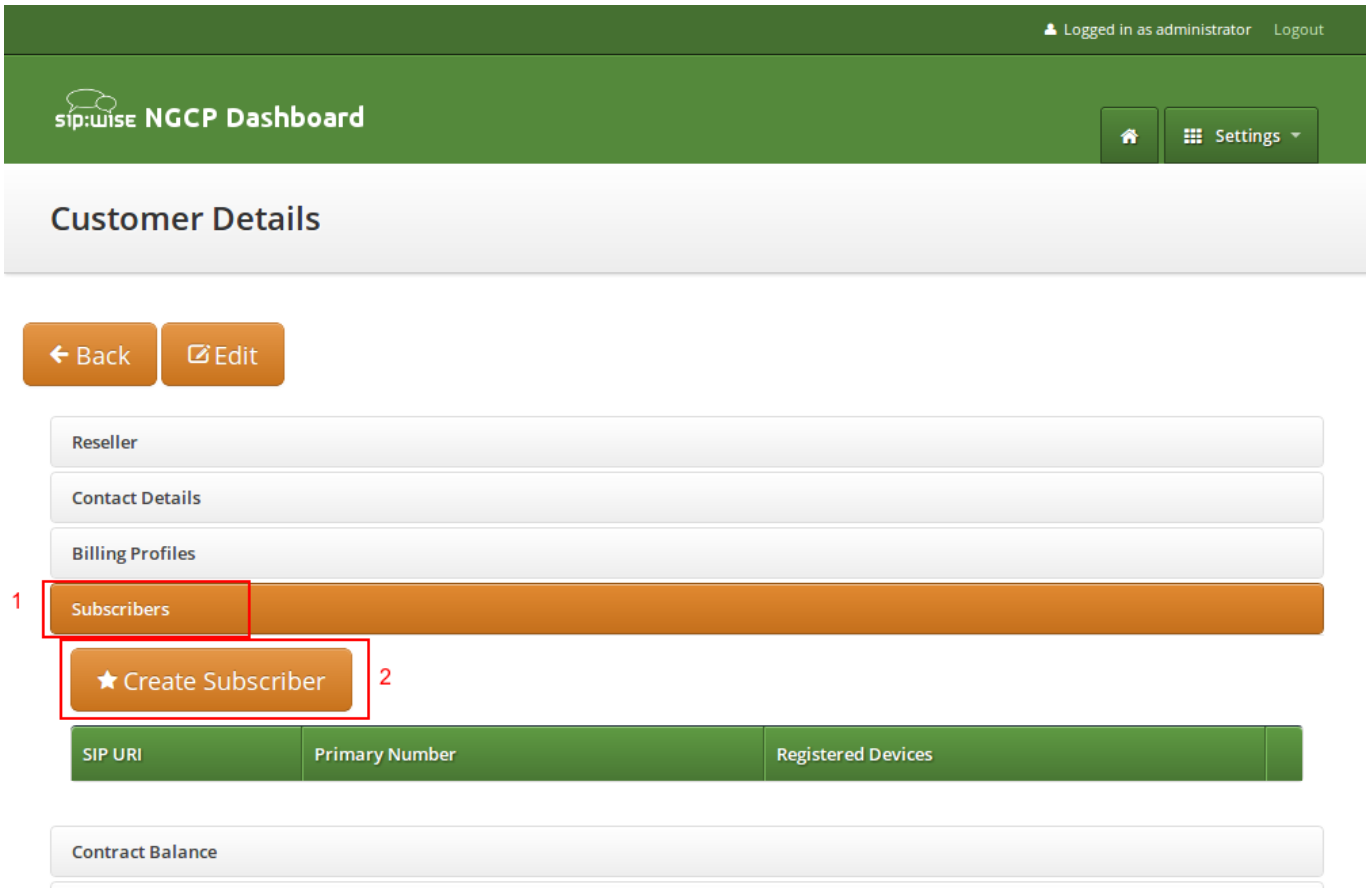
You will be redirected back to the *Contract* form. The newly created *Contact* is selected by default now, so you only have to select a *Billing Profile*. Again you can create a new one on the fly, but we will go with the default profile for now. Select it and press *Save*.

You will now see your first *Customer* in the list. Hover over the customer and click *Details* to view the details.

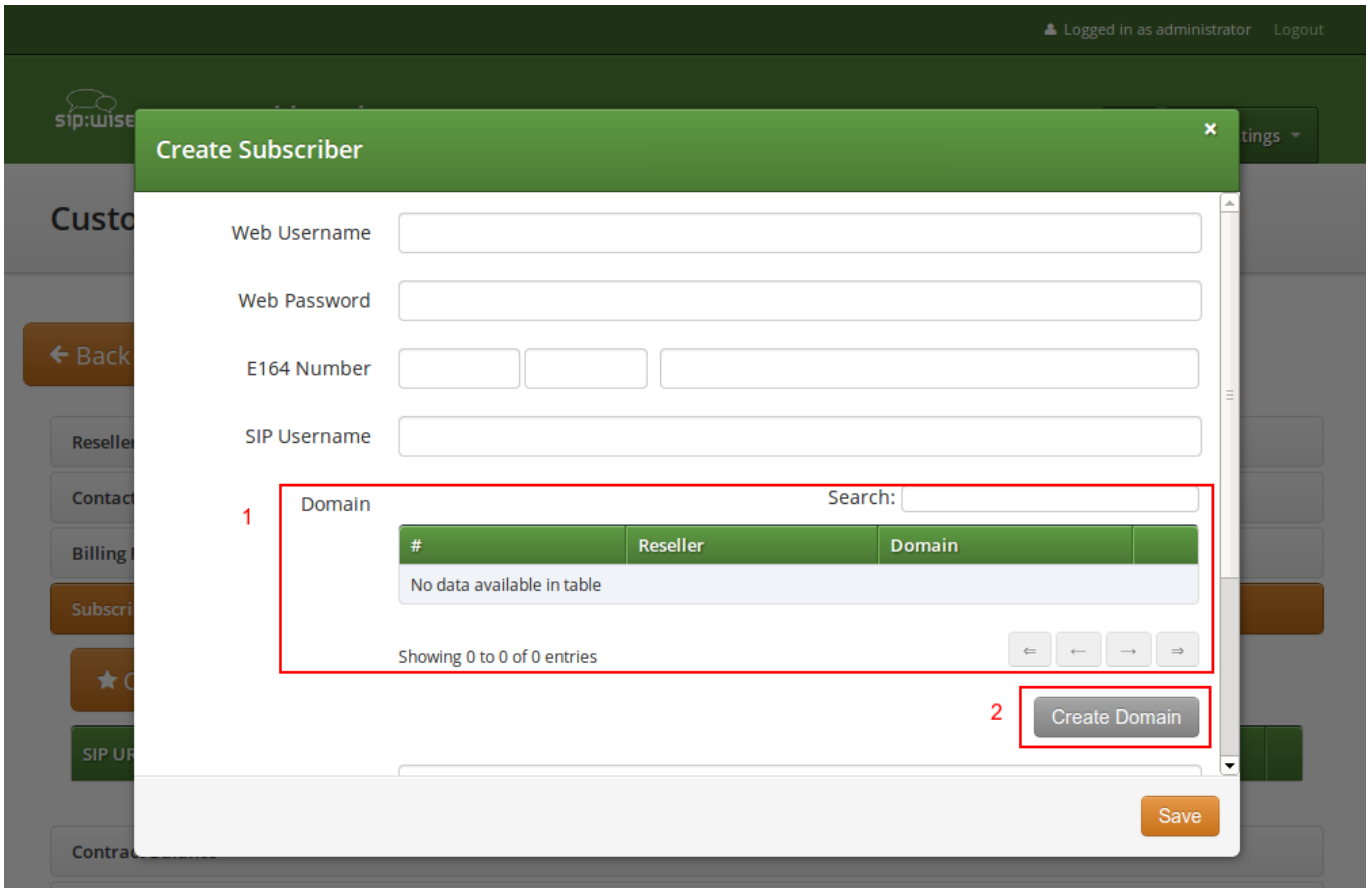


3.2 Creating a Subscriber

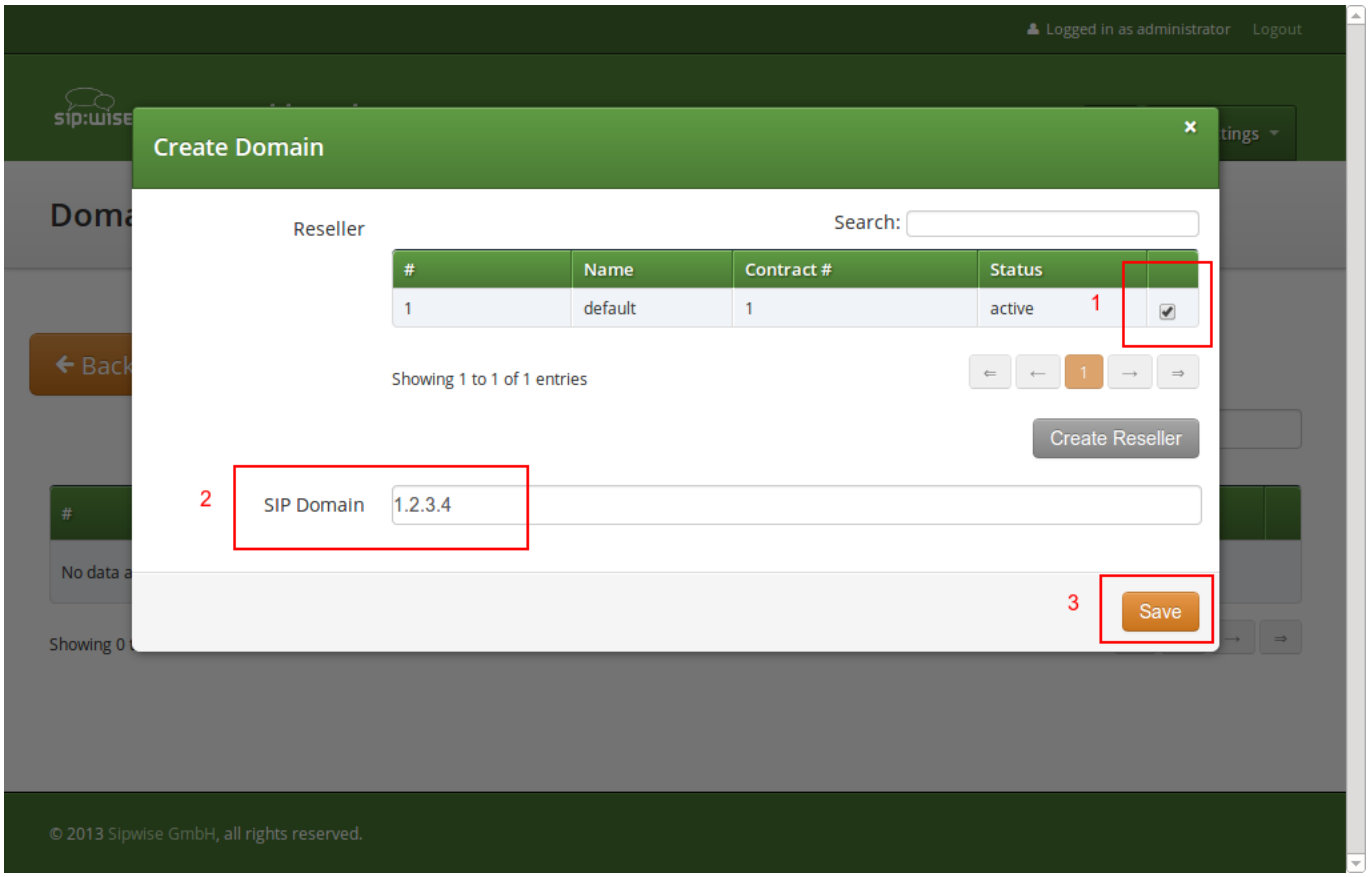
In your *Customer* details view, click on the *Subscribers* row, then click the *Create Subscriber*.



As you can see, we don't have any *SIP Domains* yet, so click on *Create Domain* to create one.



Select the *Reseller* (make sure to use the same reseller where your *Customer* is created in) and enter your domain name, then press *Save*.



Your *Domain* will be preselected now, so fill out the rest of the form:

- **Web Username**: This is the user part of the username the subscriber may use to log into her *Customer Self Care Interface*. The user part will be automatically suffixed by the SIP domain you choose for the **SIP URI**. Usually the web username is identical to the **SIP URI**, but you may choose a different naming schema.

Caution

The web username needs to be unique. The system will return a fault if you try to use the same web username twice.

- **Web Password**: This is the password for the subscriber to log into her *Customer Self Care Interface*. It must be at least 6 characters long.
- **E164 Number**: This is the telephone number mapped to the subscriber, separated into *Country Code (CC)*, *Area Code (AC)* and *Subscriber Number (SN)*. For the first tests, you can set a made-up number here and change it later when you get number blocks assigned by your PSTN interconnect partner. So in our example, we'll use *43* as CC, *99* as AC and *1001* as SN to form the phantasy number *+43 99 1001*.

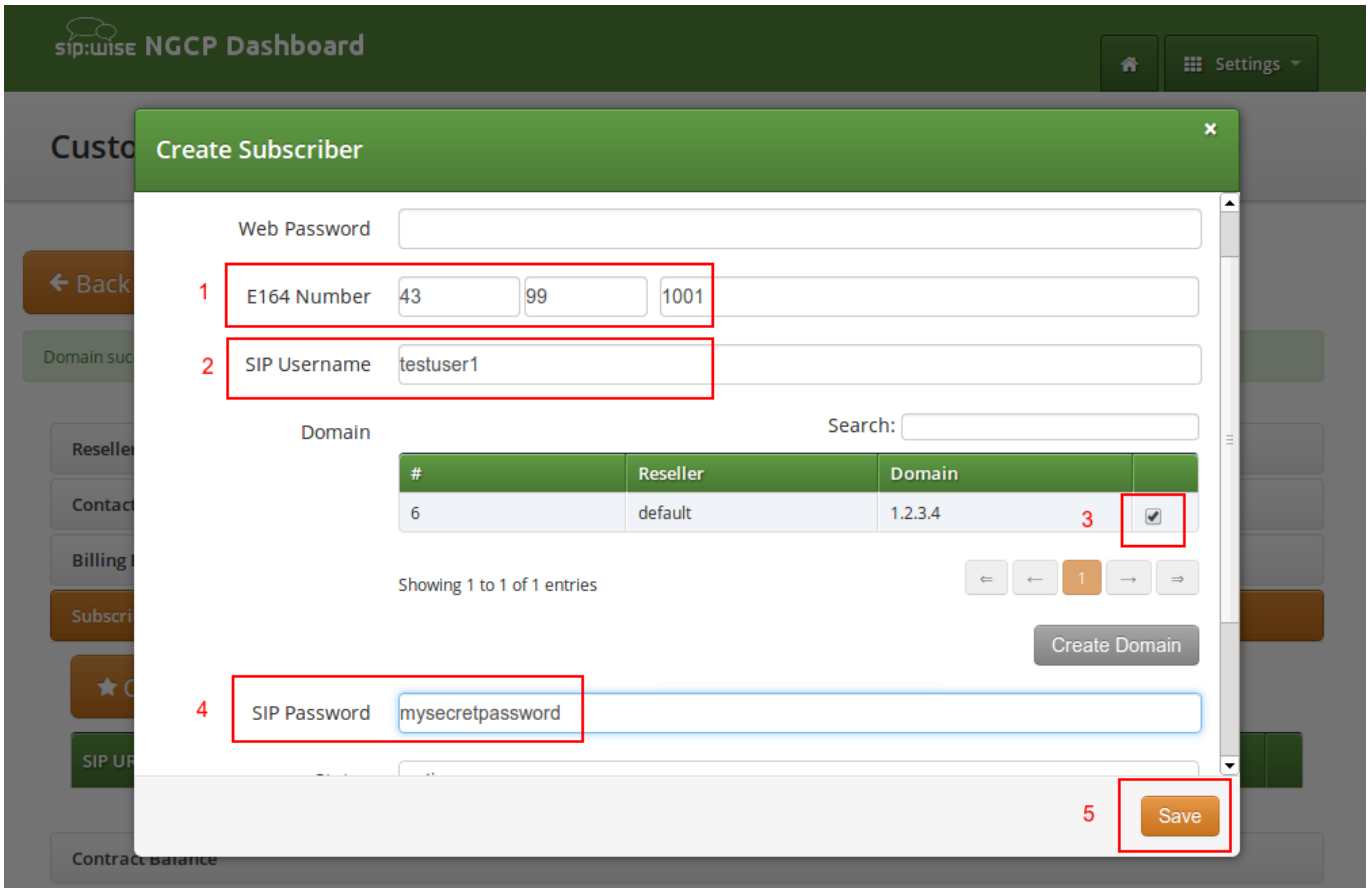
Tip

This number can actually be used to place calls between local subscribers, even if you don't have any PSTN interconnection. This comes in handy if you use phones instead of soft-clients for your tests. The format in which this number can be dialled so the subscriber is reached is defined in Section 3.6.

Important

NGCP allows single subscriber to have multiple E.164 numbers to be used as aliases for receiving incoming calls. Also NGCP supports "implicit" extensions, e.g. if a subscriber has number 012345, but somebody calls 012345100, then it first tries to send the call to number 012345100 (even though the user is registered as myusername), and only after 404 it falls back to the user-part for which the user is registered.

- **SIP Username**: The user part of the SIP URI for your subscriber.
- **SIP Domain**: The domain part of the SIP URI for your subscriber.
- **SIP Password**: The password of your subscriber to authenticate on the SIP proxy. It must be at least 6 characters long.
- **Status**: You can lock a subscriber here, but for creating one, you will most certainly want to use *active*.
- **External ID**: You can provision an arbitrary string here (e.g. an ID of a 3rd party provisioning/billing system).
- **Administrative**: If you have multiple subscribers in one account and set this option for one of them, this subscriber can administrate other subscribers via the *Customer Self Care Interface*.



Repeat the creation of *Customers* and *Subscribers* for all your test accounts. You should have at least 3 subscribers to test all the functionality of the NGCP.

Tip

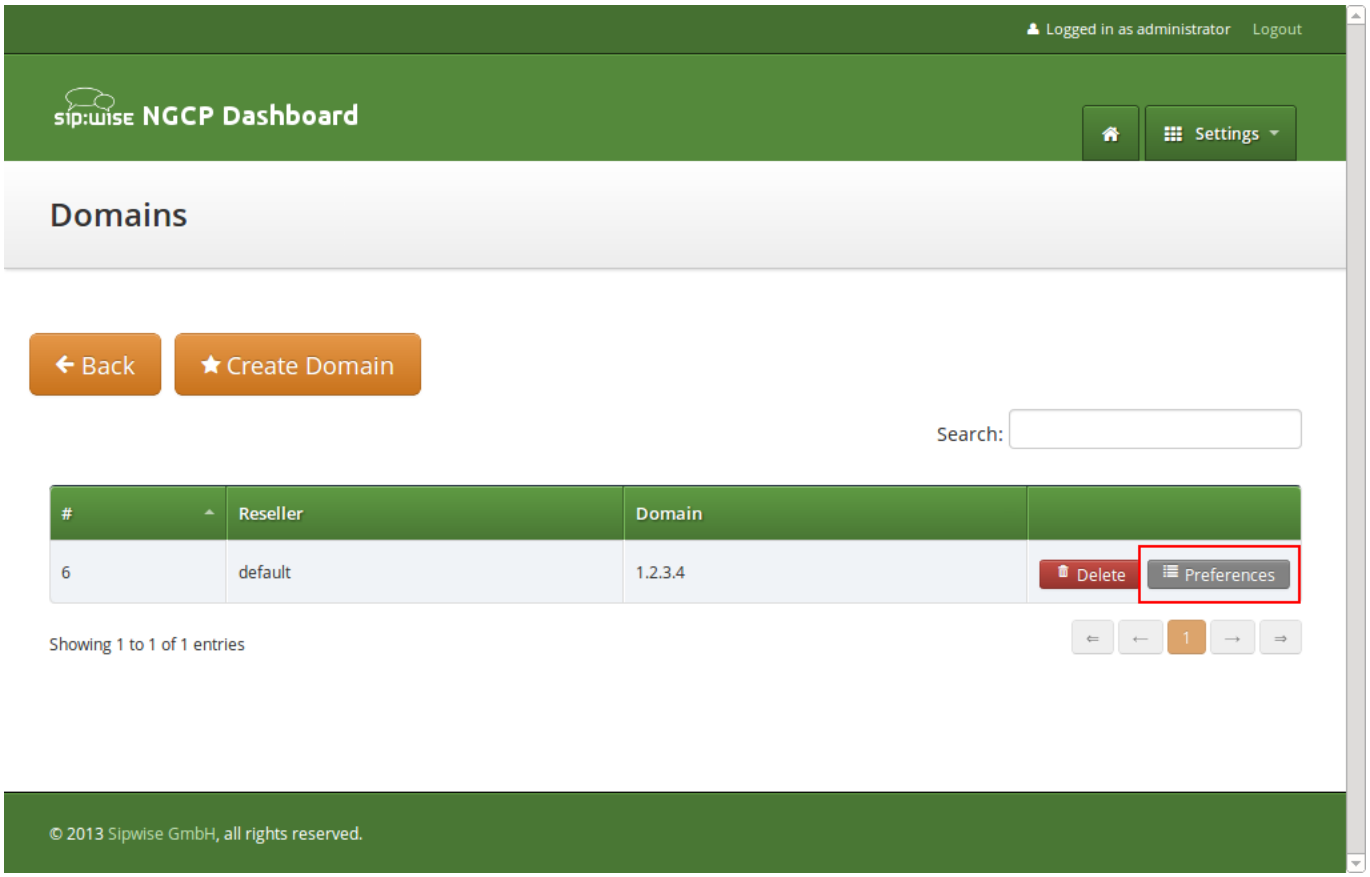
At this point, you're able to register your subscribers to the NGCP and place calls between these subscribers.

You should now revise the *Domain* and *Subscriber* Preferences.

3.3 Domain Preferences

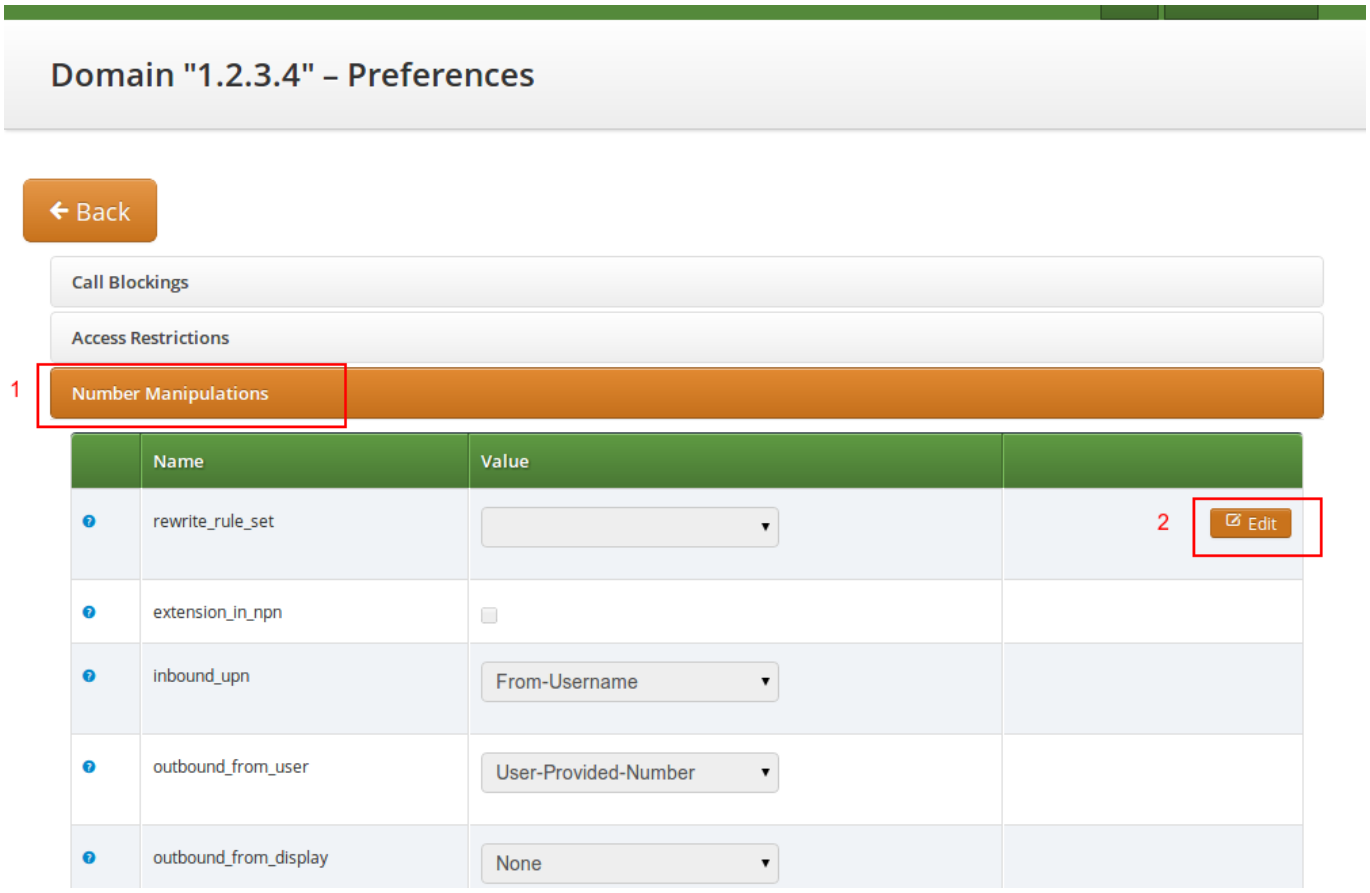
The *Domain Preferences* are the default settings for *Subscriber Preferences*, so you should set proper values there if you don't want to configure each subscriber separately. You can later override these settings in the *Subscriber Preferences* if particular subscribers need special settings.

To configure your *Domain Preferences*, go to *Settings→Domains* and click on the *Preferences* button of the domain you want to configure.

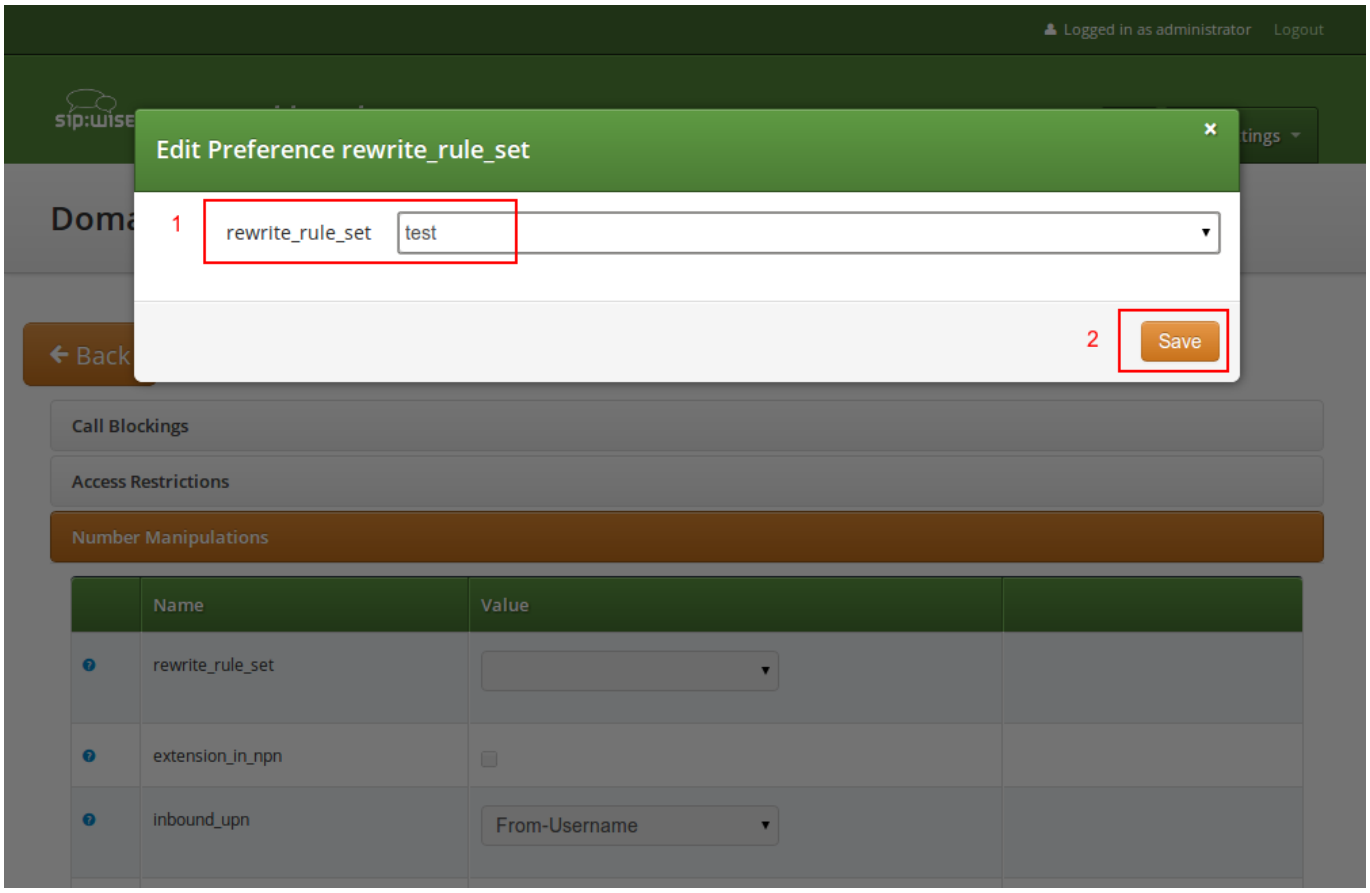


The most important settings are in the group *Number Manipulations*, where you can configure where from a SIP message to take numbers from for incoming messages, where in the SIP messages to put which numbers for outgoing SIP messages, and how these numbers are normalized to E164 format and vice versa.

To assign a *Rewrite Rule Set* to a *Domain*, create a set first as described in Section 3.6, then assign it to the domain by editing the *rewrite_rule_set* preference.



Select the *Rewrite Rule Set* and press *Save*.



Then, select the field you want the *User Provided Number* to be taken from for inbound INVITE messages. Usually the *From-Username* should be fine, but you can also take it from the *Display-Name* of the From-Header, and other options are available as well.

3.4 Subscriber Preferences

You can override the *Domain Preferences* on a subscriber basis as well. Also, there are *Subscriber Preferences* which don't have a default value in the *Domain Preferences*.

To configure your *Subscriber*, go to *Settings→Subscribers* and click *Details* on the row of your subscriber. There, click on the *Preferences* button on top.

You want to look into the *Number Manipulations* and *Access Restrictions* options in particular, which control what is used as user-provided and network-provided calling numbers.

- For outgoing calls, you may define multiple numbers or patterns to control what a subscriber is allowed to send as user-provided calling numbers using the *allowed_clis* preference.
- If *allowed_clis* does not match the number sent by the subscriber, then the number configured in *cli* (the network-provided number) preference will be used as user-provided calling number also.
- You can override any user-provided number coming from the subscriber using the *user_cli* preference.

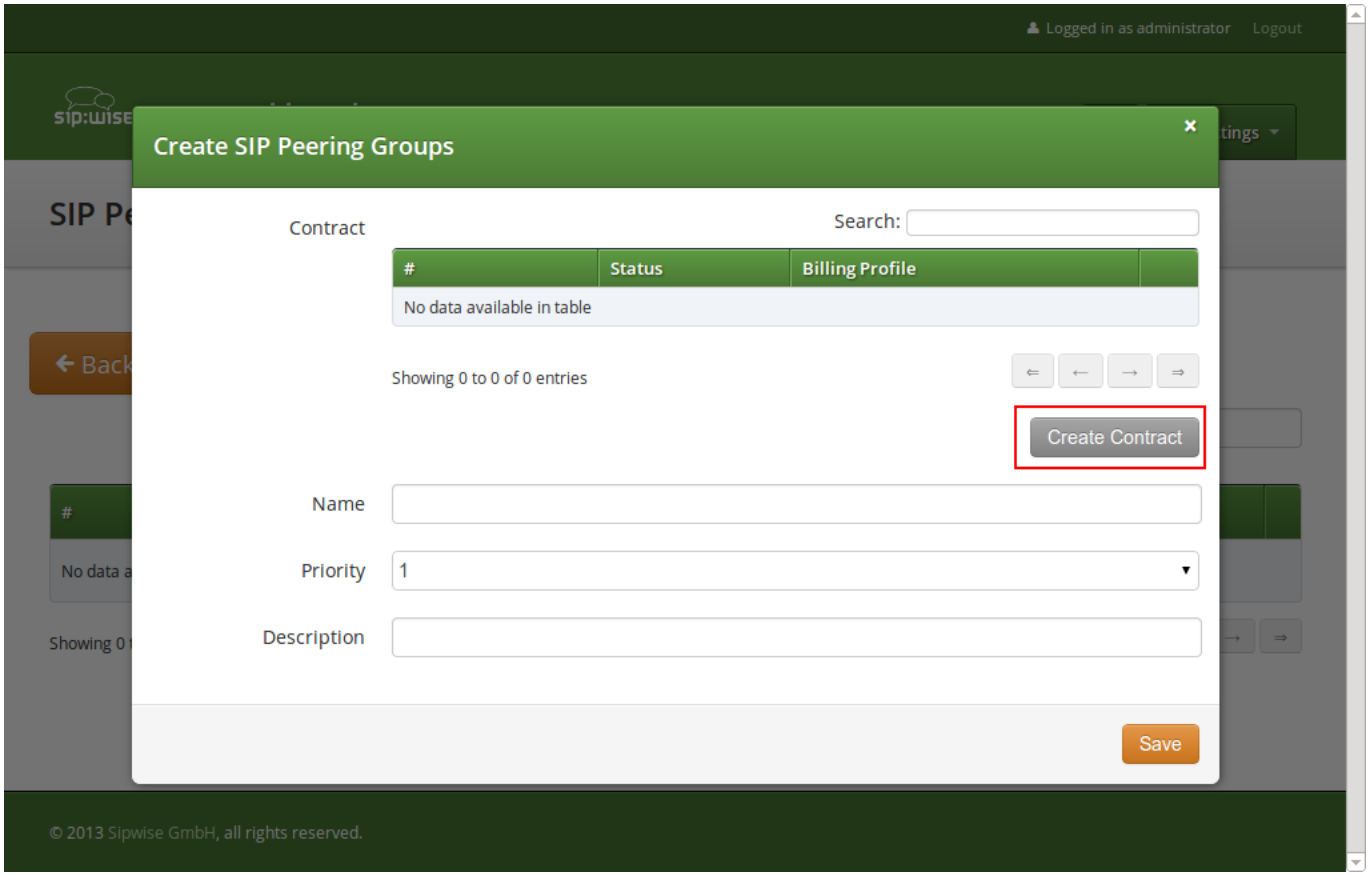
3.5 Creating Peerings

If you want to terminate calls at or allow calls from 3rd party systems (e.g. PSTN gateways, SIP trunks), you need to create SIP peerings for that. To do so, go to *Settings→Peerings*. There you can add peering groups, and for each peering group add peering servers and rules controlling which calls are routed over these groups. Every peering group needs a peering contract for correct interconnection billing.

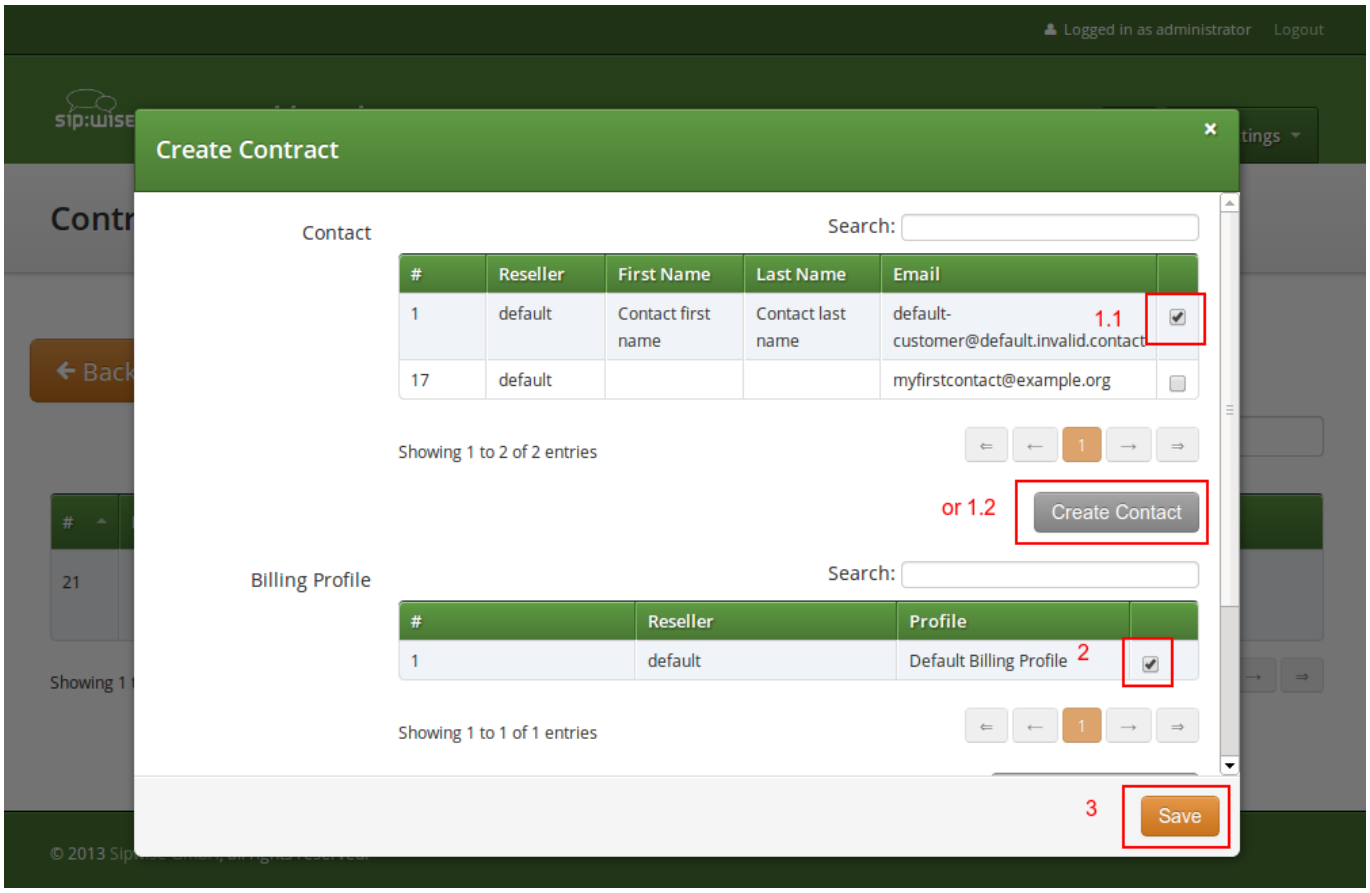
3.5.1 Creating Peering Groups

Click on *Create Peering Group* to create a new group.

In order to create a group, you must select a peering contract. You will most likely want to create one contract per peering group.

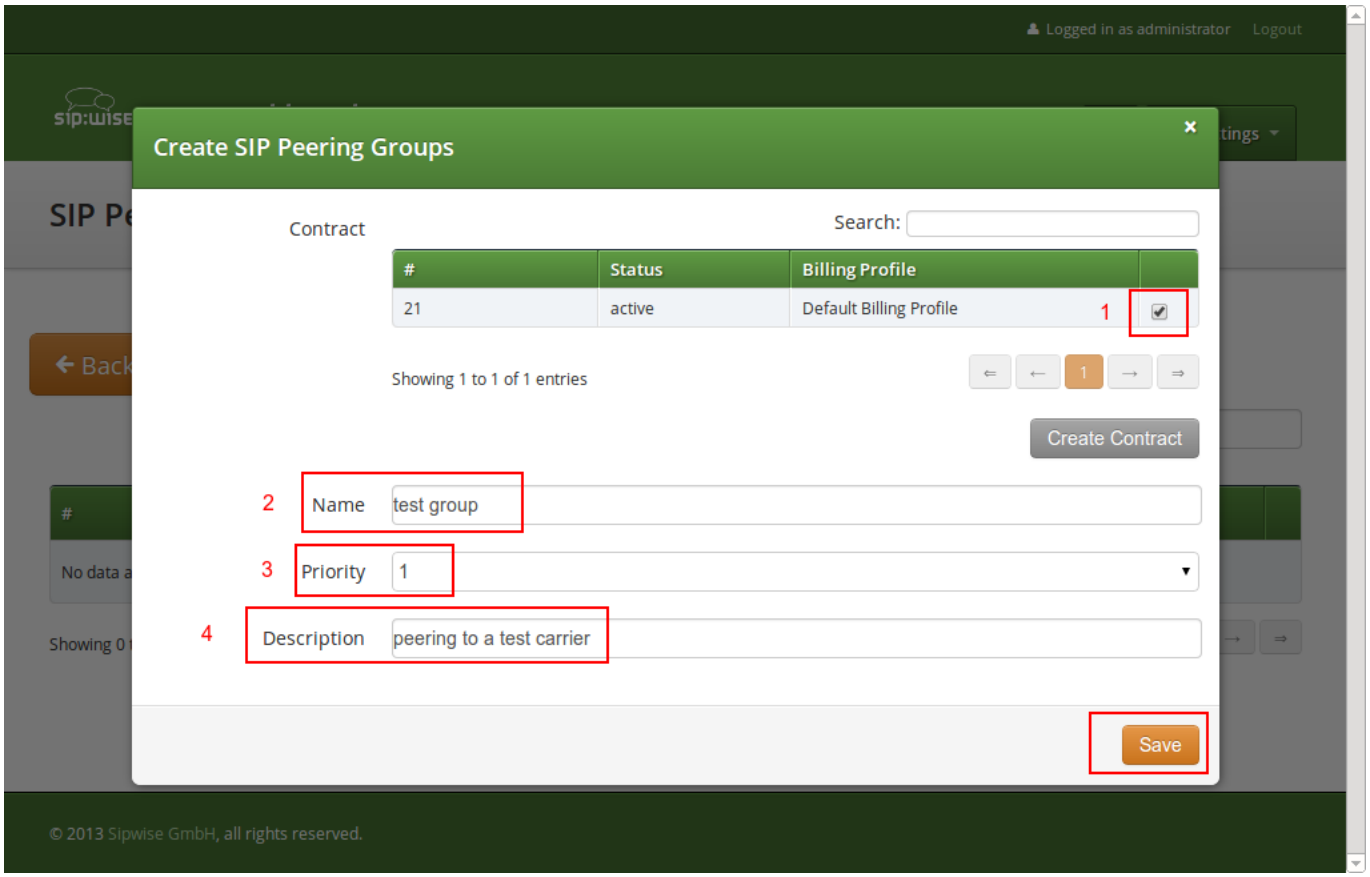


Click on *Create Contract* create a *Contact*, then select a *Billing Profile*.



Click *Save* on the *Contacts* form, and you will get redirected back to the form for creating the actual *Peering Group*. Put a name, priority and description there, for example:

- **Peering Contract:** select the id of the contract created before
- **Name:** `test group`
- **Priority:** `1`
- **Description:** `peering to a test carrier`



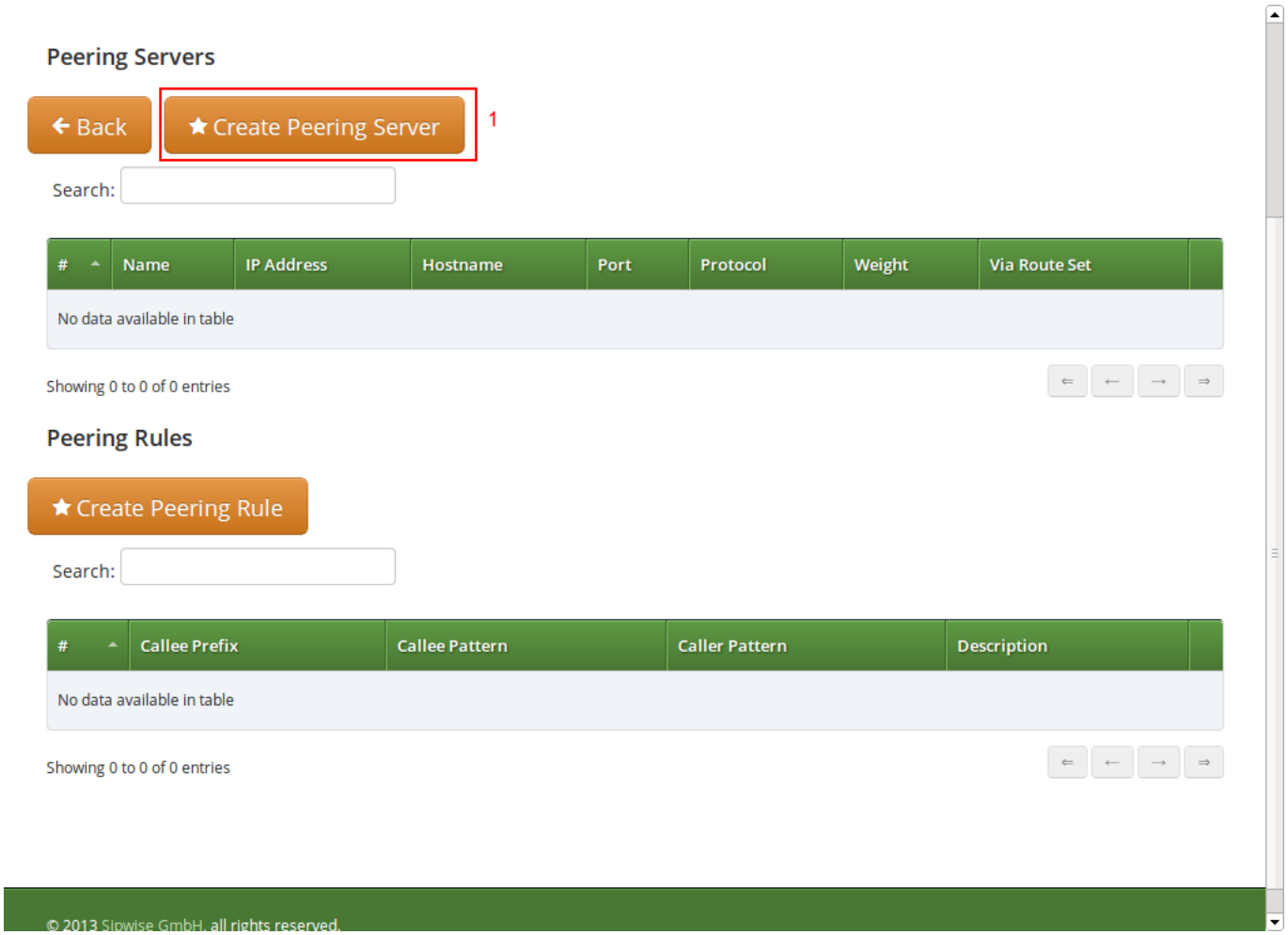
The *Priority* option defines which *Peering Group* to favor if two peering groups have peering rules matching an outbound call. *Peering Rules* are described below.

Then click *Save* to create the group.

3.5.2 Creating Peering Servers

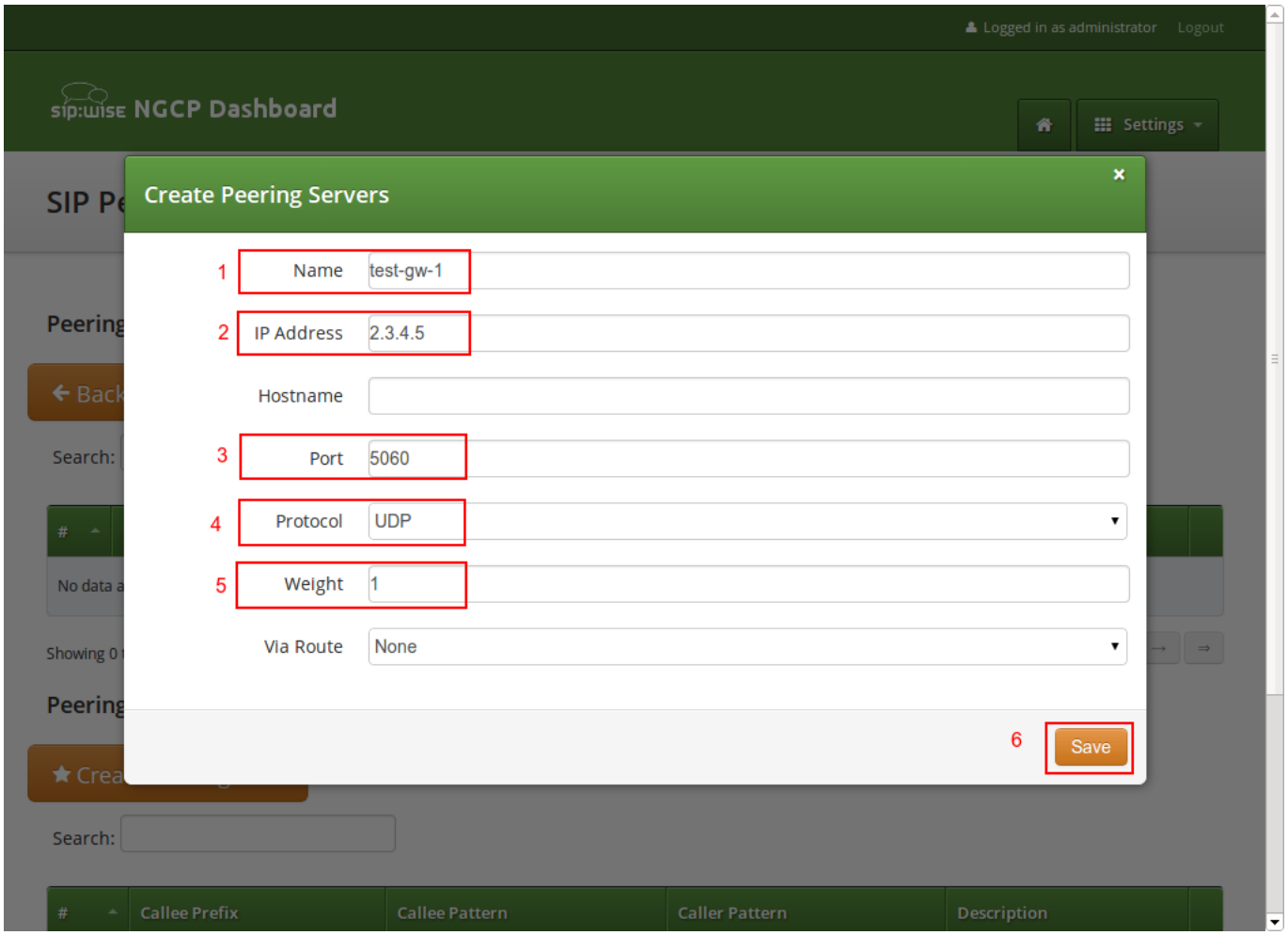
In the group created before, you need to add peering servers to route calls to and receive calls from. To do so, click on *Details* on the row of your new group in your peering group list.

To add your first *Peering Server*, click on the *Create Peering Server* button.



In this example, we will create a peering server with IP *2.3.4.5* and port *5060*:

- **Name**: `test-gw-1`
- **IP Address:** `2.3.4.5`
- **Hostname:** leave empty
- **Port:** `5060`
- **Protocol:** `UDP`
- **Weight:** `1`
- **Via Route:** `None`



Click *Save* to create the peering server.

Tip

The *hostname* field for a peering server is optional. Usually, the IP address of the peer is used as domain part in the Request URI. Some peers may require you to set a particular hostname instead of the IP address there, which can be done by filling in this field. The IP address must always be given though, and the request will always be sent to the IP address, no matter what you put into the *hostname* field.

Tip

If you want to add a peering server with an IPv6 address, enter the address without surrounding square brackets into the *IP Address* column, e.g. *::1*.

You can force an additional hop (e.g. via an external SBC) towards the peering server by using the *Via Route* option. The available options you can select there are defined in `/etc/ngcp-config/config.yml`, where you can add an array of SIP URIs in `kamailio→lb→external_sbc` like this:

```
kamailio:
  lb:
    external_sbc:
```

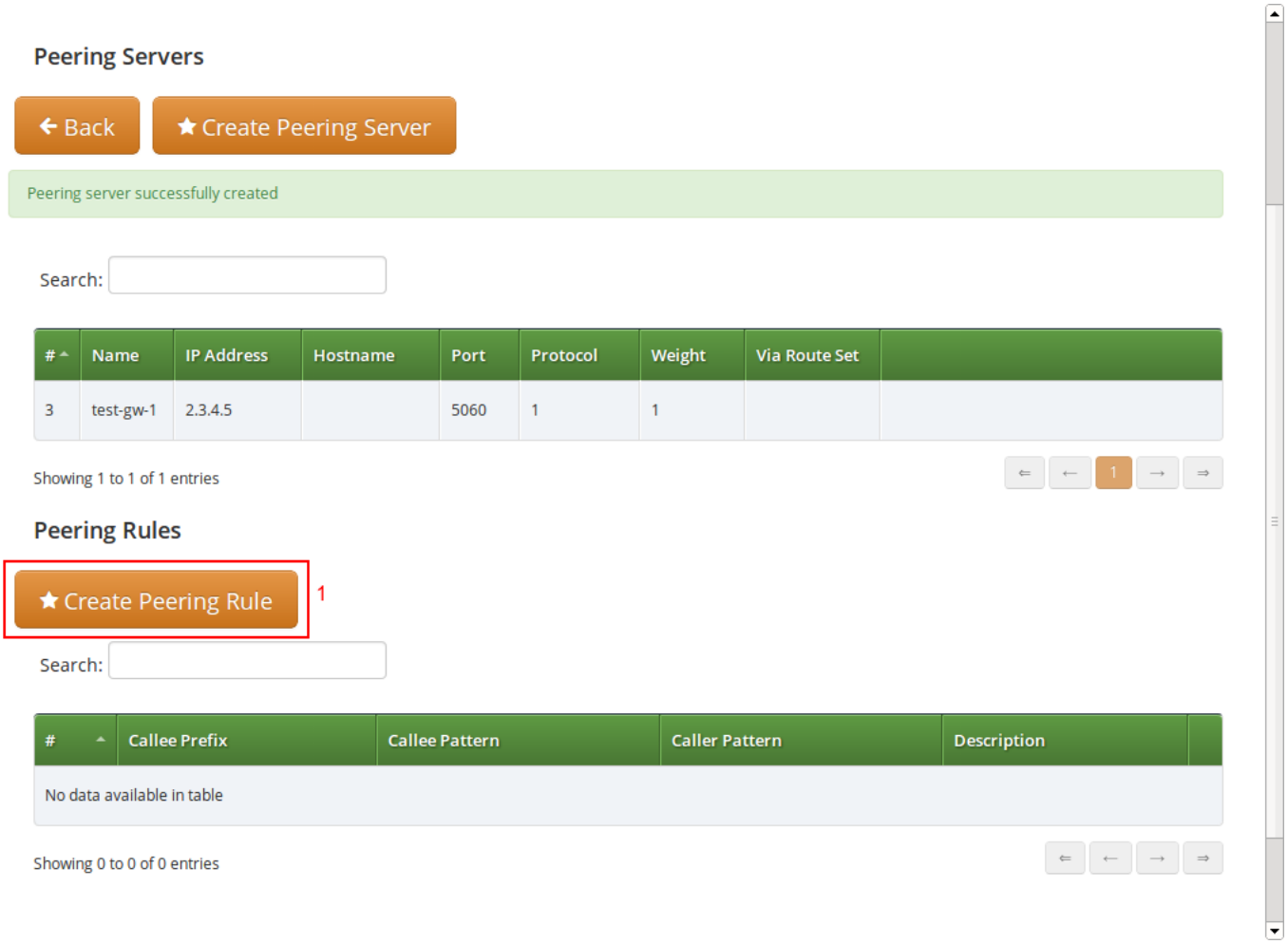
```
    - sip:192.168.0.1:5060
    - sip:192.168.0.2:5060
```

Execute `ngcpcfg apply`, then edit your peering server and select the hop from the *Via Route* selection.

Once a peering server has been created, this server can already send calls to the system.

Important

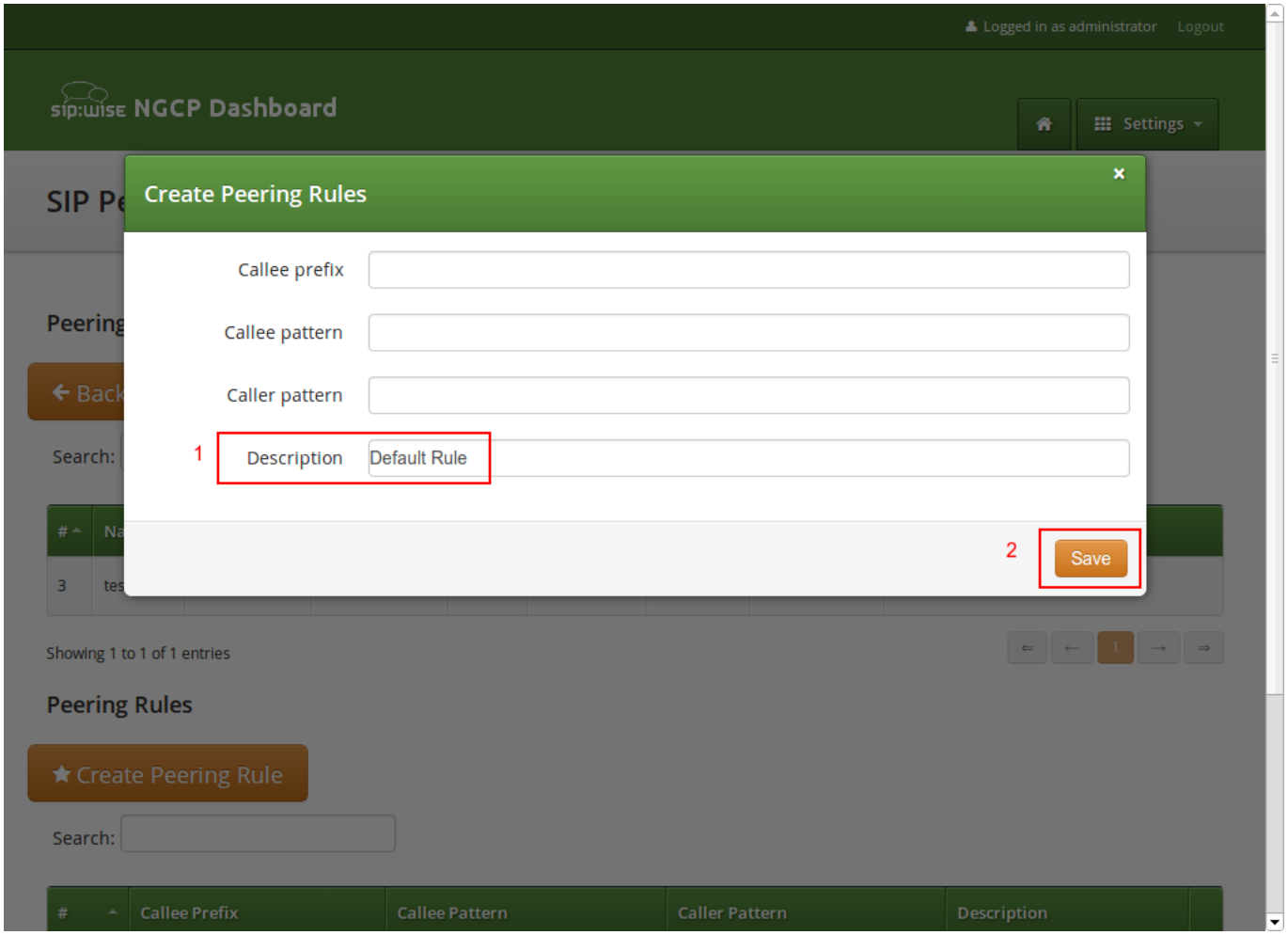
To be able to send outbound calls towards the servers in the *Peering Group*, you also need to define *Peering Rules*. They specify which source and destination numbers are going to be terminated over this group. To create a rule, click the *Create Peering Rule* button.



Since the previously created peering group will be the only one in our example, we have to add a default rule to route *all* calls via this group. To do so, create a new peering rule with the following values:

- **Callee Prefix:** leave empty
- **Callee Pattern:** leave empty
- **Caller Pattern:** leave empty

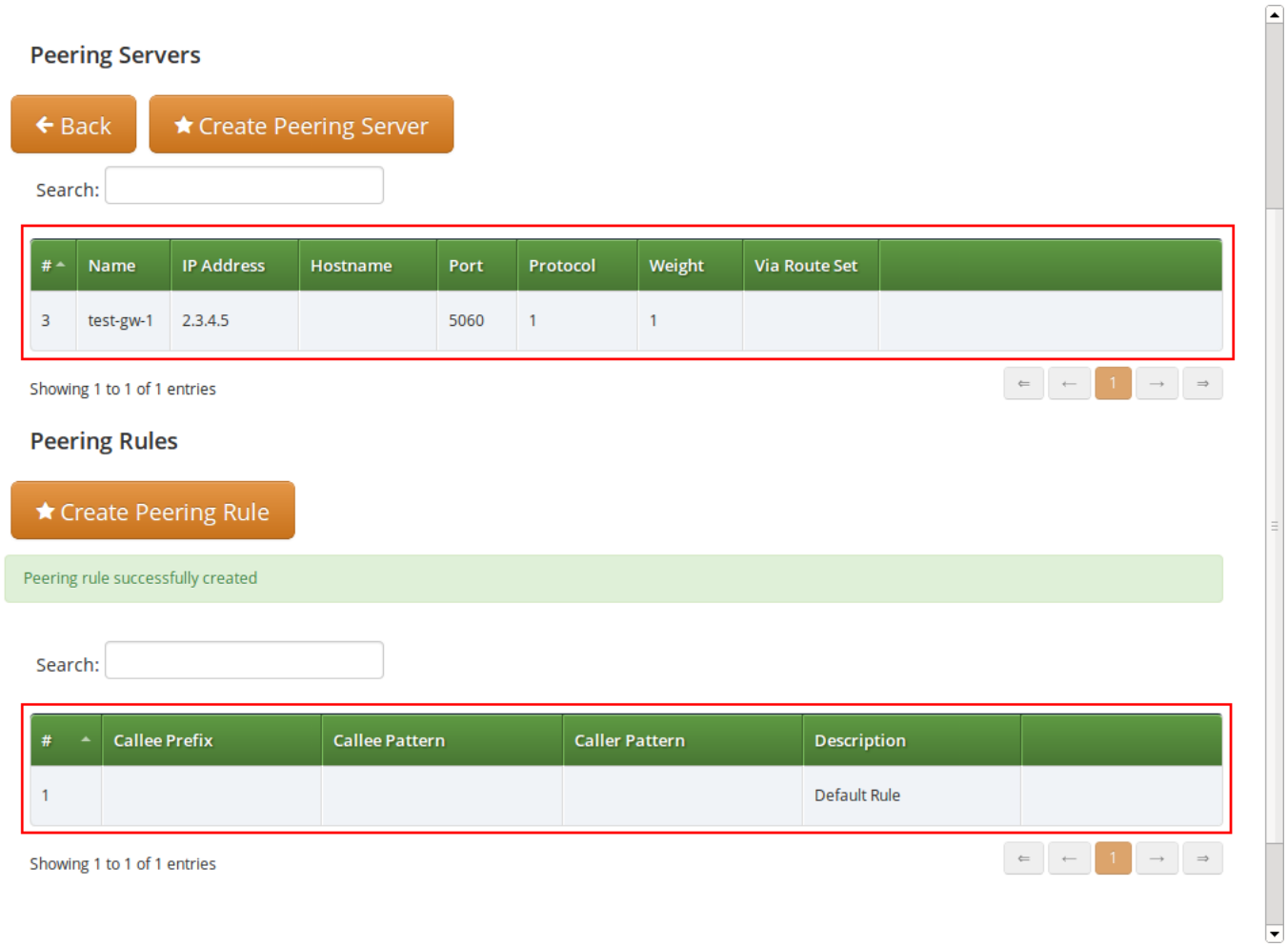
- **Description:** `Default Rule`



Then click *Save* to add the rule to your group.

Tip

If you set the caller or callee rules to refine what is routed via this peer, enter all phone numbers in full E.164 format, that is *<cc><ac><sn>*. TIP: The *Caller Pattern* field covers the whole URI including the subscriber domain, so you can only allow certain domains over this peer by putting for example `@example\.com` into this field.



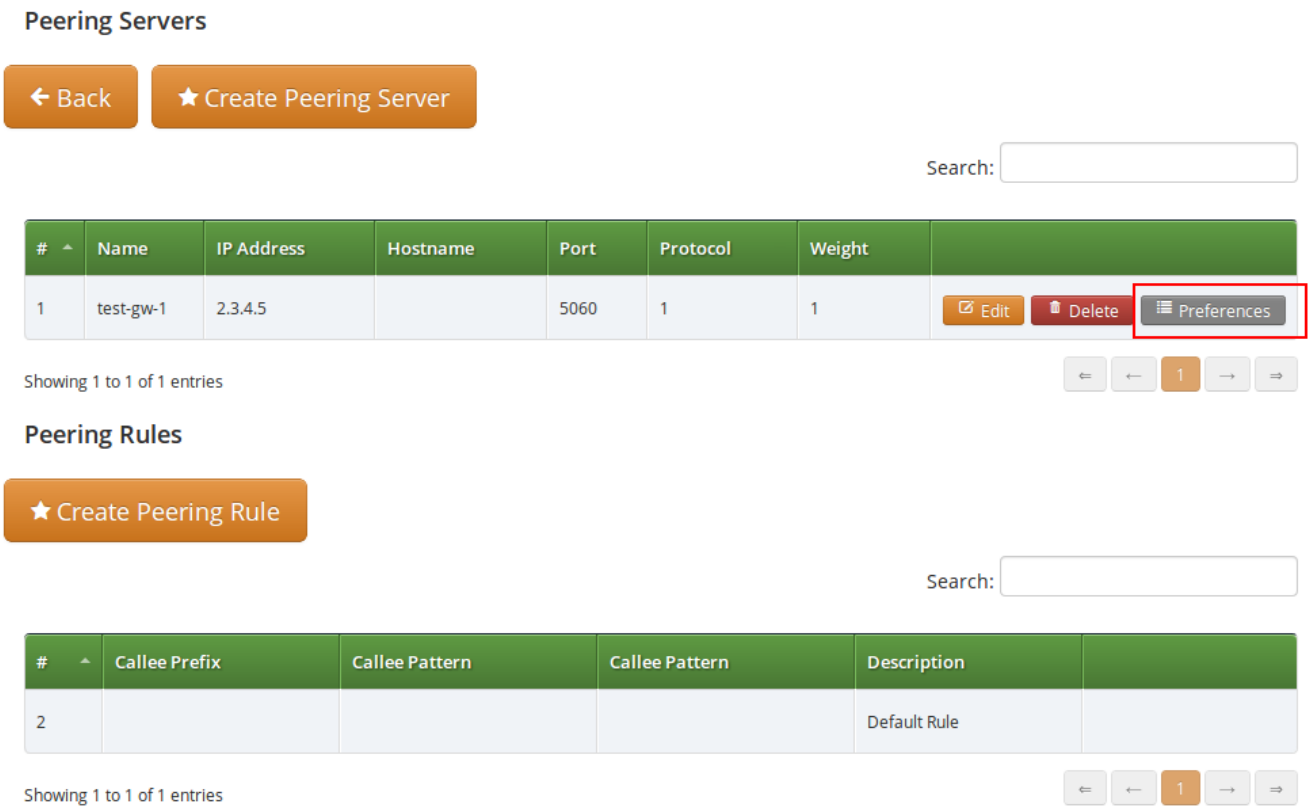
Important

The selection of peering servers for outbound calls is done in the following order: **1.** whether caller or callee pattern matched. **2.** length of the callee prefix. **3.** priority of the peering group. **4.** weight of the peering servers in the selected peering group. After one or more peering group(s) is matched for an outbound call, all servers in this group are tried, according to their weight (the bigger the weight of a server, the higher the probability that NGCP will send the call to it). If a peering server replies with SIP codes `408`, `500` or `503`, or if a peering server doesn't respond at all, the next peering server in the current peering group is used as a fallback, one after the other until the call succeeds. If no more servers are left in the current peering group, the next group which matches the peering rules is going to be used.

3.5.3 Authenticating and Registering against Peering Servers

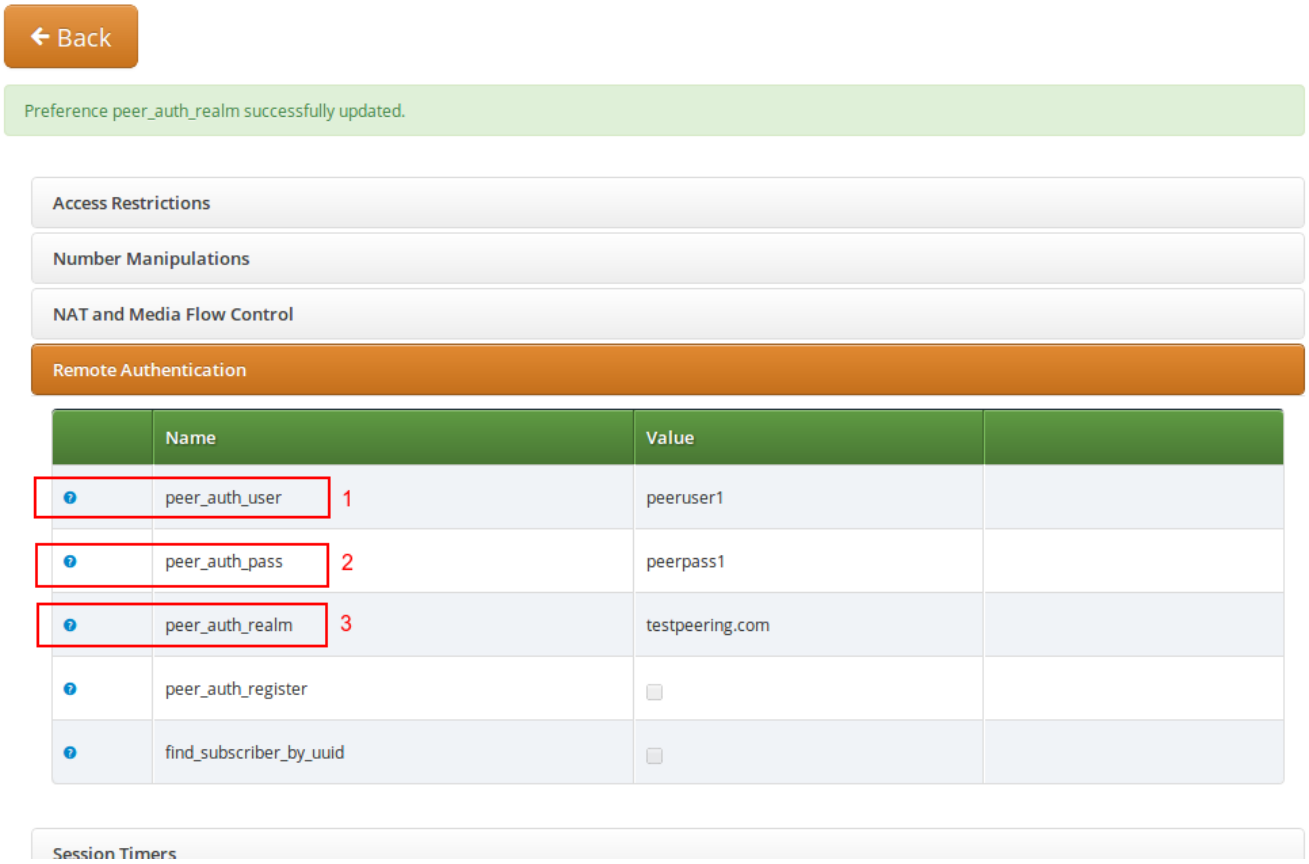
Proxy-Authentication for outbound calls

If a peering server requires the SPCE to authenticate for outbound calls (by sending a 407 as response to an INVITE), then you have to configure the authentication details in the *Preferences* view of your peer host.



To configure this setting, open the *Remote Authentication* tab and edit the following three preferences:

- **peer_auth_user:** `<username for peer auth>`
- **peer_auth_pass:** `<password for peer auth>`
- **peer_auth_realm:** `<domain for peer auth>`



Important

If you do NOT authenticate against a peer host, then the caller CLI is put into the `From` and `P-Asserted-Identity` headers, e.g. `"+4312345" <sip:+4312345@your-domain.com>`. If you DO authenticate, then the `From` header is `"+4312345" <sip:your_peer_auth_user@your_peer_auth_realm>` (the CLI is in the Display field, the peer_auth_user in the From username and the peer_auth_realm in the From domain), and the `P-Asserted-Identity` header is as usual like `<sip:+4312345@your-domain.com>`. So for presenting the correct CLI in *CLIP no screening* scenarios, your peering provider needs to extract the correct user either from the `From Display-Name` or from the `P-Asserted-Identity URI-User`.

Tip

You will notice that these three preferences are also shown in the *Subscriber Preferences* for each subscriber. There you can override the authentication details for all peer host if needed, e.g. if every user authenticates with his own separate credentials at your peering provider.

Tip

If **peer_auth_realm** is set, the system may overwrite the Request-URI with the peer_auth_realm value of the peer when sending the call to that peer or peer_auth_realm value of the subscriber when sending a call to the subscriber. Since this is rarely a desired behavior, it is disabled by default starting with NGCP release 3.2. If you need the replacement, you should set *set_ruri_to_peer_auth_realm: 'yes'* in */etc/ngcp-config/config.yml*.

Registering at a Peering Server

Unfortunately, the credentials configured above are not yet automatically used to register the SPCE at your peer hosts. There is however an easy manual way to do so, until this is addressed.

Configure your peering servers with the corresponding credentials in */etc/ngcp-config/templates/etc/ngcp-sems/etc/reg_agent.conf.tt2*, then execute *ngcpcfg apply*.

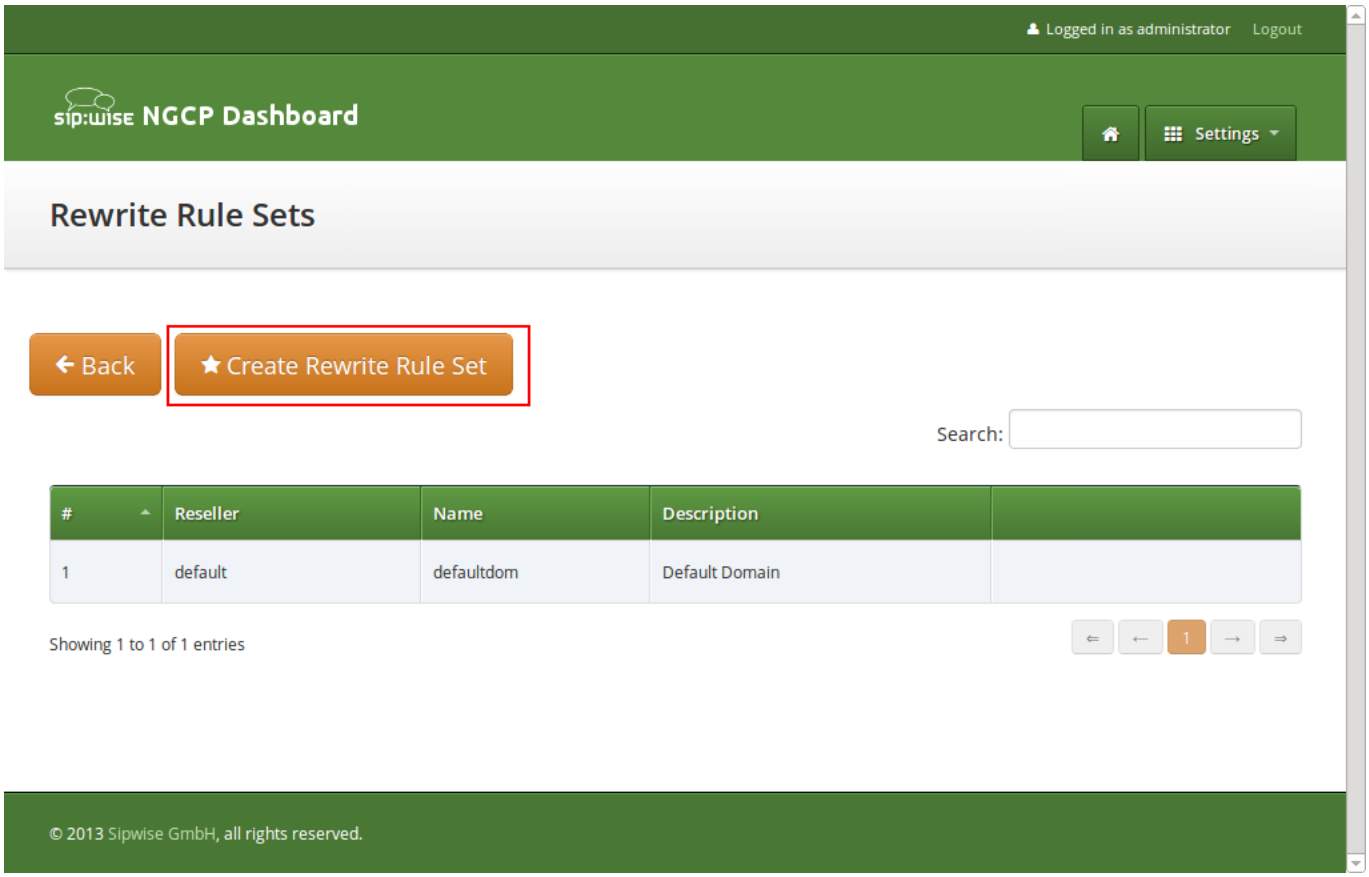
Important
Be aware that this will force SEMS to restart, which will drop all calls.

3.6 Configuring Rewrite Rule Sets

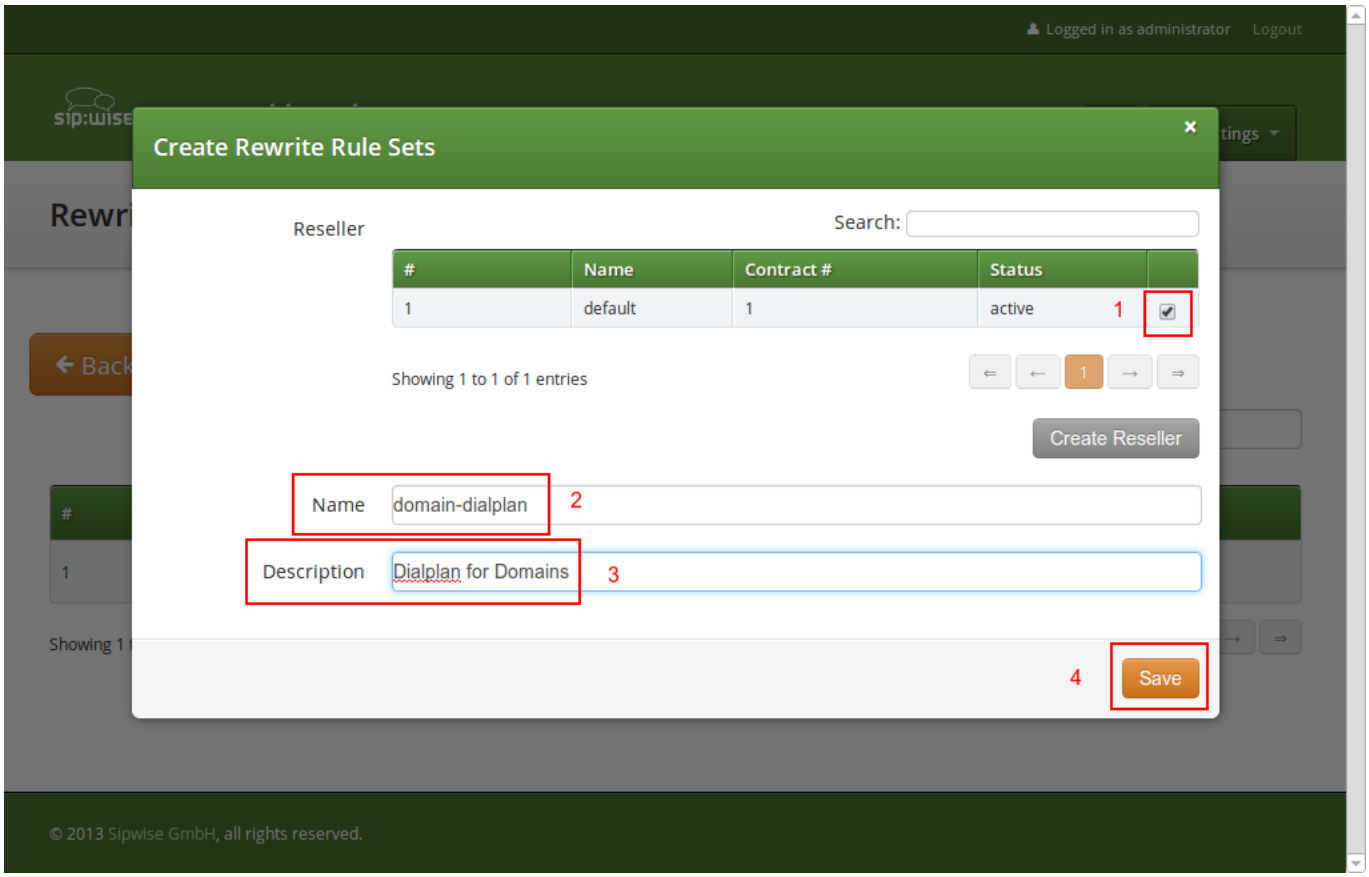
Important
On the NGCP, every phone number is treated in E.164 format *<country code><area code><subscriber number>*. Rewrite Rule Sets is a flexible tool to translate the caller and callee numbers to the proper format before the routing lookup and after the routing lookup separately. The created Rewrite Rule Sets can be assigned to the domains, subscribers and peers as a preference.

You would normally begin with creating a Rewrite Rule Set for your SIP domains. This is used to control what an end user can dial for outbound calls, and what is displayed as the calling party on inbound calls. The subscribers within a domain inherit Rewrite Rule Sets of that domain, unless this is overridden by a subscriber Rewrite Rule Set preference.

To create a new Rewrite Rule Set, go to *Settings→Rewrite Rule Sets*. There you can create a Set identified by a name. This name is later shown in your peer-, domain- and user-preferences where you can select the rule set you want to use.

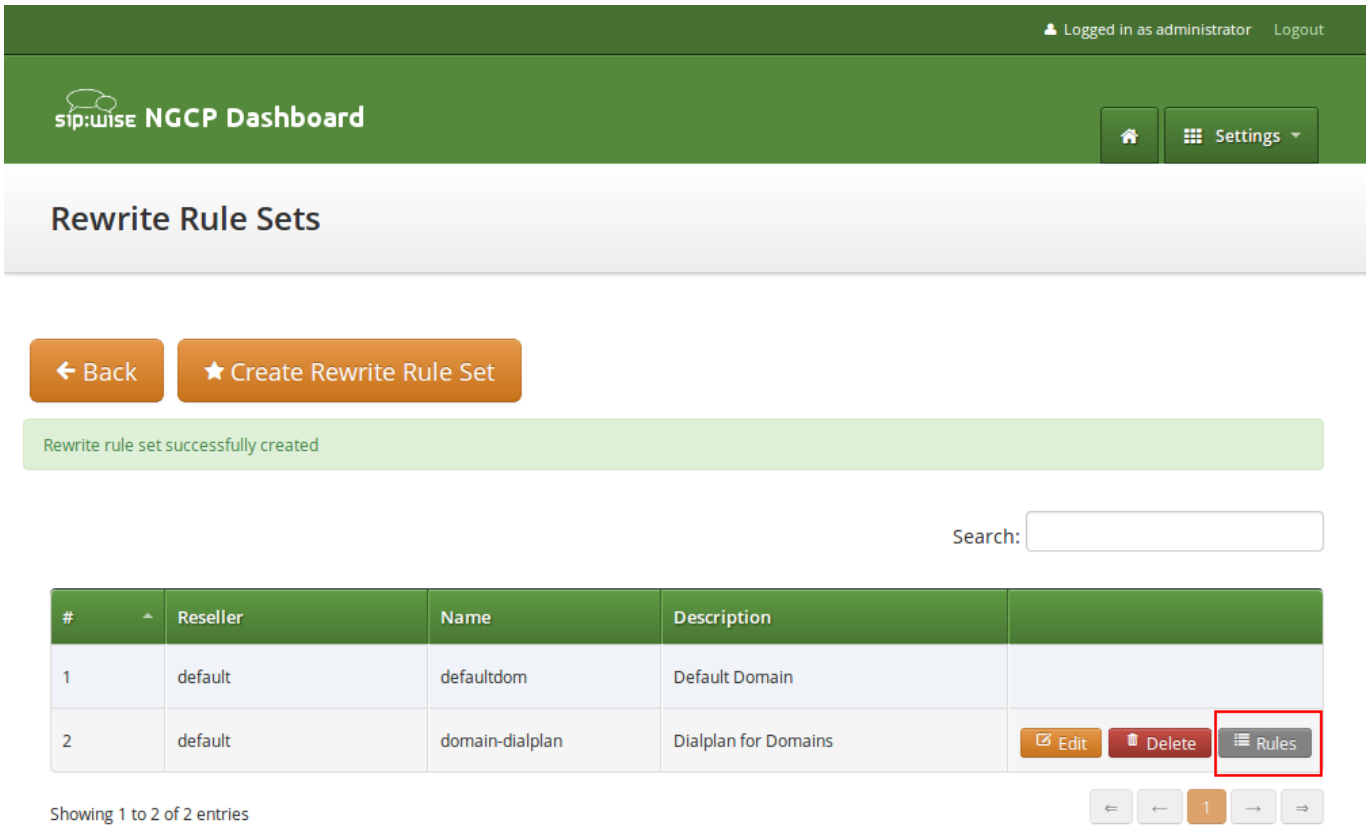


Click *Create Rewrite Rule Set* and fill in the form accordingly.



Press the *Save* button to create the set.

To view the *Rewrite Rules* within a set, hover over the row and click the *Rules* button.



The rules are ordered by *Caller* and *Callee* as well as direction *Inbound* and *Outbound*.

Tip

In Europe, the following formats are widely accepted: *+<cc><ac><sn>*, *00<cc><ac><sn>* and *0<ac><sn>*. Also, some countries allow the areacode-internal calls where only subscriber number is dialed to reach another number in the same area. Within this section, we will use these formats to show how to use rewrite rules to normalize and denormalize number formats.

3.6.1 Inbound Rewrite Rules for Caller

These rules are used to normalize user-provided numbers (e.g. passed in *From Display Name* or *P-Preferred-Identity* headers) into E.164 format. In our example, we'll normalize the three different formats mentioned above into E.164 format.

To create the following rules, click on the *Create Rewrite Rule* for each of them and fill them with the values provided below.

STRIP LEADING 00 OR +

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`
- Description: `International to E.164`
- Direction: `Inbound`

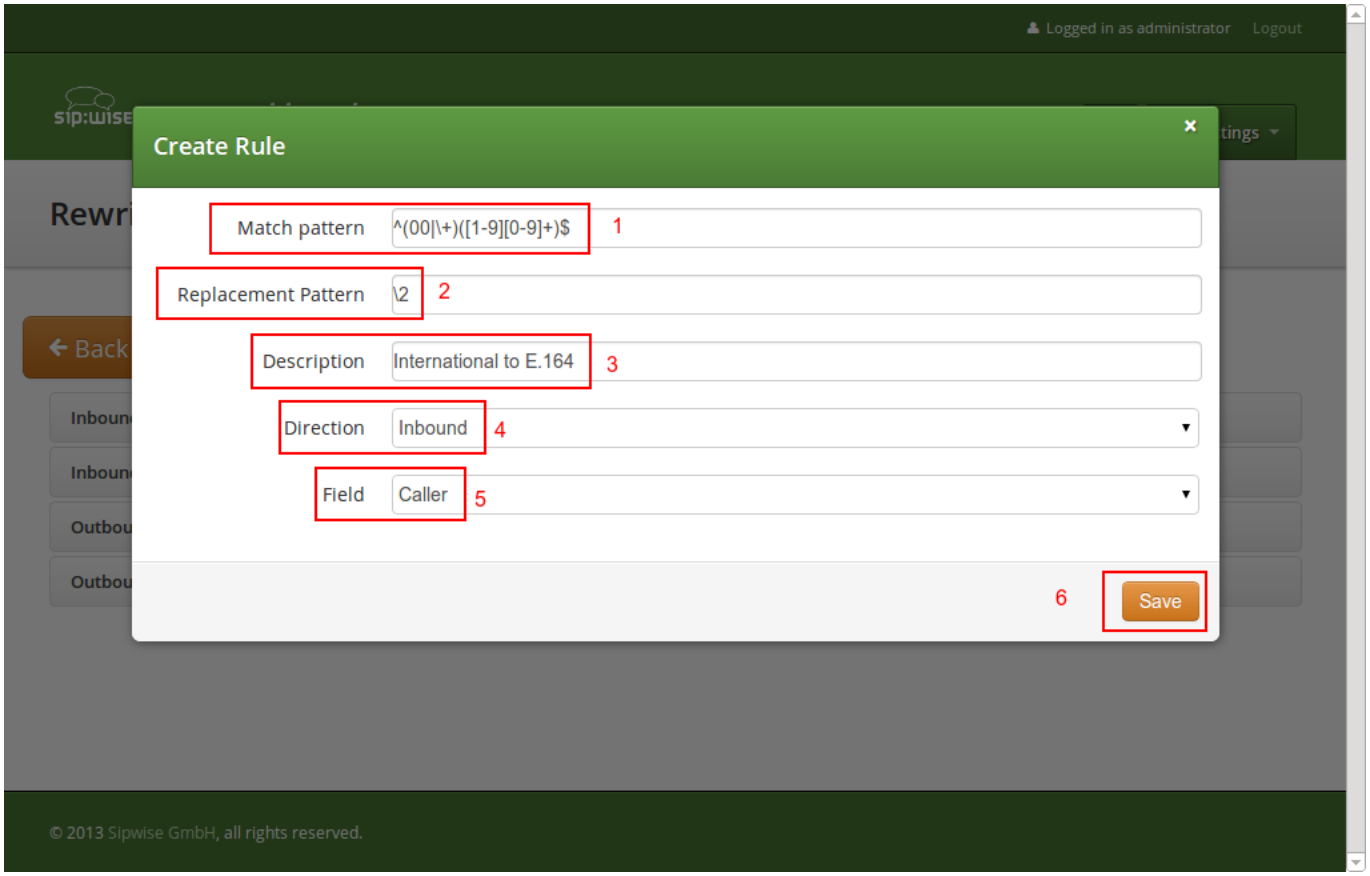
- Field: `Caller`

REPLACE 0 BY CALLER'S COUNTRY CODE:

- Match Pattern: `^0([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}\1`
- Description: `National to E.164`
- Direction: `Inbound`
- Field: `Caller`

NORMALIZE LOCAL CALLS:

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}${caller_ac}\1`
- Description: `Local to E.164`
- Direction: `Inbound`
- Field: `Caller`



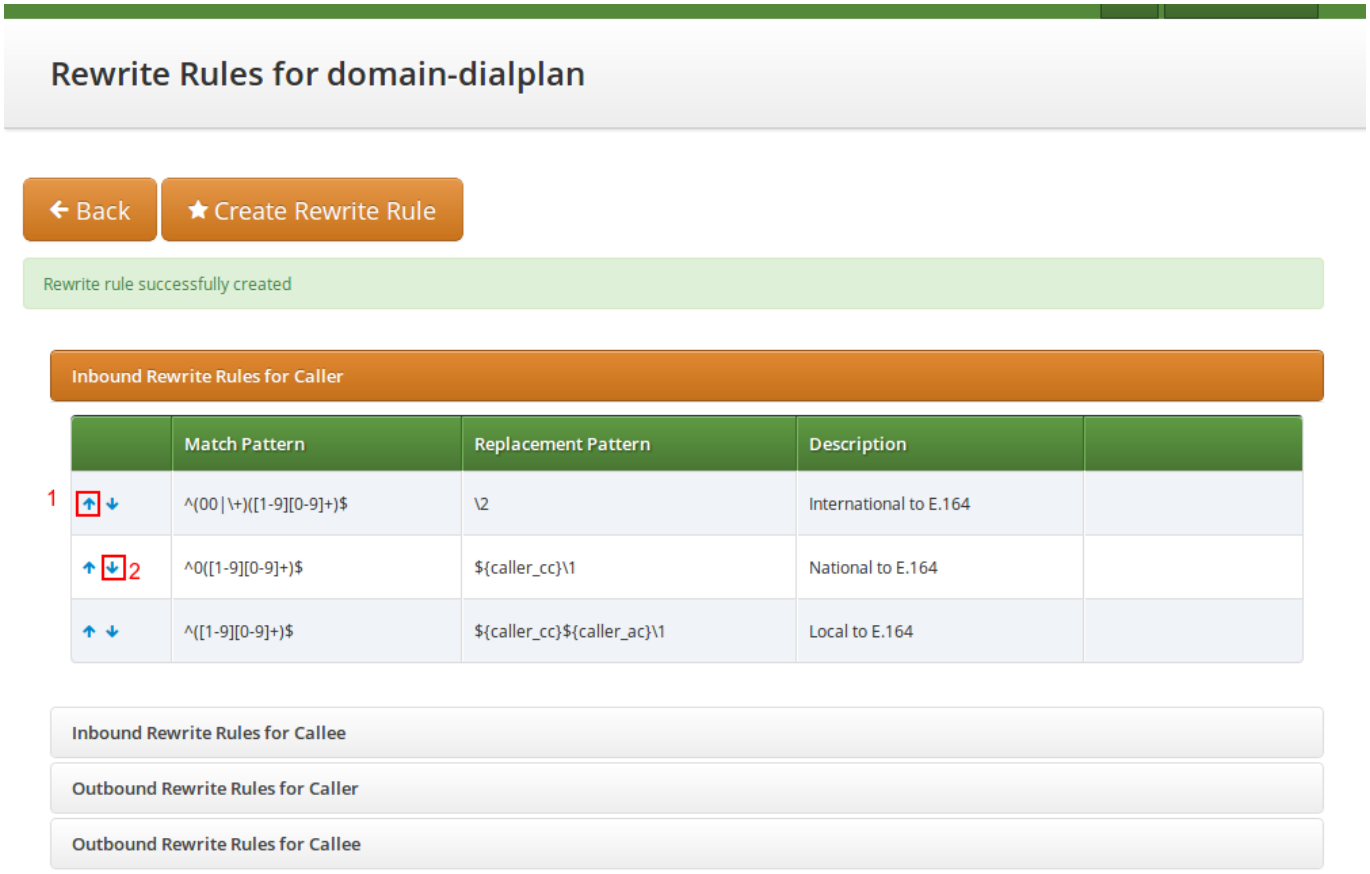
Normalization for national and local calls is possible with special variables `${caller_cc}` and `${caller_ac}` that can be used in Replacement Pattern and are substituted by the country and area code accordingly during the call routing.

Important

These variables are only being filled in when a call originates from a subscriber (because only then the cc/ac information is known by the system), so you can not use them when a calls comes from a SIP peer (the variables will be just empty in this case).

Tip

When routing a call, the rewrite processing is stopped after the first match of a rule, starting from top to bottom. If you have two rules (e.g. a generic one and a more specific one), where both of them would match some numbers, reorder them with the up/down arrows into the appropriate position.



3.6.2 Inbound Rewrite Rules for Callee

These rules are used to rewrite the number the end user dials to place a call to a standard format for routing lookup. In our example, we again allow the three different formats mentioned above and again normalize them to E.164, so we put in the same rules as for the caller.

STRIP LEADING 00 OR +

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`

- Description: `International to E.164`
- Direction: `Inbound`
- Field: `Callee`

REPLACE `0` BY CALLER'S COUNTRY CODE:

- Match Pattern: `^0([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}\1`
- Description: `National to E.164`
- Direction: `Inbound`
- Field: `Callee`

NORMALIZE AREACODE-INTERNAL CALLS:

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}${caller_ac}\1`
- Description: `Local to E.164`
- Direction: `Inbound`
- Field: `Callee`

Tip

Our provided rules will only match if the caller dials a numeric number. If he dials an alphanumeric SIP URI, none of our rules will match and no rewriting will be done. You can however define rules for that as well. For example, you could allow your end users to dial `support` and rewrite that to your support hotline using the match pattern `^support$` and the replace pattern `43800999000` or whatever your support hotline number is.

3.6.3 Outbound Rewrite Rules for Caller

These rules are used to rewrite the calling party number for a call to an end user. For example, if you want the device of your end user to show *0<ac><sn>* if a national number calls this user, and *00<cc><ac><sn>* if an international number calls, put the following rules there.

REPLACE AUSTRIAN COUNTRY CODE `43` BY `0`

- Match Pattern: `^43([1-9][0-9]+)$`
- Replacement Pattern: `0\1`
- Description: `E.164 to Austria National`

- Direction: `Outbound`
- Field: `Caller`

PREFIX `00` FOR INTERNATIONAL CALLER

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `00\1`
- Description: `E.164 to International`
- Direction: `Outbound`
- Field: `Caller`

Tip

Note that both of the rules would match a number starting with `43`, so reorder the national rule to be above the international one (if it's not already the case).

3.6.4 Outbound Rewrite Rules for Callee

These rules are used to rewrite the called party number immediately before sending out the call on the network. This gives you an extra flexibility by controlling the way request appears on a wire, when your SBC or other device expects the called party number to have a particular tech-prefix. It can be used on calls to end users too if you want to do some processing in intermediate SIP device, e.g. apply legal intercept selectively to some subscribers.

PREFIX `SIPSP#` FOR ALL CALLS

- Match Pattern: `^([0-9]+)$`
- Replacement Pattern: `sipsp#\1`
- Description: `Intercept this call`
- Direction: `Outbound`
- Field: `Callee`

3.6.5 Emergency Number Handling

Configuring Emergency Numbers is also done via Rewrite Rules.

For Emergency Calls from a subscriber to the platform, you need to define an *Inbound Rewrite Rule For Callee*, which adds a prefix `emergency_` to the number (and can rewrite the number completely as well at the same time). If the proxy detects a call to a SIP URI starting with `emergency_`, it will enter a special routing logic bypassing various checks which might make a normal call fail (e.g. due to locked or blocked numbers, insufficient credits or exceeding the max. amount of parallel calls).

TAG AN EMERGENCY CALL

- Match Pattern: `^(911|112)$`
- Replacement Pattern: `emergency_\1`
- Description: `Tag Emergency Numbers`
- Direction: `Inbound`
- Field: `Callee`

To route an Emergency Call to a Peer, you can select a specific peering group by adding a peering rule with a *callee prefix* set to `emergency_` to a peering group.

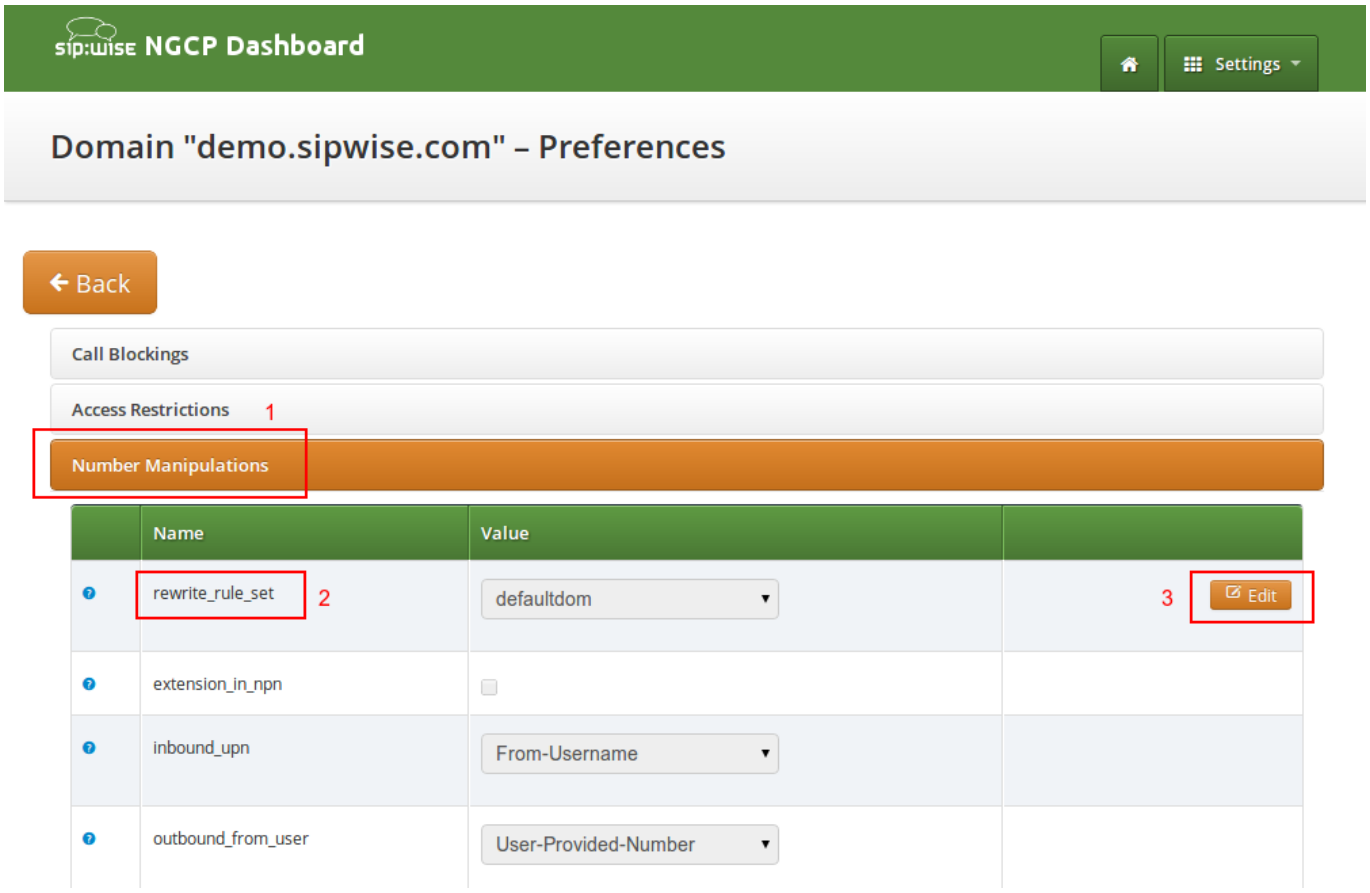
In order to normalize the emergency number to a valid format accepted by the peer, you need to assign an *Outbound Rewrite Rule For Callee*, which strips off the `emergency_` prefix. You can also use the variables `${caller_emergency_cli}`, `${caller_emergency_prefix}` and `${caller_emergency_suffix}` as well as `${caller_ac}` and `${caller_cc}`, which are all configurable per subscriber to rewrite the number into a valid format.

NORMALIZE EMERGENCY CALL FOR PEER

- Match Pattern: `^emergency_(.+)$`
- Replacement Pattern: `${caller_emergency_prefix}${caller_ac}\1`
- Description: `Normalize Emergency Numbers`
- Direction: `Outbound`
- Field: `Caller`

3.6.6 Assigning Rewrite Rule Sets to Domains and Subscribers

Once you have finished to define your Rewrite Rule Sets, you need to assign them. For sets to be used for subscribers, you can assign them to their corresponding domain, which then acts as default set for all subscribers. To do so, go to *Settings→Domains* and click *Preferences* on the domain you want the set to assign to. Click on *Edit* and select the Rewrite Rule Set created before.



You can do the same in the *Preferences* of your subscribers to override the rule on a subscriber basis. That way, you can finely control down to an individual user the dial-plan to be used. Go to *Settings→Subscribers*, click the *Details* button on the subscriber you want to edit, the click the *Preferences* button.

3.6.7 Creating Dialplans for Peering Servers

For each peering server, you can use one of the Rewrite Rule Sets that was created previously as explained in Section 3.6 (keep in mind that special variables `${caller_ac}` and `${caller_cc}` can not be used when the call comes from a peer). To do so, click on the name of the peering server, look for the preference called *Rewrite Rule Sets*.

If your peering servers don't send numbers in E.164 format *<cc><ac><sn>*, you need to create *Inbound Rewrite Rules* for each peering server to normalize the numbers for caller and callee to this format, e.g. by stripping leading + or put them from national into E.164 format.

Likewise, if your peering servers don't accept this format, you need to create *Outbound Rewrite Rules* for each of them, for example to append a + to the numbers.

4 Advanced Subscriber Configuration

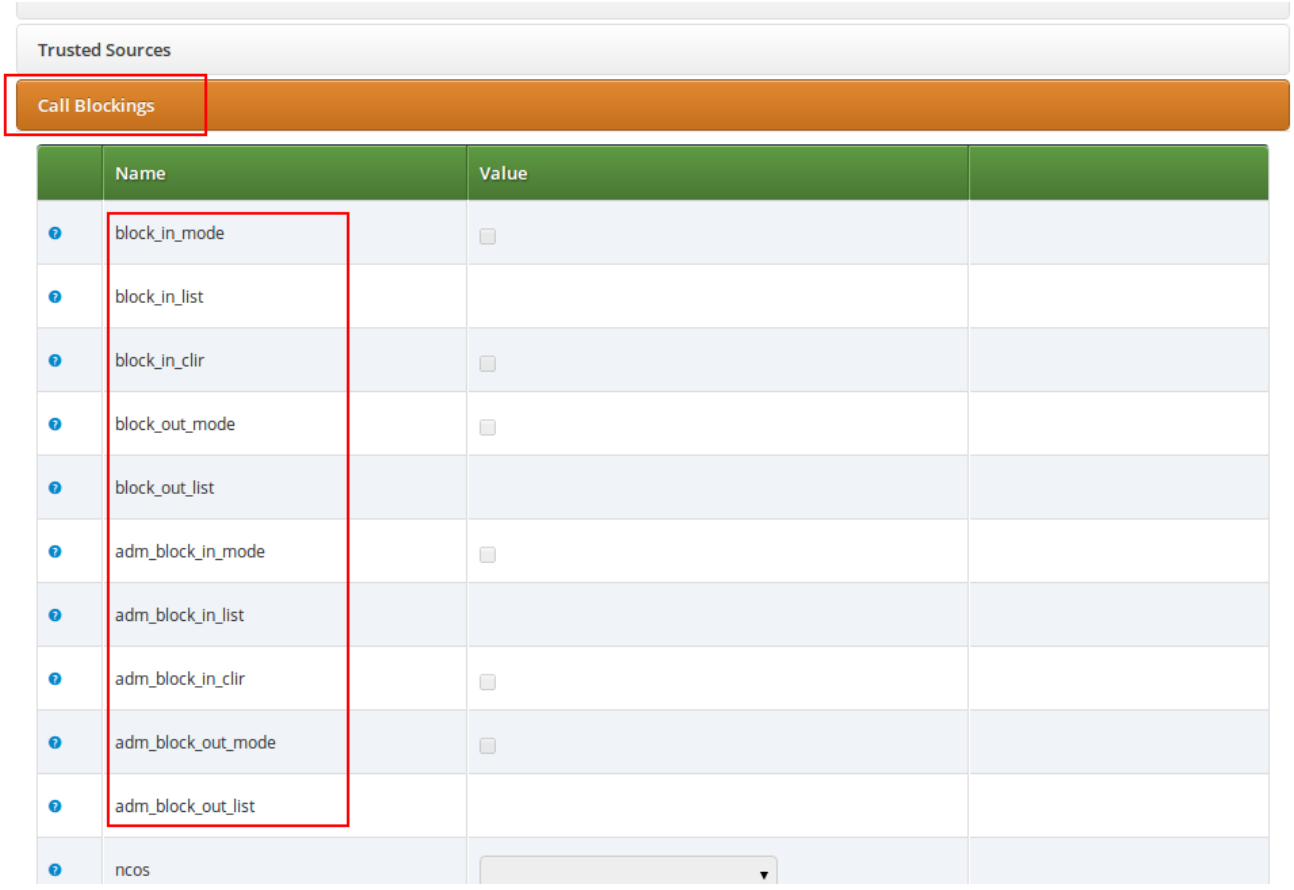
The sip:carrier provides a large amount of subscriber features in order to offer compelling VoIP services to end customers, and also to cover as many deployment scenarios as possible. In this chapter, we will go over the features and describe their behavior and their use cases.

4.1 Access Control for SIP Calls

There are two different methods to provide fine-grained call admission control to both subscribers and admins. One is *Block Lists*, where you can define which numbers or patterns can be called from a subscriber to outbound direction and which numbers or patterns are allowed to call a subscriber in inbound direction. The other is *NCOS Levels*, where the admin predefines rules for outbound calls, which are grouped in certain levels. The user can then just choose the level, or the admin can restrict a user to a certain level. Also sip:carrier offers some options to restrict the IP addresses that subscriber is allowed to use the service from. The following chapters will discuss these features in detail.

4.1.1 Block Lists

Block Lists provide a way to control which users/numbers are able to call or to be called, based on a subscriber level, and can be found in the *Call Blockings* section of the subscriber preferences.



Block Lists are separated into *Administrative Block Lists* (*adm_block_*) and *Subscriber Block Lists* (*block_*). They both have

the same behavior, but Administrative Block Lists take higher precedence. Administrative Block Lists are only accessible by the system administrator and can thus be used to override any Subscriber Block Lists, e.g. to block certain destinations. The following break-down of the various block features apply to both types of lists.

Block Modes

Block lists can either be *whitelists* or *blacklists* and are controlled by the User Preferences *block_in_mode*, *block_out*mode__ and their administrative counterparts.

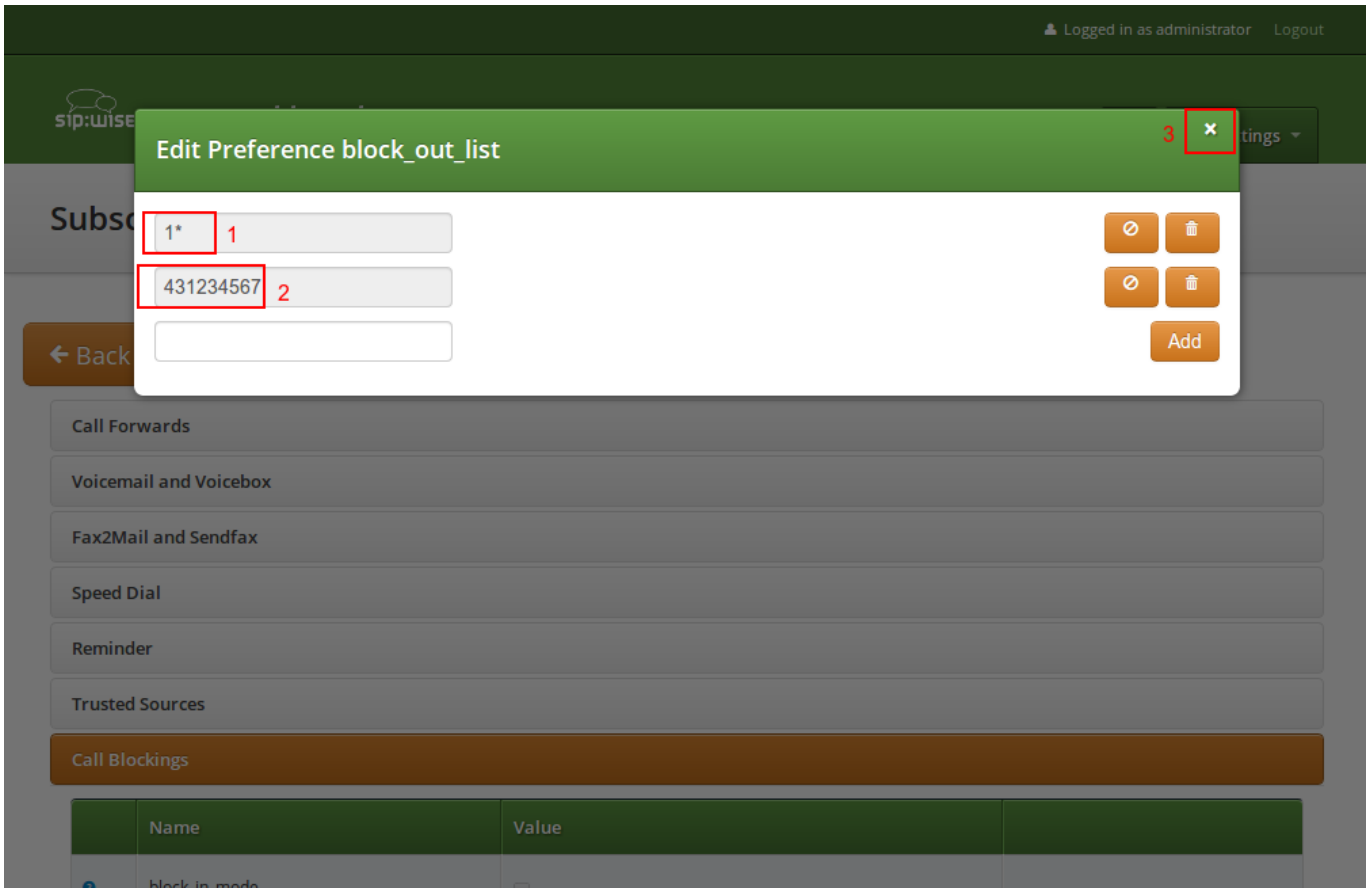
- The *blacklist* mode (option is not checked tells the system to **allow anything except the entries in the list**. This mode is used if you want to just block certain numbers and allow all the rest.
- The *whitelist* mode indicates to **reject anything except the entries in the list**. This is used if you want to enforce a strict policy and allow only selected destinations or sources.

You can change a list mode from one to the other at any time.

Block Lists

The list contents are controlled by the User Preferences *block_in_list*, *block_out_list* and their administrative counterparts. Click on the *Edit* button in the *Preferences* view to define the list entries.

In block list entries, you can provide shell patterns like `*` and `[]`. The behavior of the list is controlled by the *block_xxx_mode* feature (so they are either allowed or rejected). In our example above we have *block_out_mode* set to *blacklist*, so all calls to US numbers and to the Austrian number +431234567 are going to be rejected.



Click the *Close* icon once you're done editing your list.

Block Anonymous Numbers

For incoming call, the User Preference *block_in_clir* and *adm_block_in_clir* controls whether or not to reject incoming calls with number supression (either "[Aa]nonymous" in the display- or user-part of the From-URI or a header *Privacy: id* is set). This flag is independent from the Block Mode.

4.1.2 NCOS Levels

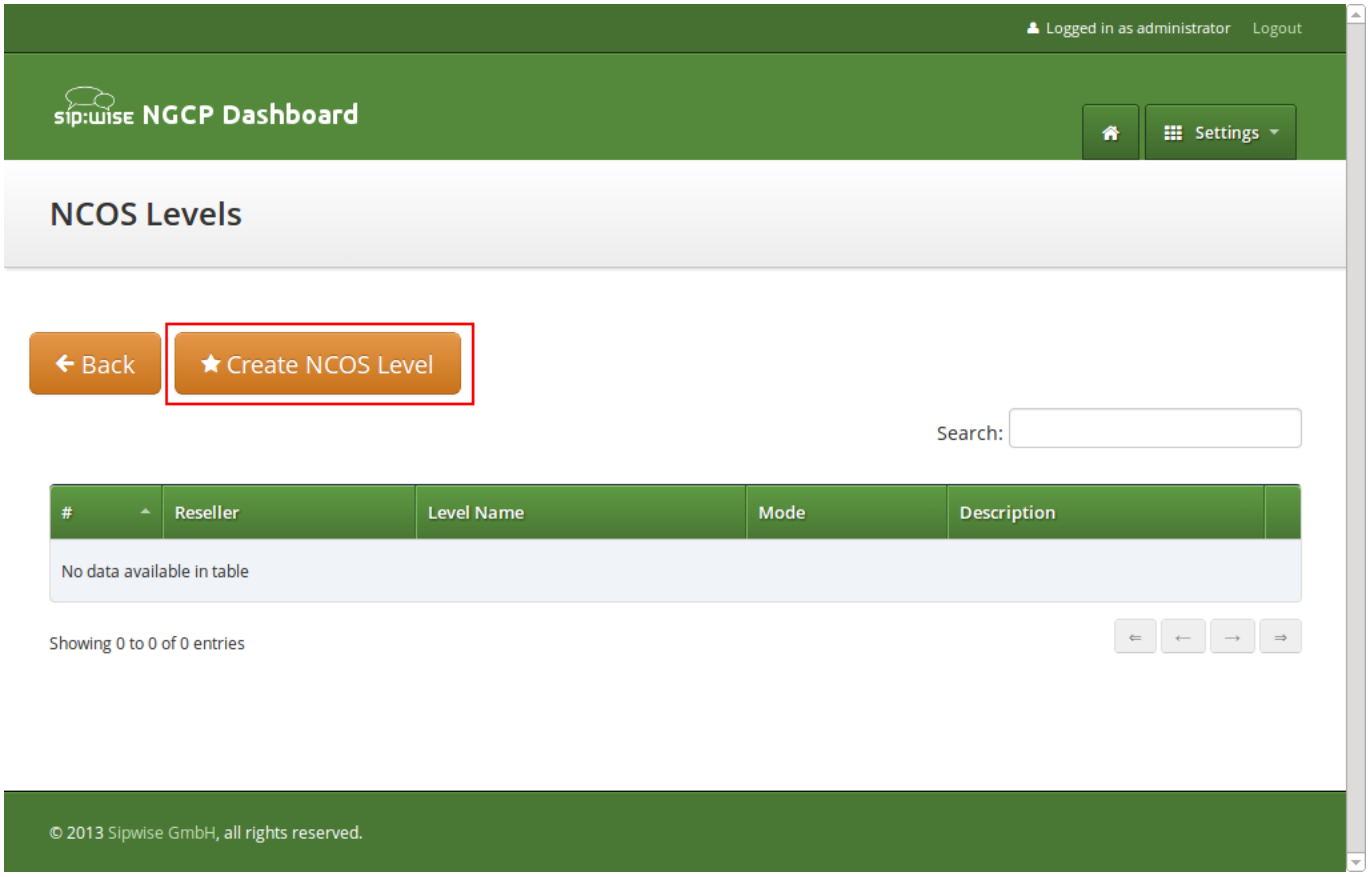
NCOS Levels provide predefined lists of allowed or denied destinations for outbound calls of local subscribers. Compared to *Block Lists*, they are much easier to manage, because they are defined on a global scope, and the individual levels can then be assigned to each subscriber. Again there is the distinction for user- and administrative-levels.

NCOS levels can either be *whitelists* or *blacklists*.

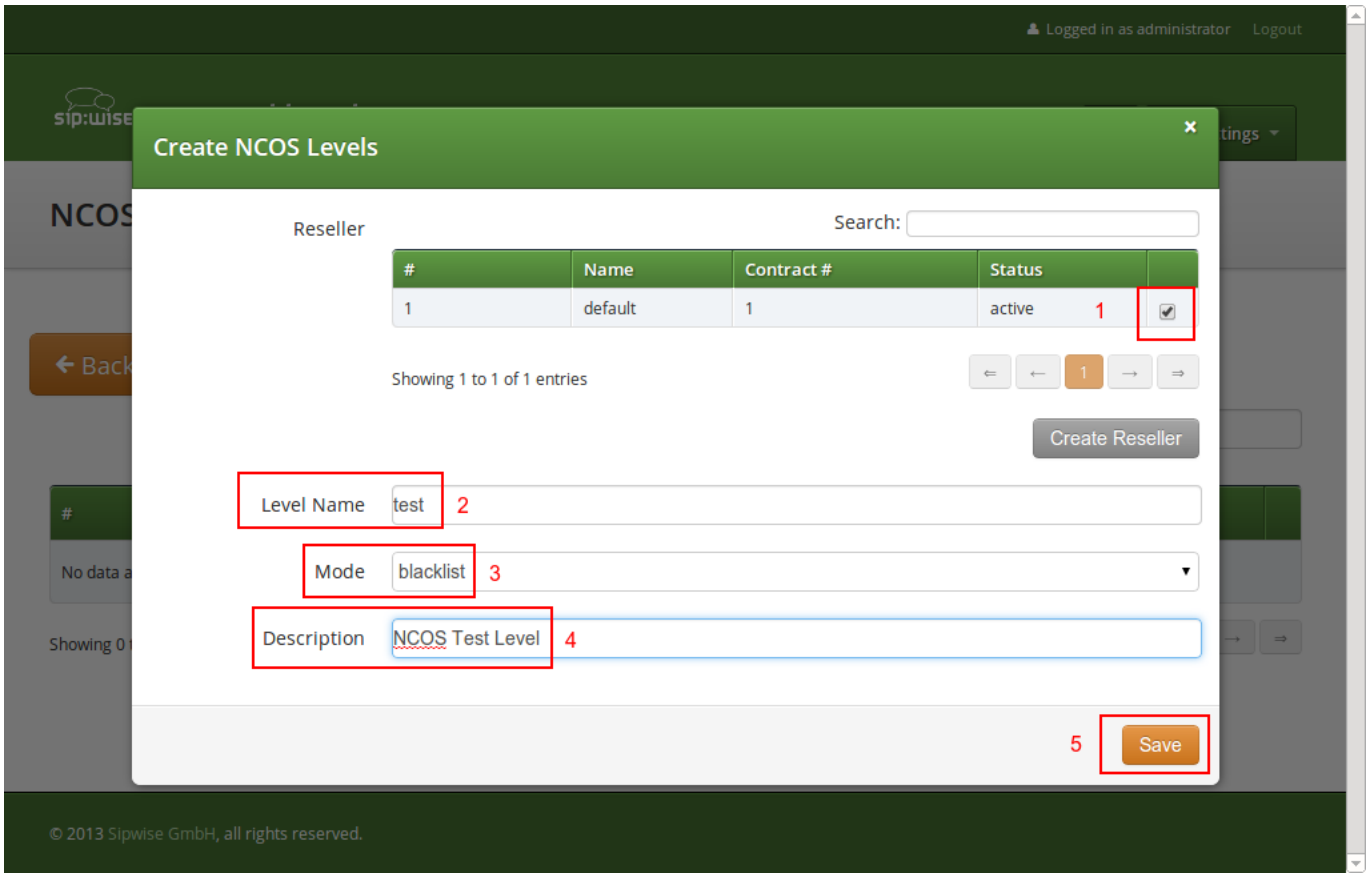
- The *blacklist* mode indicates to **allow everything except the entries in this level**. This mode is used if you want to just block certain destinations and allow all the rest.
- The *whitelist* mode indicates to **reject anything except the entries in this level**. This is used if you want to enforce a strict policy and allow only selected destinations.

Creating NCOS Levels

To create an NCOS Level, go to *Settings→NCOS Levels* and press the *Create NCOS Level* button.

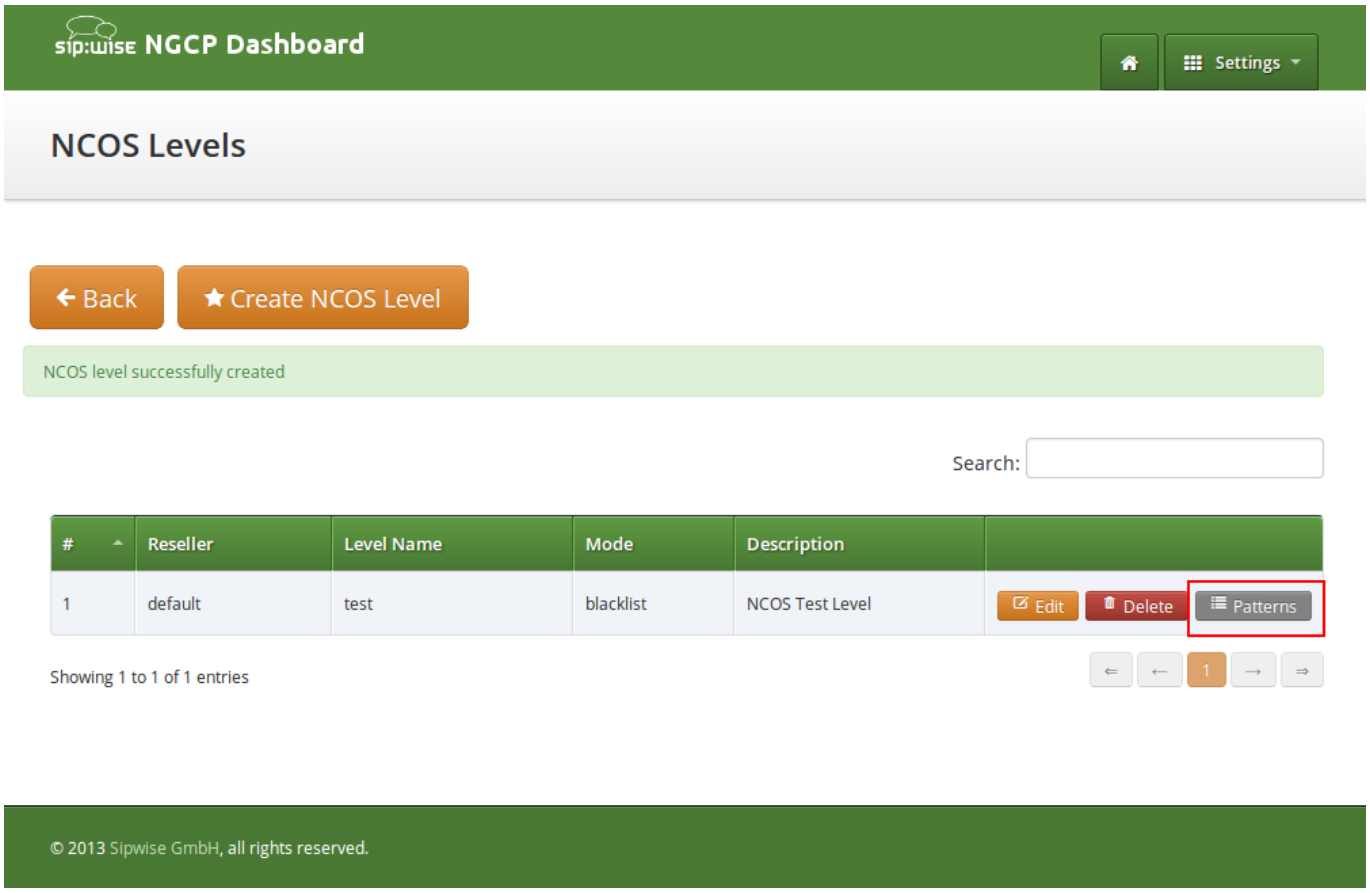


Select a reseller, enter a name, select the mode and add a description, then click the *Save* button.

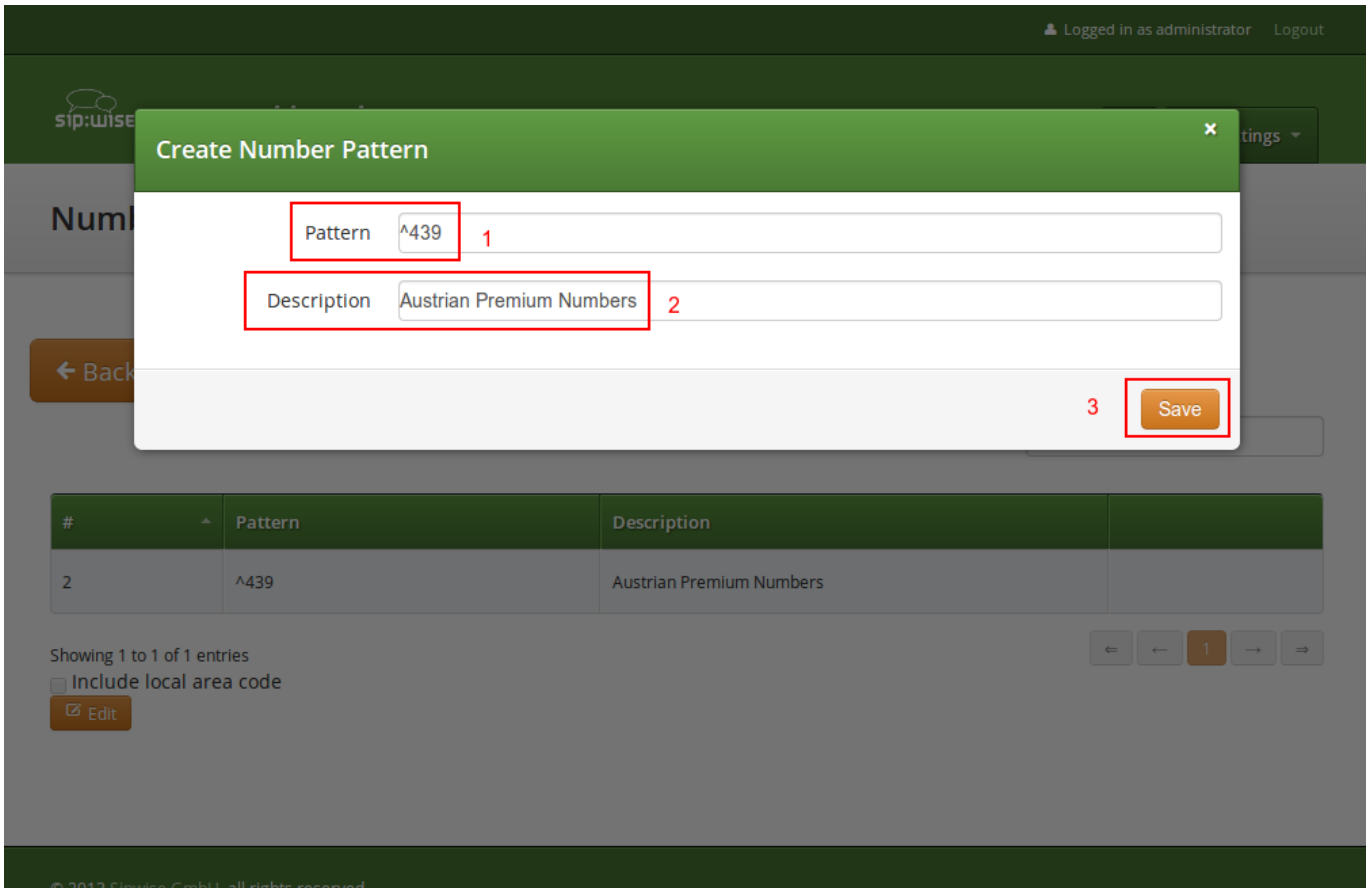


Creating Rules per NCOS Level

To define the rules within the newly created NCOS Level, click on the *Patterns* button of the level.



In the *Number Patterns* view you can create multiple patterns to define your level, one after the other. Click on the *Create Pattern Entry* Button on top and fill out the form.

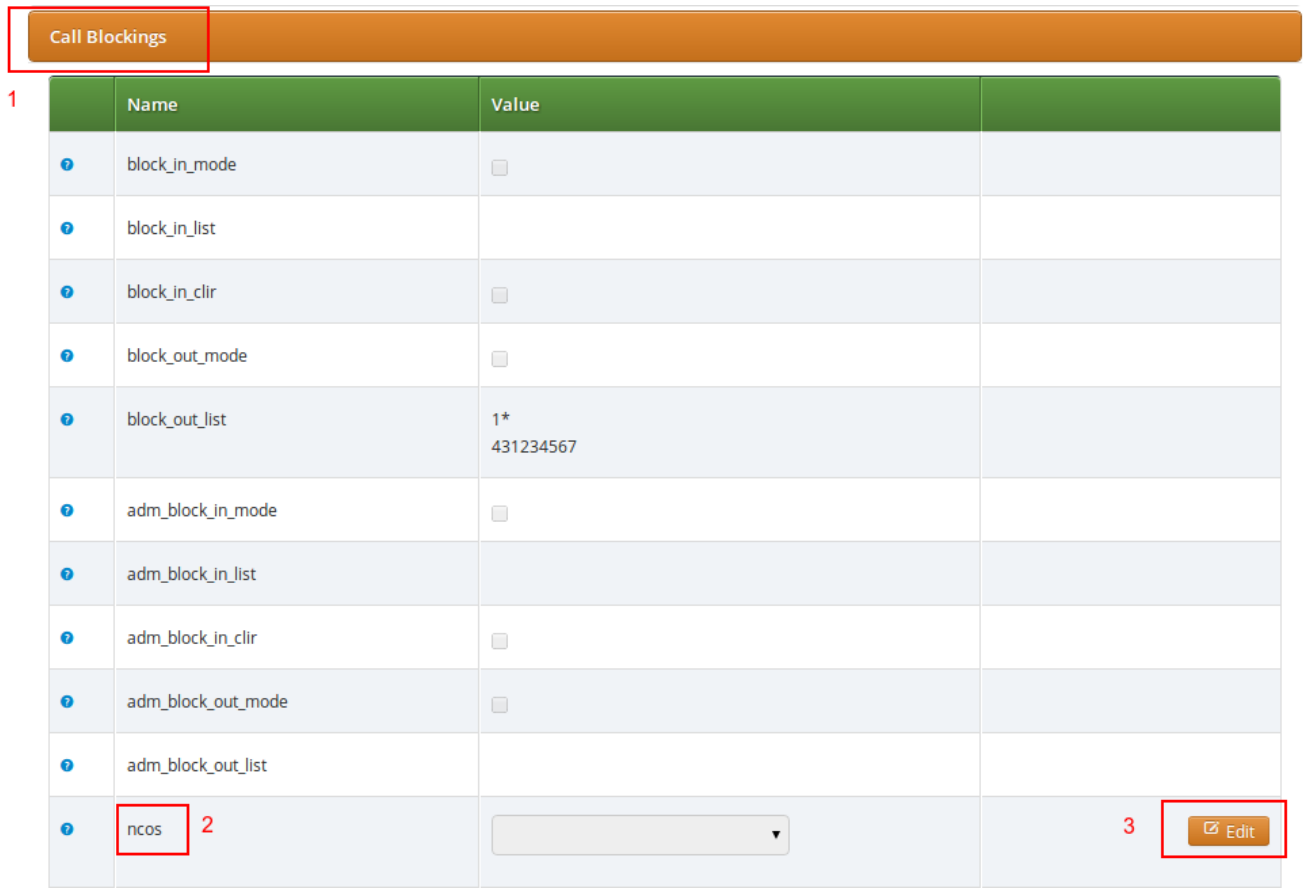


In this example, we block (since the mode of the level is *blacklist*) all numbers starting with `439`. Click the *Save* button to save the entry in the level.

The option *include local area code in list* for a blacklist means that calls within the area code of the subscribers are denied, and for whitelist that they are allowed, respectively. For example if a subscriber has country-code 43 and area-code 1, then selecting this checkbox would result in an implicit entry `^431`.

Assigning NCOS Levels to Subscribers/Domains

Once you've defined your NCOS Levels, you can assign them to local subscribers. To do so, navigate to *Settings→Subscribers*, search for the subscriber you want to edit, press the *Details* button and go to the *Preferences* View. There, press the *Edit* button on either the *ncos* or *adm*ncos *setting in the* Call Blockings__ section.



You can assign the NCOS level to all subscribers within a particular domain. To do so, navigate to *Settings→Domains*, select the domain you want to edit and click *Preferences*. There, press the *Edit* button on either *ncos* or *admin_ncos* in the *Call Blockings* section.

Note: if both domain and subscriber have same NCOS preference set (either *ncos* or *adm_ncos*, or both) the subscriber's preference is used. This is done so that you can override the domain-global setting on the subscriber level.

Assigning NCOS Level for Forwarded Calls to Subscribers/Domains

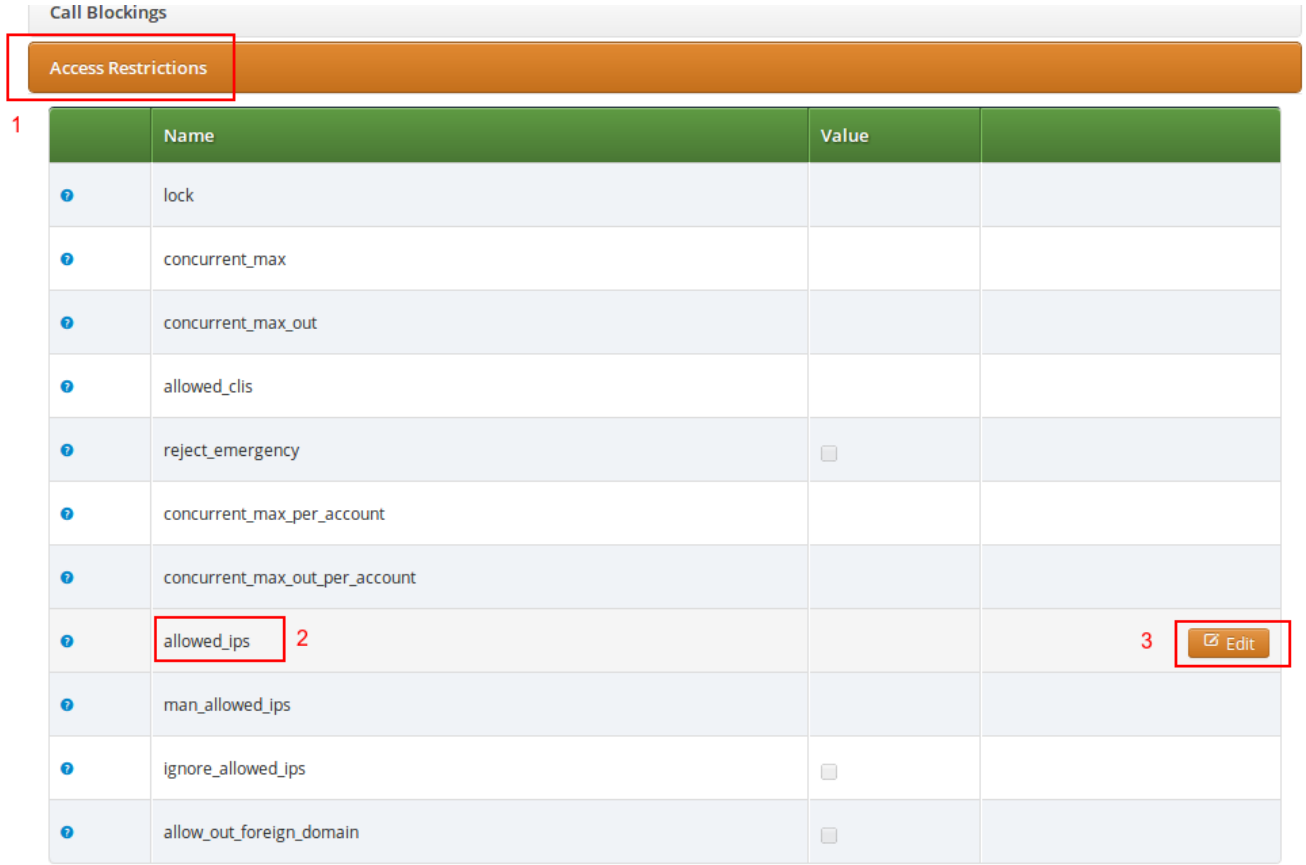
In some countries there are regulatory requirements that prohibit subscribers from forwarding their numbers to special numbers like emergency, police etc. While the sip:carrier does not deny provisioning Call Forward to these numbers, the administrator can prevent the incoming calls from being actually forwarded to numbers defined in the NCOS list: just select the appropriate NCOS level in the domain's or subscriber's preference *adm_cf_ncos*. This NCOS will apply only to the Call Forward from the subscribers and not to the normal outgoing calls from them.

4.1.3 IP Address Restriction

The sip:carrier provides subscriber preference *allowed_ips* to restrict the IP addresses that subscriber is allowed to use the service from. If the REGISTER or INVITE request comes from an IP address that is not in the allowed list, the sip:carrier will reject it with a 403 message. Also a voice message can be played when the call attempt is rejected (if configured).

By default, *allowed_ips* is an empty list which means that subscriber is not restricted. If you want to configure a restriction, navigate

to *Settings→Subscribers*, search for the subscriber you want to edit, press *Details* and then *Preferences* and press *Edit* for the *allowed_ips* preference in the *Access Restrictions* section.



Press the Edit button to the right of empty drop-down list.

You can enter multiple allowed IP addresses or IP address ranges one after another. Click the *Add* button to save each entry in the list. Click the *Delete* button if you want to remove some entry.

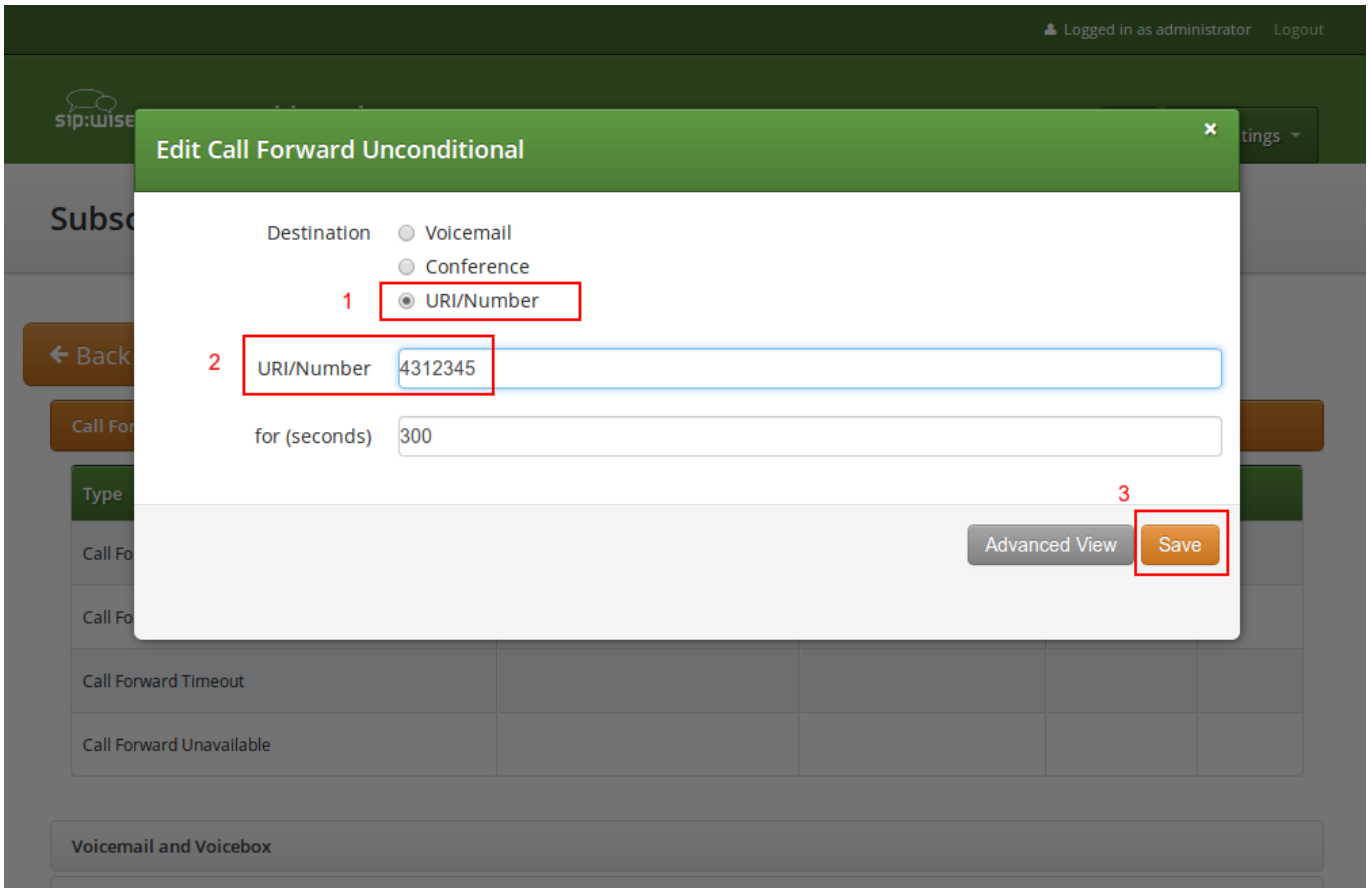
4.2 Call Forwarding and Call Hunting

The sip:carrier provides the capabilities for normal *call forwarding* (deflecting a call for a local subscriber to another party immediately or based on events like the called party being busy or doesn't answer the phone for a certain number of seconds) and *serial call hunting* (sequentially executing a group of deflection targets until one of them succeeds). Targets can be stacked, which means if a target is also a local subscriber, it can have another call forward or hunt group which is executed accordingly.

Call Forwards and Call Hunting Groups can either be executed unconditionally or based on a *Time Set Definition*, so you can define deflections based on time period definitions (e.g. Monday to Friday 8am to 4pm etc).

4.2.1 Setting a simple Call Forward

Go to your *Subscriber Preferences* and click *Edit* on the Call Forward Type you want to set (e.g. *Call Forward Unconditional*).



If you select *URI/Number* in the *Destination* field, you also have to set a *URI/Number*. The timeout defines for how long this destination should be tried to ring.

4.2.2 Advanced Call Hunting

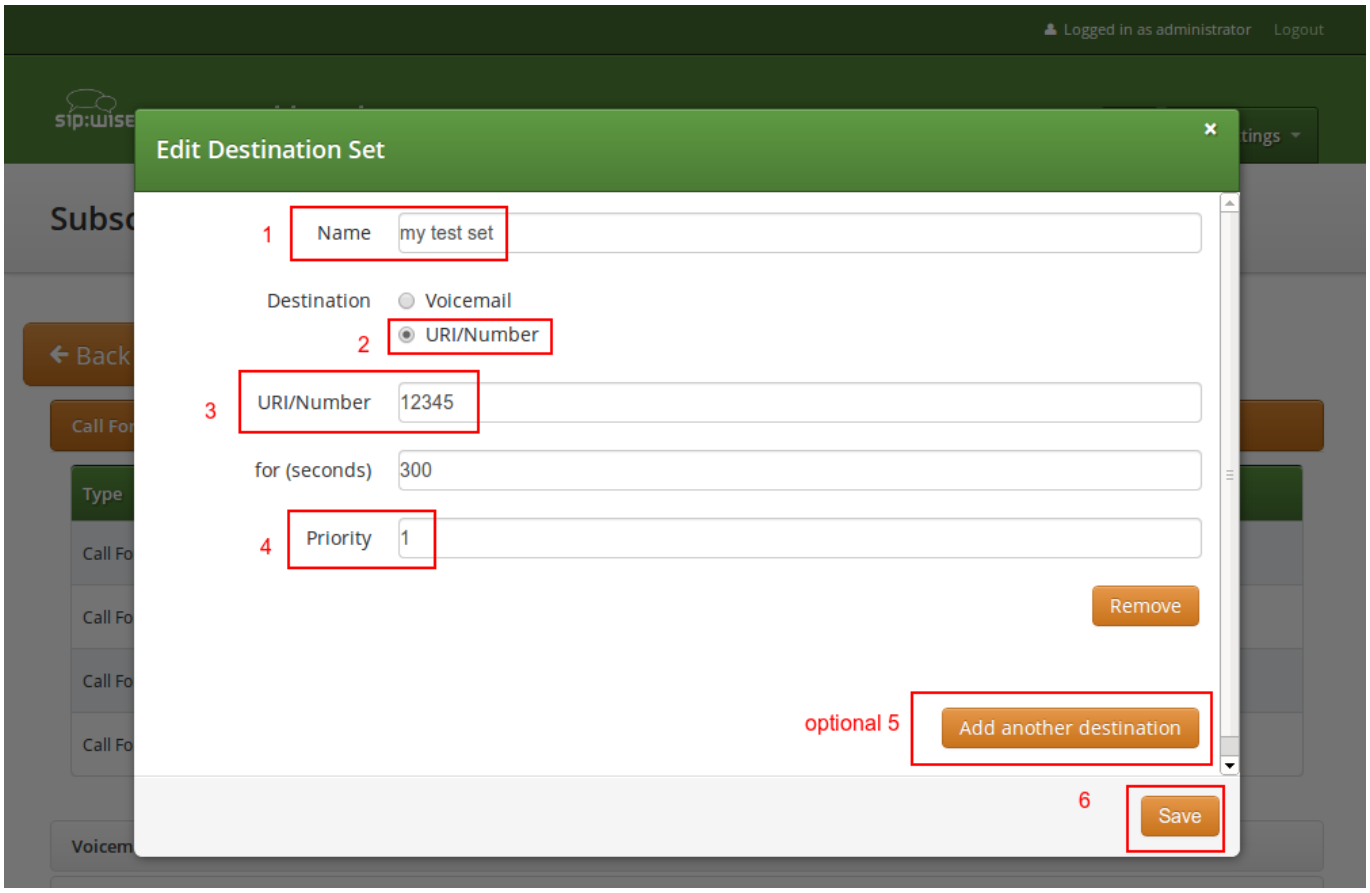
If you want multiple destinations to be executed one after the other, you need to change into the *Advanced View* when editing your call forward. There, you can select multiple *Destination Set*/*Time Set* pairs to be executed.

A *Destination Set* is a list of destinations to be executed one after another.

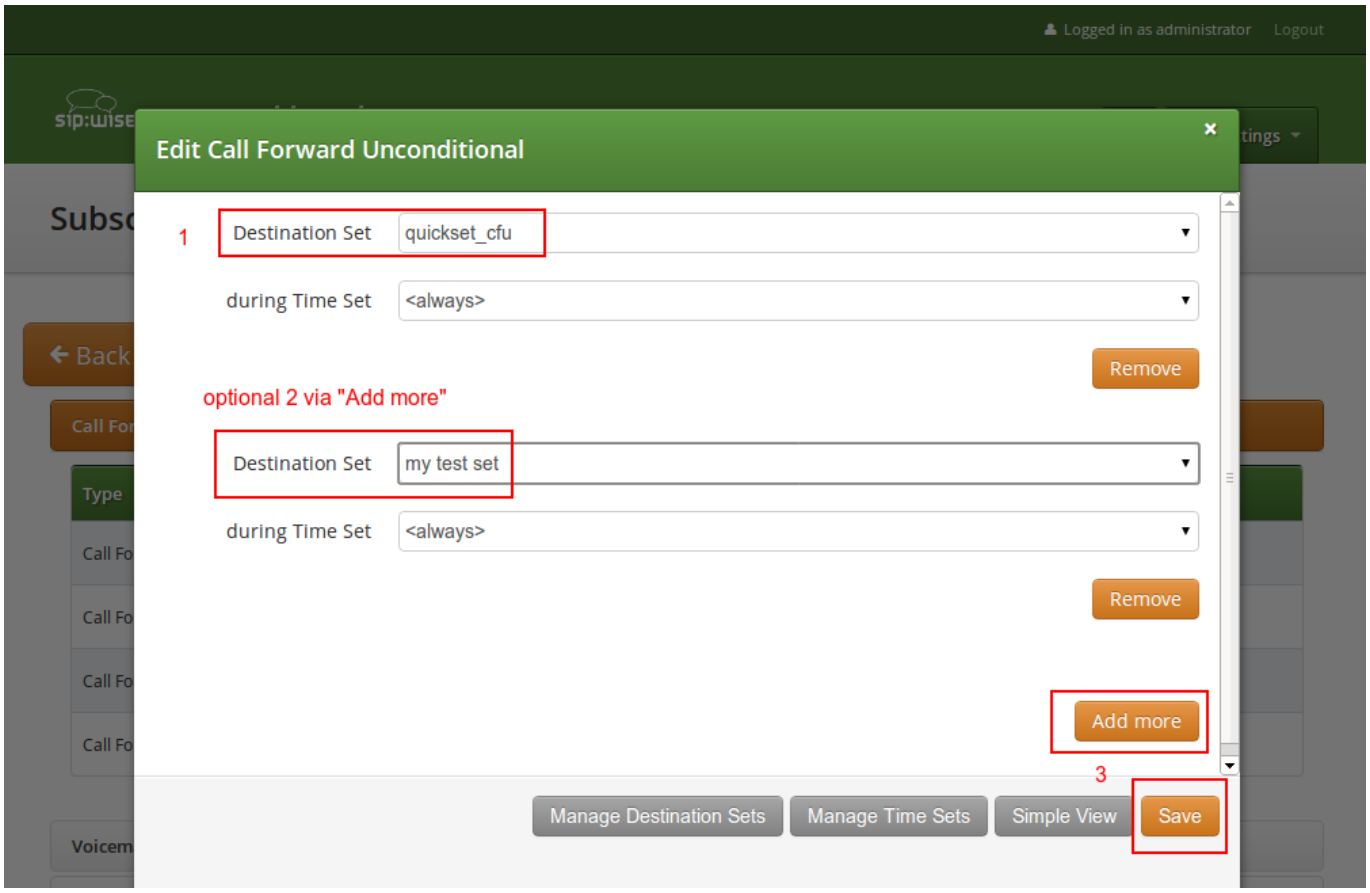
A *Time Set* is a time definition when to execute this *Destination Set*.

Configuring Destination Sets

Click on *Manage Destination Sets* to see a list of available sets. The *quickset_cfu* has been implicitly created during our creation of a simple call forward. You can edit it to add more destinations, or you can create a new destination set.



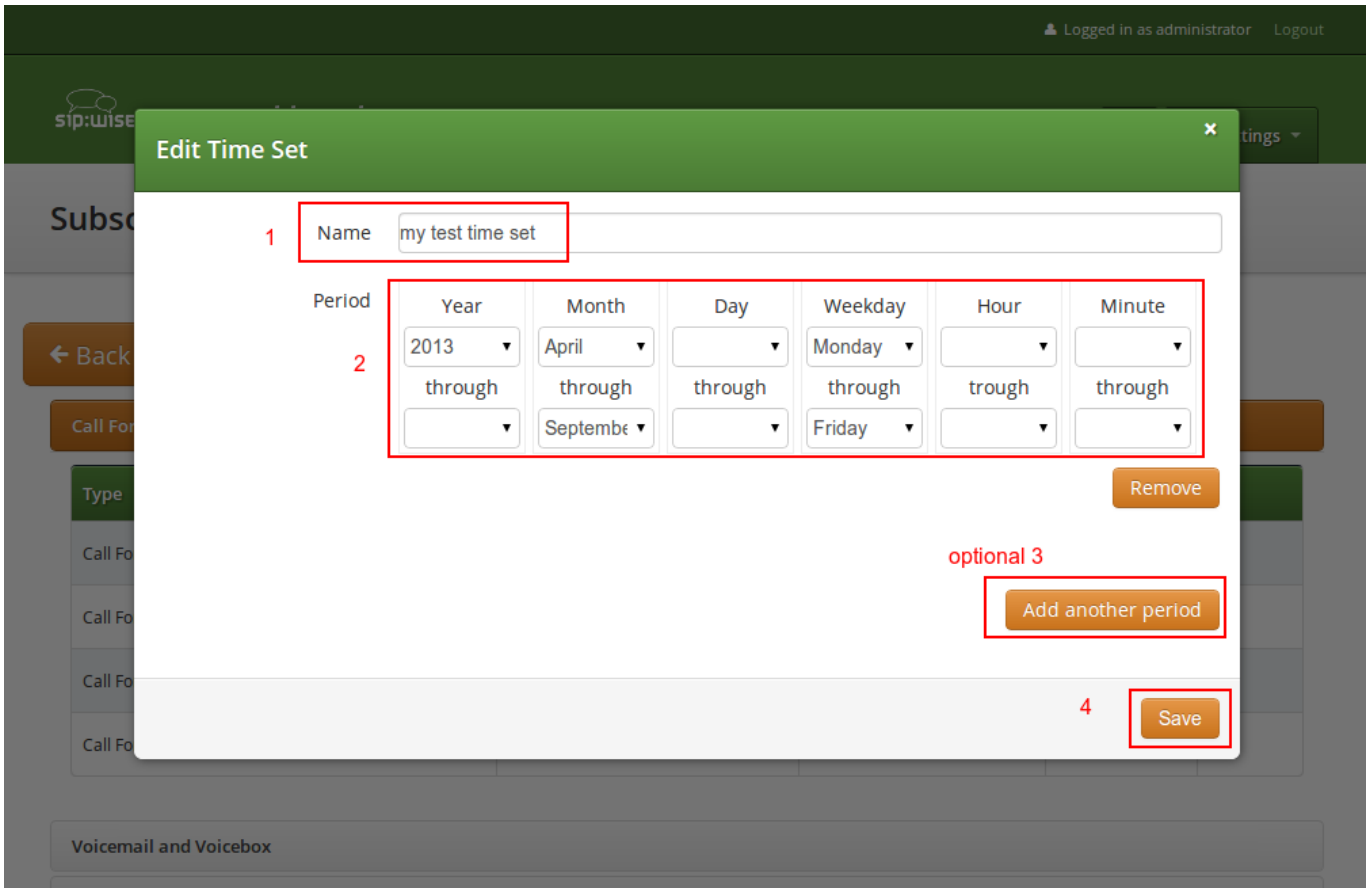
When you close the *Destination Set* Overview, you can now assign your new set in addition or instead of the *quickset_cfu* set.



Press *Save* to store your settings.

Configuring Time Sets

Click on *Manage Time Sets* in the advanced call-forward menu to see a list of available time sets. By default there are none, so you have to create one.



You need to provide a *Name*, and a list of *Periods* where this set is active. If you only set the top setting of a date field (like the *Year* setting in our example above), then it's valid for just this setting (like the full year of *2013* in our case). If you provide the bottom setting as well, it defines a period (like our *Month* setting, which means from beginning of April to end of September).

Important

the period is a *through* definition, so it covers the full range. If you define an *Hour* definition *8-16*, then this means from *08:00* to *16:59:59* (unless you filter the *Minutes* down to something else).

If you close the *Time Sets* management, you can assign your new time set to the call forwards you're configuring.

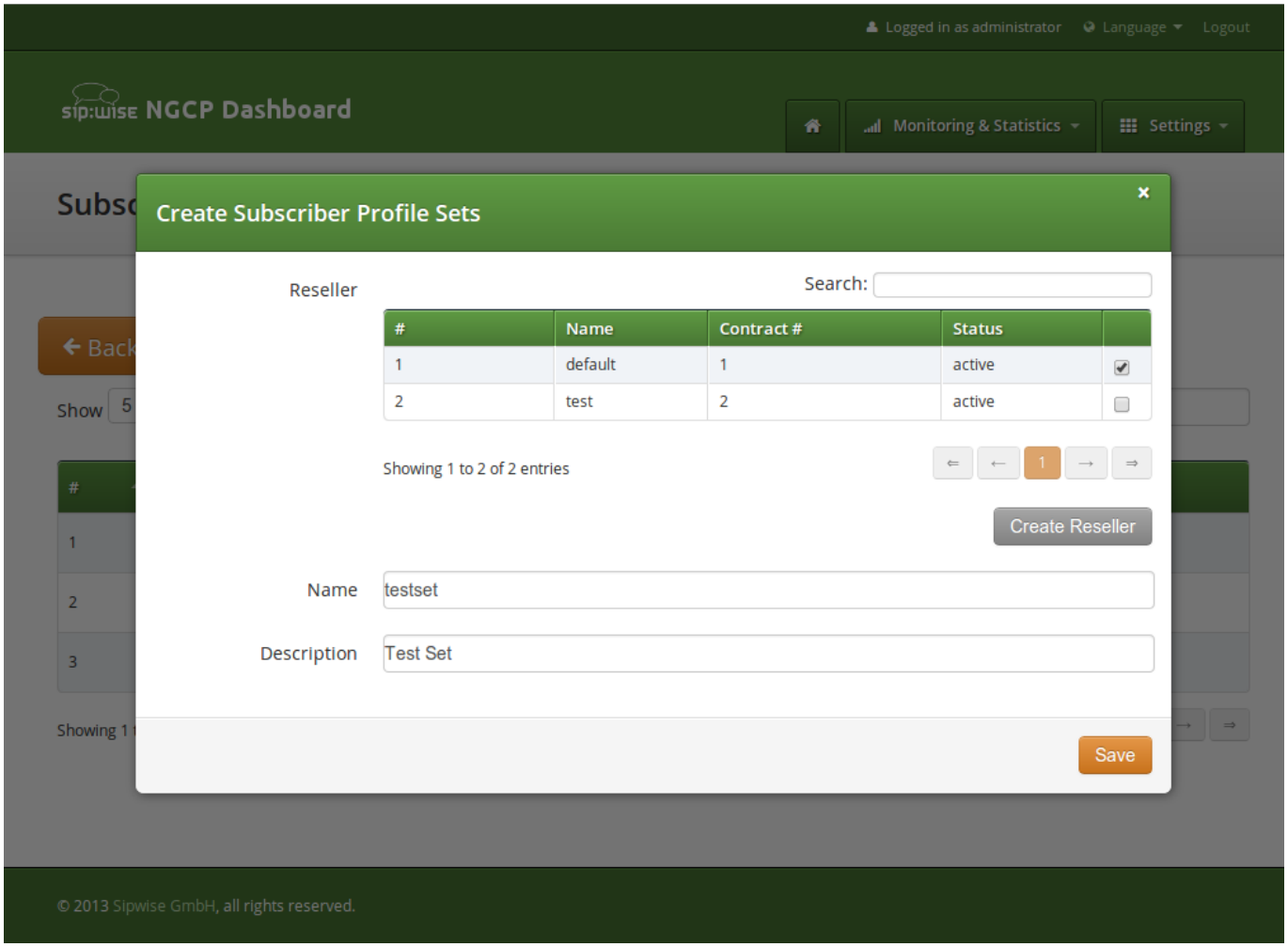
4.3 Limiting Subscriber Preferences via Subscriber Profiles

The preferences a subscriber can provision by himself via the CSC can be limited via profiles within profile sets assigned to subscribers.

4.3.1 Subscriber Profile Sets

Profile sets define containers for profiles. The idea is to define profile sets with different profiles by the administrator (or the reseller, if he is permitted to do so). Then, a subscriber with administrative privileges can re-assign profiles within his profile sets for the subscribers of his customer account.

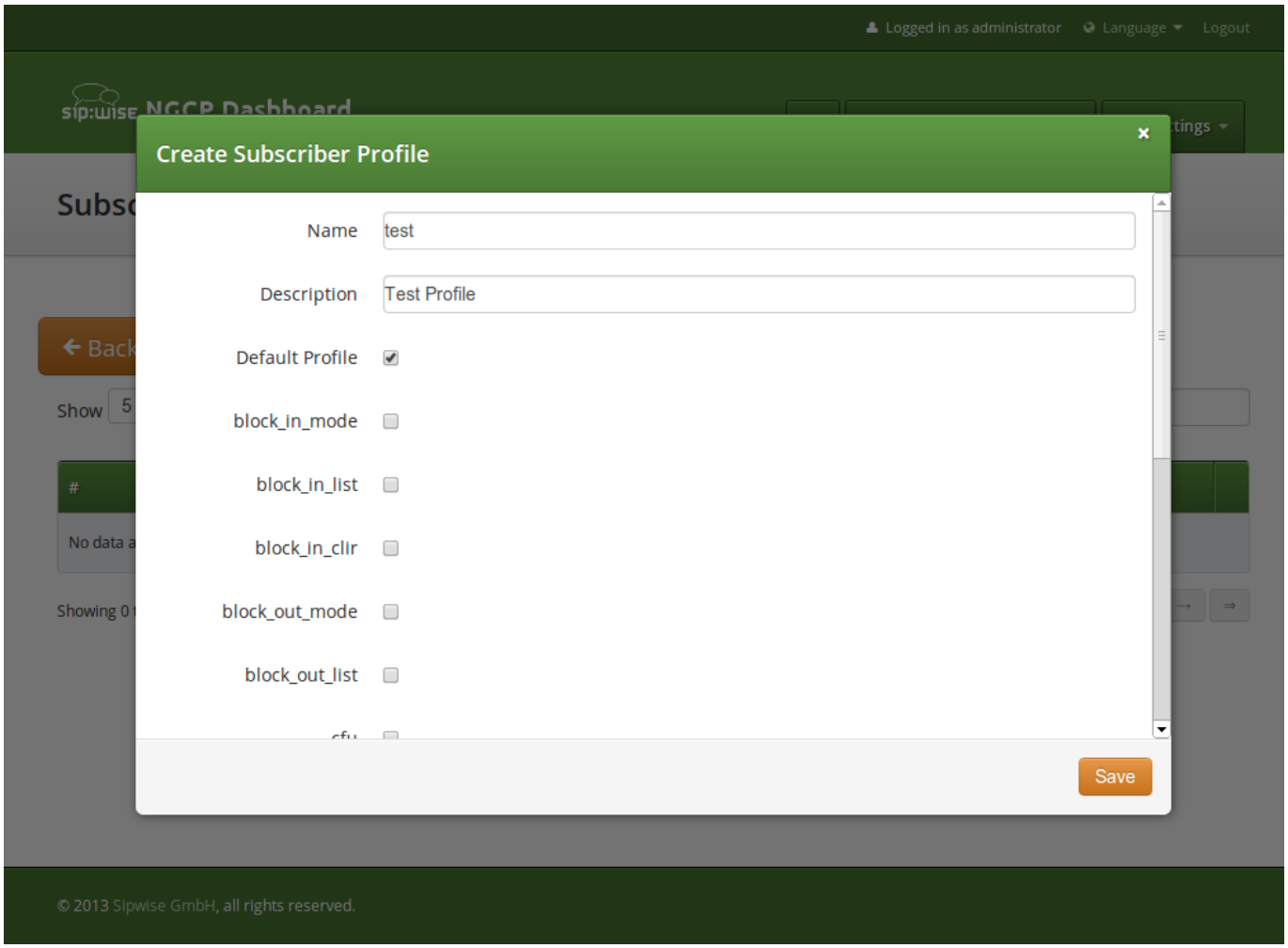
Profile Sets can be defined in *Settings→Subscriber Profiles*. To create a new Profile Set, click *Create Subscriber Profile Set*.



You need to provide a reseller, name and description.

To create Profiles within a Profile Set, hover over the Profile Set and click the *Profiles* button.

Profiles within a Profile Set can be created by clicking the *Create Subscriber Profile* button.



Checking the *Default Profile* option causes this profile to get assigned automatically to all subscribers, who have the profile set assigned. Other options define the user preferences which should be made available to the subscriber.

4.4 Voicemail System

4.4.1 Accessing the IVR Menu

For a subscriber to manage his voicebox via IVR, there are two ways to access the voicebox. One is to call the URI `voicebox@yourdomain` from the subscriber itself, allowing password-less access to the IVR, as the authentication is already done on SIP level. The second is to call the URI `voiceboxpass@yourdomain` from any subscriber, causing the system to prompt for a mailbox and a PIN.

Mapping numbers and codes to IVR access

Since access might need to be provided from external networks like PSTN/Mobile, and since certain SIP phones don't support calling alphanumeric numbers to dial `voicebox`, you can map any arbitrary number to the voicebox URIs using rewrite rules.

To do so, you can provision a match pattern like `^(00|\+)12345$` with a replace pattern `voicebox` or `voiceboxpass` to

map a number to either password-less or password-based IVR access.

External IVR access

When reaching `voiceboxpass`, the subscriber is prompted for her mailbox number and a password. All numbers assigned to a subscriber are valid input (primary number and any alias number). By default, the required format is in E.164, so the subscriber needs to enter the full number including country code, for example `4912345` if she got assigned a German number.

You can globally configure a rewrite rule in `config.yml` using `asterisk.voicemail.normalize_match` and `asterisk.voicemail.normalize_replace`, allowing you to customize the format a subscriber can enter, e.g. having `^0([1-9][0-9]+)$` as match part and `49$1` as replace part to accept German national format.

4.4.2 IVR Menu Structure

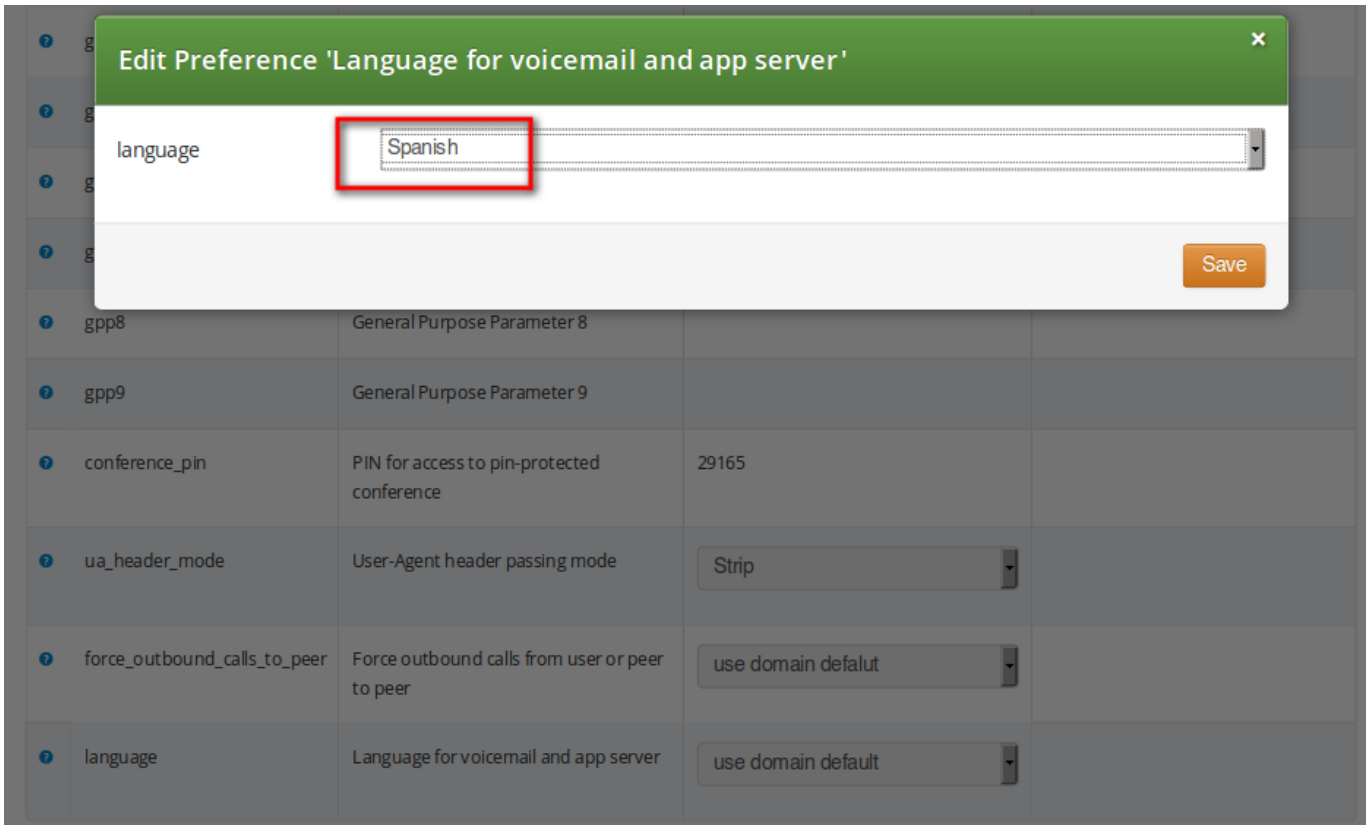
The following list shows you how the voicebox menu is structured.

- *1* Read voicemail messages
  - *3* Advanced options
    - *3* To Hear messages Envelope
    - *** Return to the main menu
  - *4* Play previous message
  - *5* Repeat current message
  - *6* Play next message
  - *7* Delete current message
  - *9* Save message in a folder
    - *0* Save in new Messages
    - *1* Save in old Messages
    - *2* Save in Work Messages
    - *3* Save in Family Messages
    - *4* Save in Friends Messages
    - *#* Return to the main menu
- *2* Change folders
  - *0* Switch to new Messages
  - *1* Switch to old Messages
  - *2* Switch to Work Messages
  - *3* Switch to Family Messages
  - *4* Switch to Friends Messages
  - *#* Get Back

- *3* Advanced Options
  - *** To return to the main menu
- *0* Mailbox options
  - *1* Record your unavailable message
    - *1* accept it
    - *2* Listen to it
    - *3* Rerecord it
  - *2* Record your busy message
    - *1* accept it
    - *2* Listen to it
    - *3* Rerecord it
  - *3* Record your name
    - *1* accept it
    - *2* Listen to it
    - *3* Rerecord it
  - *4* Record your temporary greetings
    - *1* accept it
    - *2* Listen to it
    - *3* Rerecord it
  - *5* Change your password
  - *** To return to the main menu
- *** Help
- *#* Exit

4.5 Configuring Subscriber IVR Language

The language for the Voicemail system IVR or Vertical Service Codes (VSC) IVRs may be set using the subscriber or domain preference *language*.

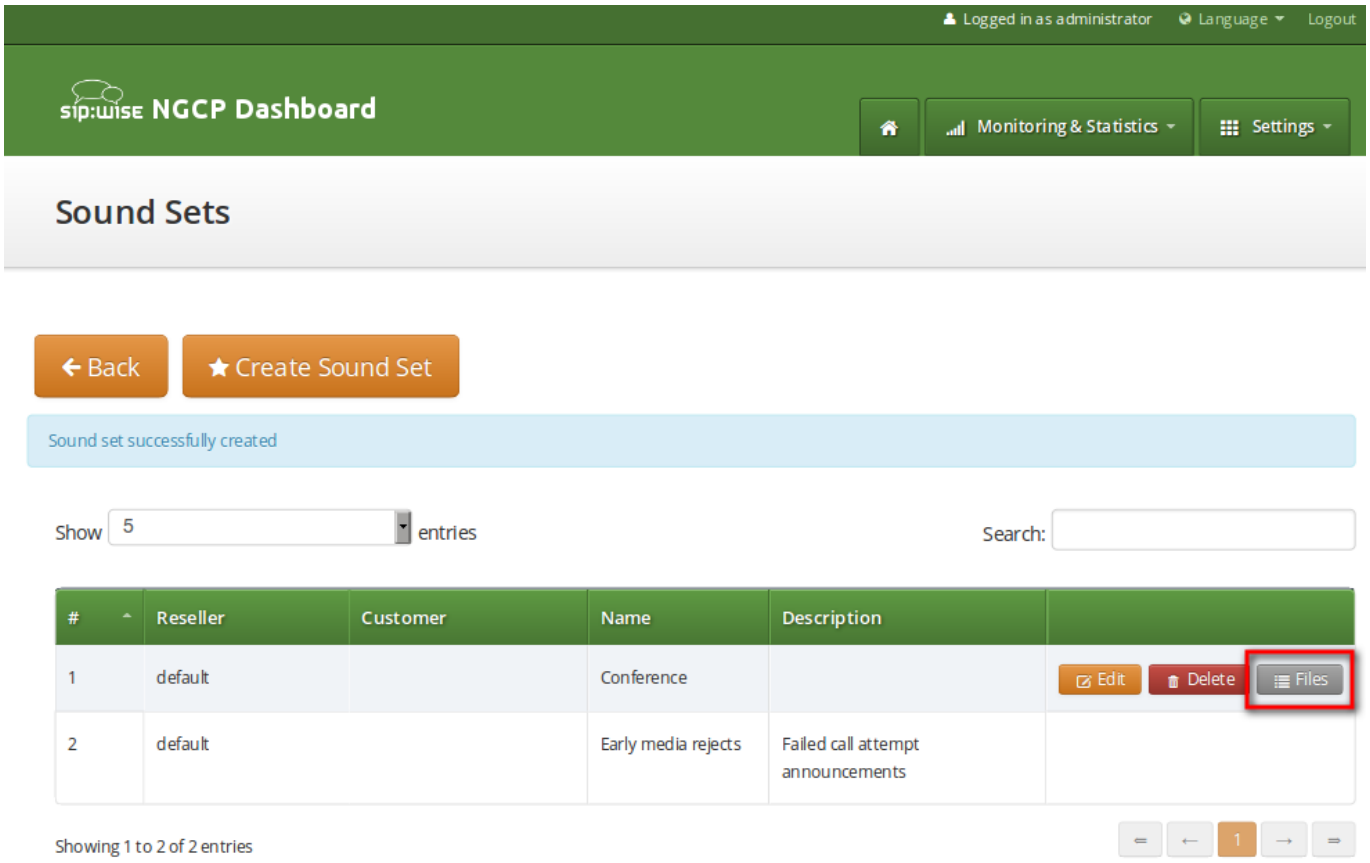


The sip:carrier provides the pre-installed prompts for the Voicemail in the English, Spanish, French and Italian languages and the pre-installed prompts for the Vertical Service Codes IVRs in English only.

The other IVRs such as the Conference system and the error announcements use the Sound Sets configured in NGCP Panel and uploaded by the administrator in his language of choice.

4.6 Sound Sets

The sip:carrier provides the administrator with ability to upload the voice prompts such as conference prompts or call error announcements on the *Sound Sets page*. There is a preference *sound_set* on Domain and Subscriber levels to link subscribers to the sound set that they should hear (as usual the subscriber preference overrides the domain one). Sound Sets can be defined in *Settings→Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button.



Note

You may use 8 or 16 bit mono WAV audio files for all of the voice prompts.

4.6.1 Configuring Early Reject Sound Sets

The call error announcements are grouped under *Early Rejects* section. Unfold the section and click *Upload* next to the sound handles (Names) that you want to use. Choose a WAV file from your file system, and click the Loopplay setting if you want to play the file in a loop instead of just once. Click Save to upload the file.

The call error announcements are played to the user in early media hence the name "Early Reject". If you don't provide the sound files for any handles they will not be used and the sip:carrier will fallback to sending the error response code back to the user.

Table 1: Early Reject Sound Sets

Handle	Description	Message played
block_in	This is what the calling party hears when a call is made from a number that is blocked by the incoming block list (*adm_block_in_list*, *block_in_list* subscriber preferences)	Your call is blocked by the number you are trying to reach.
block_out	This is what the calling party hears when a call is made to a number that is blocked by the outgoing block list (*adm_block_out_list*, *block_out_list* subscriber preferences)	Your call to the number you are trying to reach is blocked.
block_ncos	This is what the calling party hears when a call is made to a number that is blocked by the NCOS level assigned to the subscriber or domain (the NCOS level chosen in *ncos* and *adm_ncos* preferences)	Your call to the number you are trying to reach is not permitted.

Table 1: (continued)

Handle	Description	Message played
block_override_pin_wrong	Announcement played to calling party if it used wrong PIN code to override the outgoing user block list or the NCOS level for this call (the PIN set by *block_out_override_pin* and *adm_block_out_override_pin* preferences)	The PIN code you have entered is not correct.
locked_in	Announcement played on incoming call to a subscriber that is locked for incoming calls	The number you are trying to reach is currently not permitted to receive calls.
locked_out	Announcement played on outgoing call to subscriber that is locked for outgoing calls	You are currently not allowed to place outbound calls.
max_calls_in	Announcement played on incoming call to a subscriber who has exceeded the *concurrent_max* limit by sum of incoming and outgoing calls or whose customer has exceeded the *concurrent_max_per_account* limit by sum of incoming and outgoing calls	The number you are trying to reach is currently busy. Please try again later.
	max_calls_out	Announcement played on outgoing call to a subscriber who has exceeded the *concurrent_max* (total limit) or *concurrent_max_out* (limit on number of outbound calls) or whose customer has exceeded the *concurrent_max_per_account* or *concurrent_max_out_per_account* limit
All outgoing lines are currently in use. Please try again later.	max_calls_peer	Announcement played on calls from the peering if that peer has reached the maximum number of concurrent calls (configured by admin in *concurrent_max* preference of peering server)
The network you are trying to reach is currently busy. Please try again later.	unauth_caller_ip	This is what the calling party hears when it tries to make a call from unauthorized IP address or network (*allowed_ips*, *man_allowed_ips* preferences)

Table 1: (continued)

Handle	Description	Message played
You are not allowed to place calls from your current network location.	relaying_denied	Announcement played on inbound call from trusted IP (e.g. external PBX) with non-local Request-URI domain
The network you are trying to reach is not available.	invalid_speeddial	This is what the calling party hears when it calls an empty speed-dial slot
The speed dial slot you are trying to use is not available.	cf_loop	Announcement played when the called subscriber has the call forwarding configured to itself
The number you are trying to reach is forwarded to an invalid destination.	callee_offline	Announcement played on incoming call to the subscriber which is currently not registered
The number you are trying to reach is currently not available. Please try again later.	callee_busy	Announcement played on incoming call to the subscriber which is currently busy (486 response from the UAS)
The number you are trying to reach is currently busy. Please try again later.	callee_unknown	Announcement that is played on call to unknown or invalid number (not associated with any of our subscribers/hunt groups)
The number you are trying to reach is not in use.	callee_tmp_unavailable	Announcement played on incoming call to the subscriber which is currently unavailable (408, other 4xx or no response code or 30x with malformed contact)
The number you are trying to reach is currently not available. Please try again later.	peering_unavailable	Announcement played in case of outgoing off-net call when there is no peering rule matching this destination and/or source
The network you are trying to reach is not available.	voicebox_unavailable	Announcement played on call to voicebox if the voicemail server is not configured (system operation is impaired)
The voicemail of the number you are trying to reach is currently not available. Please try again later.	emergency_unsupported	Announcement played when emergency destination is dialed but the emergency calls are administratively prohibited for this user or domain (*reject_emergency* preference is enabled)
You are not allowed to place emergency calls from this line. Please use a different phone.	no_credit	Announcement played when prepaid account has insufficient balance to make a call to this destination

4.7 Conference System

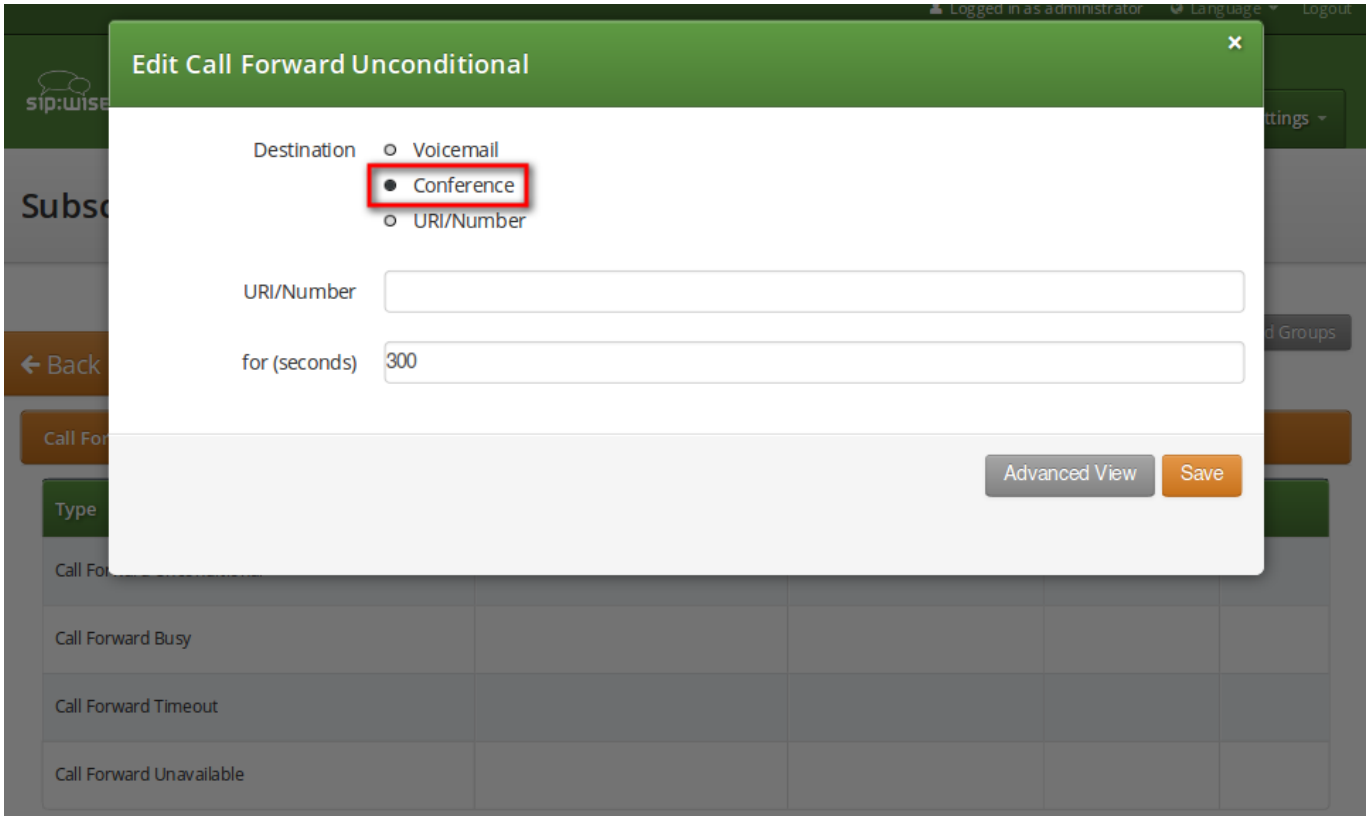
The sip:carrier provides the simple pin-protected conferencing service built using the SEMS DSM scripting language. Hence it is open for all kinds of modifications and extensions.

Template files for the sems conference scripts stored in */etc/ngcp-config/templates/etc/ngcp-sems/*:

- IVR script: /etc/ngcp-config/templates/etc/ngcp-sems/dsm/confpin.dsm.tt2
- Config: /etc/ngcp-config/templates/etc/ngcp-sems/dsm/confpin.conf.tt2

4.7.1 Configuring Call Forward to Conference

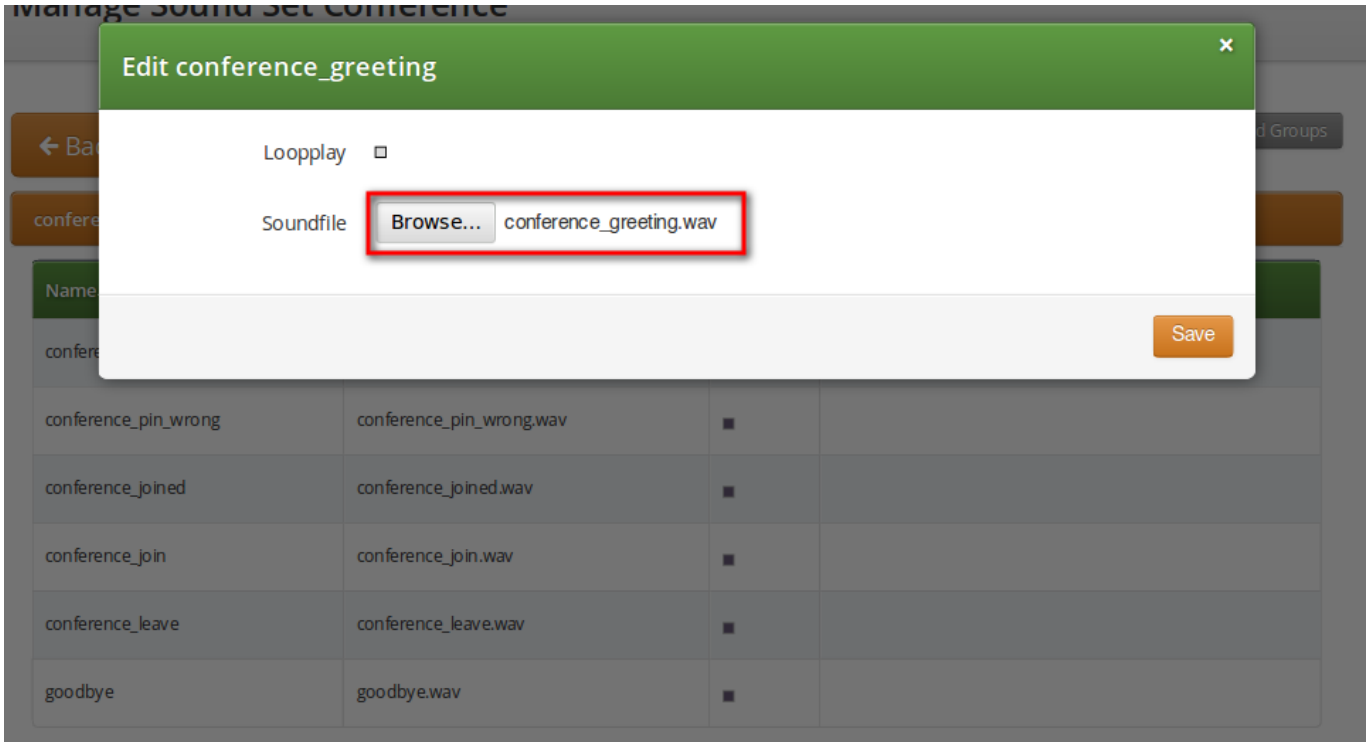
Go to your *Subscriber Preferences* and click *Edit* on the Call Forward Type you want to set (e.g. *Call Forward Unconditional*).



You should select *Conference* option in the *Destination* field and leave the *URI/Number* empty. The timeout defines for how long this destination should be tried to ring.

4.7.2 Configuring Conference Sound Sets

Sound Sets can be defined in *Settings→Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button.



Upload the following files:

Table 2: Conference Sound Sets

Handle	Message played
conference_greeting	Welcome to the conferencing service. Please enter your PIN, followed by the pound key.
conference_pin_wrong	You have entered an invalid PIN number. Please try again.
conference_joined	You will be placed into the conference.
conference_join	A person has joined the conference.
conference_leave	A person has left the conference.
goodbye	Goodbye.

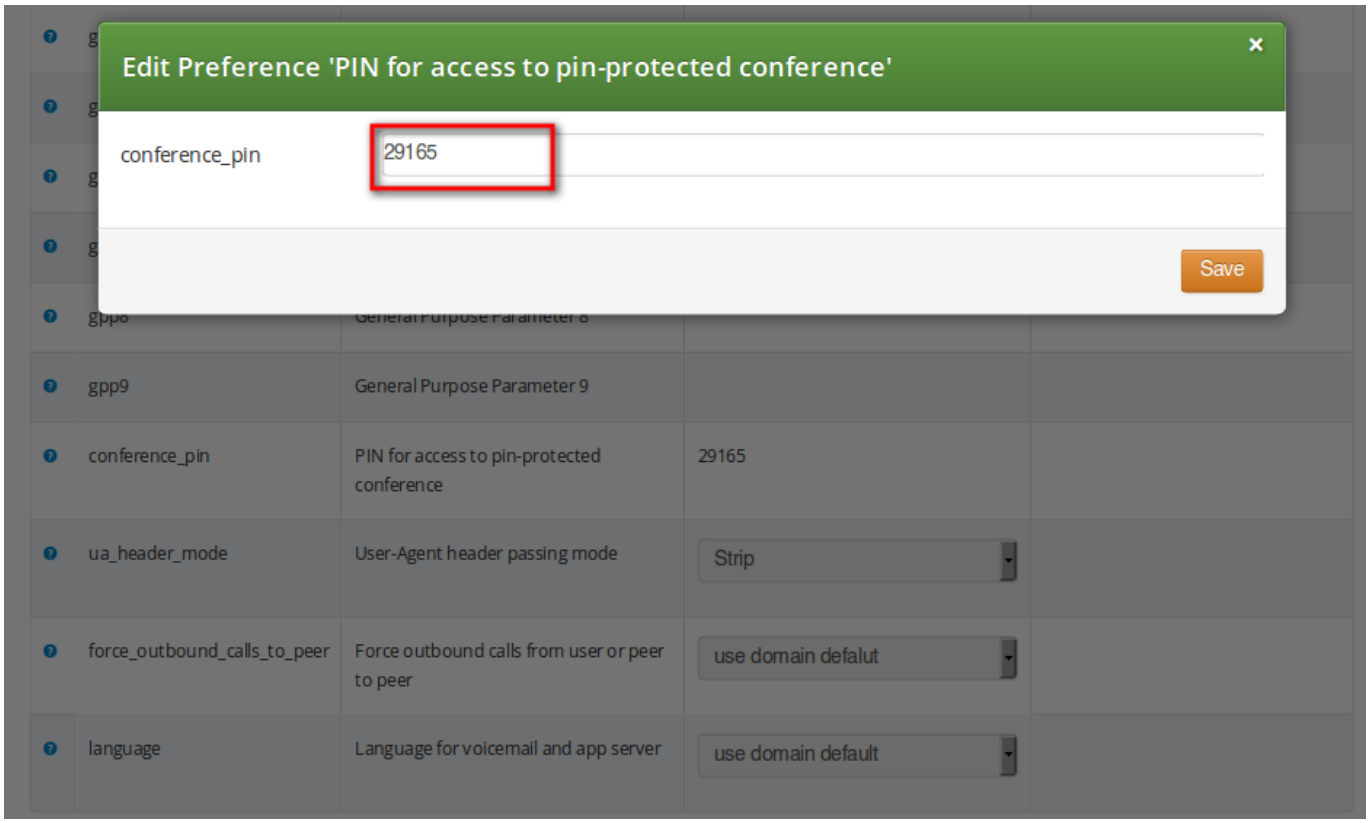
Note

You may use 8 or 16 bit mono WAV audio files.

Then set the preference *sound_set* on the Domain or Subscriber level in order to assign the Sound Set you have just created to the subscriber (as usual the subscriber preference overrides the domain one).

4.7.3 Entering the Conference with a PIN

It is mandatory to configure the PIN code for entrance to the conference on the same subscriber which has the Call Forwarding active. Responsible for this is the *conference_pin* preference in the Internals section of subscriber preferences.



When calling the conference IVR you are requested to enter this PIN. Upon the successful entry of the PIN the caller hears the announcement that he is going to be placed into a conference and at the same time this is announced to all participants in the conference.

5 Customer Self-Care Interfaces

There are two ways for end users to maintain their subscriber settings: via the *Customer Self-Care Web Interface* and via *Vertical Service Codes* using their SIP phones.

5.1 The Customer Self-Care Web Interface

The NGCP provides a web panel for end users (CSC panel) to maintain their subscriber accounts, which is running on *https://<ce-ip>*. Every subscriber can log in there, change subscriber feature settings, view their call lists, retrieve voicemail messages and trigger calls using the click-to-dial feature.

5.1.1 Login Procedure

To log into the CSC panel, the end user has to provide his full web username (e.g. `user1@1.2.3.4`) and the web password defined in Section 3.2. Once logged in, he can change his web password in the *Account* section. This will NOT change his SIP password, so if you control the end user devices, you can auto-provision the SIP password into the device and keep it secret, and just hand over the web password to the customer. This way, the end user will only be able to place calls with this auto-provisioned device and not with an arbitrary soft-phone, but can nonetheless manage his account via the CSC panel.

Important

You can simplify the login procedure for one SIP domain in such a way that users in this domain only need to pass the user part (e.g. `user1`) as a username instead of the full web username to log in by setting the parameter *www_csc→site_domain* in the config file */etc/ngcp-config/config.yml* to the corresponding domain (e.g. `1.2.3.4`) and execute `ngcpcfg apply`.

5.1.2 Site Customization

As an operator, you can change the appearance of the CSC panel by modifying a couple of parameters in the section *www_csc→site_config* of the config file */etc/ngcp-config/config.yml*. Modify the site title, your company details and the logo to reflect your use case.

You can also enable/disable specific languages a user can choose from in the CSC panel. Currently, English (`en`), French (`fr`), German (`de`) and Spanish (`es`) are supported and English is activated by default.

After changing one or more of the parameters in this file, execute `ngcpcfg apply` to activate the changes.

5.2 The Vertical Service Code Interface

Vertical Service Codes (VSC) are codes a user can dial on his phone to provision specific features for his subscriber account. The format is `*<code>*<value>` to activate a specific feature, and `#<code>` or `#<code>#` to deactivate it. The *code* parameter is a two-digit code, e.g. `72`. The *value* parameter is the value being set for the corresponding feature.

Important

The *value* user input is normalized using the Rewrite Rules Sets assigned to domain as described in Section 3.6.

By default, the following codes are configured for setting features. The examples below assume that there is a domain rewrite rule normalizing the number format *0<ac><sn>* to *<cc><ac><sn>* using *43* as country code.

- **72** - enable *Call Forward Unconditional* e.g. to 431000 by dialing `*72*01000`, and disable it by dialing `#72`.
- **90** - enable *Call Forward on Busy* e.g. to 431000 by dialing `*90*01000`, and disable it by dialing `#90`.
- **92** - enable *Call Forward on Timeout* e.g. after 30 seconds of ringing to 431000 by dialing `*92*30*01000`, and disable it by dialing `#92`.
- **93** - enable *Call Forward on Not Available* e.g. to 431000 by dialing `*93*01000`, and disable it by dialing `#93`.
- **50** - set *Speed Dial Slot*, e.g. set slot 1 to 431000 by dialing `*50*101000`, which then can be used by dialing `*1`.
- **55** - set *One-Shot Reminder Call* e.g. to 08:30 by dialing `*55*0830`.
- **31** - set *Calling Line Identification Restriction* for one call, e.g. to call 431000 anonymously dial `*31*01000`.
- **80** - call using *Call Block Override PIN*, number should be prefixed with a block override PIN configured in admin panel to disable the outgoing user/admin block list and NCOS level for a call. For example, when override PIN is set to 7890, dial `*80*789001000` to call 431000 bypassing block lists.

You can change any of the codes (but not the format) in */etc/ngcp-config/config.yml* in the section *sems→vsc*. After the changes, execute `ngcpcfg apply`.

Caution

If you have the EMTAs under your control, make sure that the specified VSCs don't overlap with EMTA-internal VSCs, because the VSC calls must be sent to the NGCP via SIP like normal telephone calls.

5.3 The Voicemail Interface

NGCP offers several ways to access the Voicemail box.

The CSC panel allows your users to listen to voicemail messages from the web browser, delete them and call back the user who left the voice message. User can setup voicemail forwarding to the external email and the PIN code needed to access the voicebox from any telephone also from the CSC panel.

To manage the voice messages from SIP phone: simply dial internal voicemail access number `2000`.

To change the access number: look for the parameter *voicemail_number* in */etc/ngcp-config/config.yml* in the section *sems→vsc*. After the changes, execute `ngcpcfg apply`.

Tip

To let the callers leave a voice message when user is not available he should enable Call Forward to Voicebox. The Call Forward can be provisioned from the CSC panel as well as by dialing Call Forward VSC with the voicemail number. E.g. when parameter *voicemail_number* is set to *9999*, a Call Forward on Not Available to the Voicebox is set if the user dials `*93*9999`. As a result, all calls will be redirected to the Voicebox if SIP phone is not registered.

To manage the voice messages from any phone:

- As an operator, you can setup some DID number as external voicemail access number: for that, you should add a special rewrite rule (Inbound Rewrite Rule for Callee, see Section 3.6.) on the incoming peer, to rewrite that DID to "voiceboxpass". Now when user calls this number the call will be forwarded to the voicemail server and he will be prompted for mailbox and password. The mailbox is the full E.164 number of the subscriber account and the password is the PIN set in the CSC panel.
- The user can also dial his own number from PSTN, if he setup Call Forward on Not Available to the Voicebox, and when reaching the voicemail server he can interrupt the "user is unavailable" message by pressing * key and then be prompted for the PIN. After entering PIN and confirming with # key he will enter own voicemail menu. PIN is random by default and must be kept secret for that reason.

6 Billing Configuration

This chapter describes the steps necessary to rate calls and export rated CDRs (call detail records) to external systems.

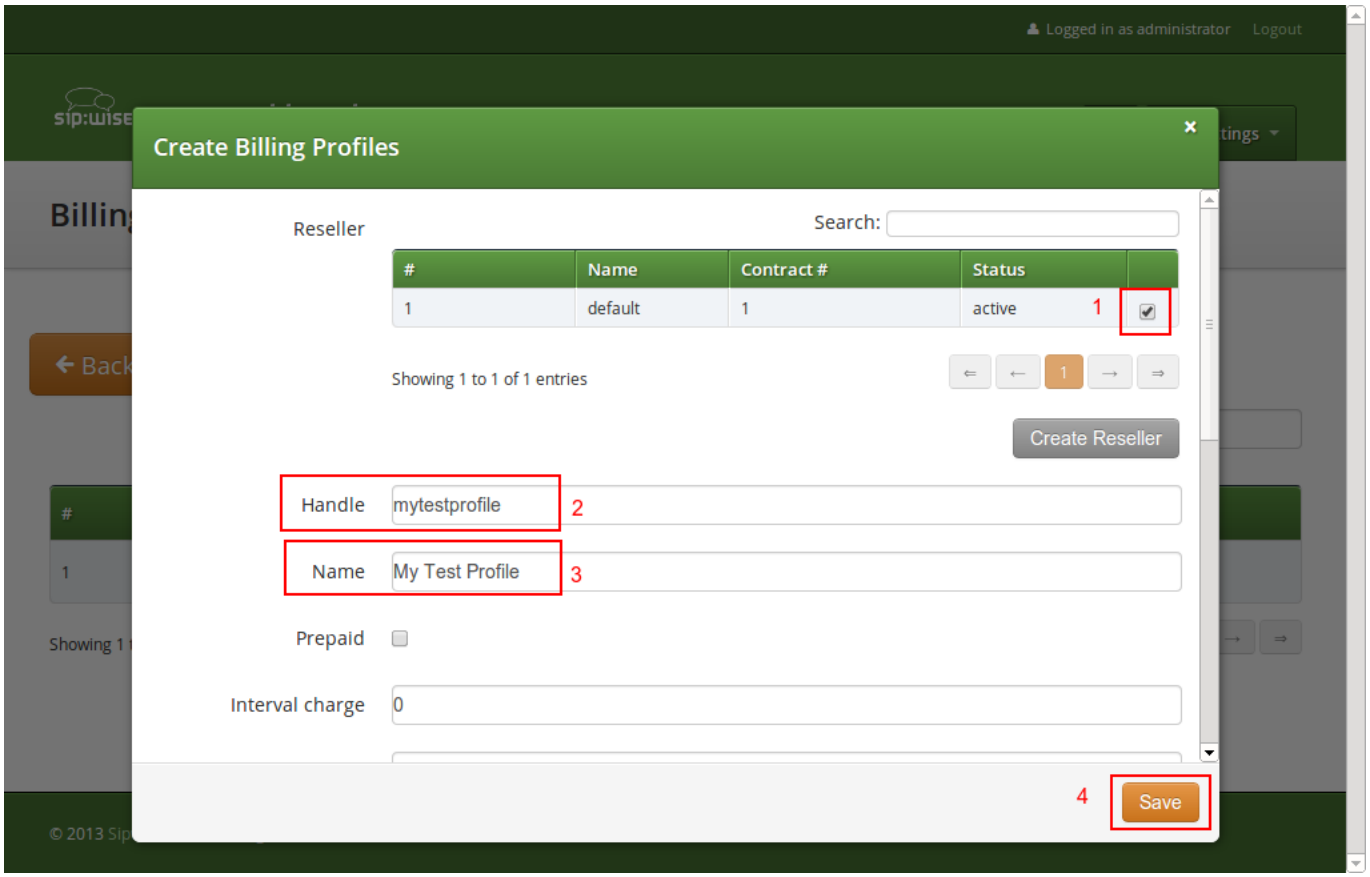
6.1 Billing Data Import

Service billing on the NGCP is based on billing profiles, which may be assigned to VoIP accounts and SIP peerings. The design focuses on a simple, yet flexible approach, to support arbitrary dial-plans without introducing administrative overhead for the system administrators. The billing profiles may define a base fee and free time or free money per billing interval. Unused free time or money automatically expires at the end of the billing interval.

Each profile may have call destinations (usually based on E.164 number prefix matching) with configurable fees attached. Call destination fees each support individual intervals and rates, with a different duration and/or rate for the first interval. (e.g.: charge the first minute when the call is opened, then every 30 seconds, or make it independent of the duration at all) It is also possible to specify different durations and/or rates for peak and off-peak hours. Peak time may be specified based on weekdays, with additional support for manually managed dates based on calendar days. The call destinations can finally be grouped for an overview on user's invoices by specifying a zone in two detail levels. (E.g.: national landline, national mobile, foreign 1, foreign 2, etc.)

6.1.1 Creating Billing Profiles

The first step when setting up billing data is to create a billing profile, which will be the container for all other billing related data. Go to *Settings→Billing* and click on *Create Billing Profile*.



The fields *Reseller*, *Handle* and *Name* are mandatory.

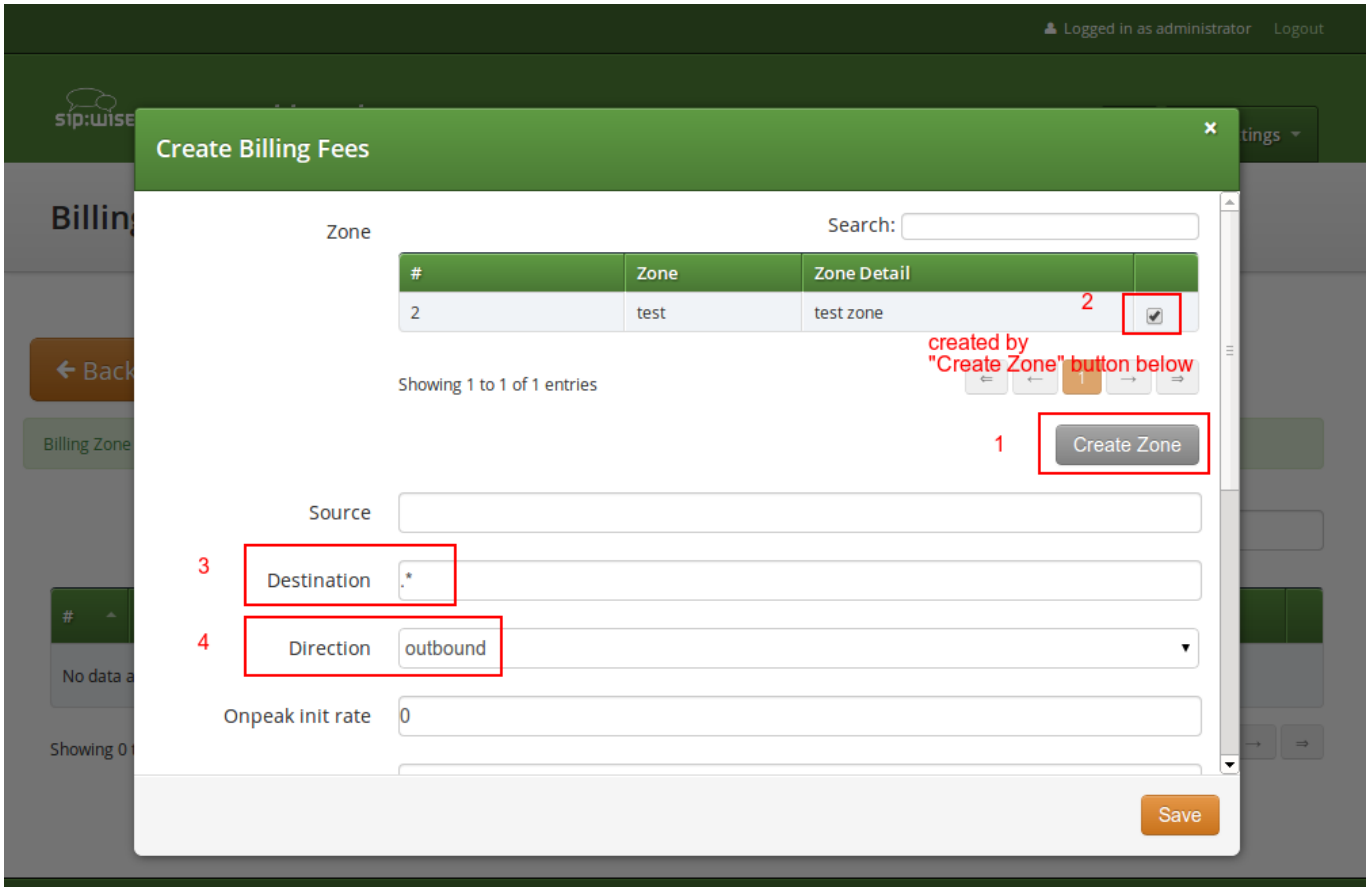
- **Reseller**: The reseller this billing profile belongs to.
- **Handle**: A unique, permanently fixed string which is used to attach the billing profile to a VoIP account or SIP peering contract.
- **Name**: A free form string used to identify the billing profile in the *Admin Panel*. This may be changed at any time.
- **Interval charge**: A base fee for the billing interval, specifying a monetary amount (represented as a floating point number) in whatever currency you want to use.
- **Interval free time**: If you want to include free calling time in your billing profile, you may specify the number of seconds that are available every billing interval. See *Creating Billing Fees* below on how to select destinations which may be called using the free time.
- **Interval free cash**: Same as for *interval free time* above, but specifies a monetary amount which may be spent on outgoing calls. This may be used for example to implement a minimum turnover for a contract, by setting the *interval charge* and *interval free cash* to the same values.
- **Fraud monthly limit**: The monthly fraud detection limit (in Cent) for accounts with this billing profile. If the call fees of an account reach this limit within a billing interval, an action can be triggered.
- **Fraud monthly lock**: a choice of *none, foreign, outgoing, incoming, global*. Specifies a lock level which will be used to lock the account and his subscribers when *fraud monthly limit* is exceeded.
- **Fraud monthly notify**: An email address or comma-separated list of email addresses that will receive notifications when *fraud monthly limit* is exceeded.

- **Fraud daily limit**: The fraud detection limit (in Cent) for accounts with this billing profile. If the call fees of an account reach this limit within a calendar day, an action can be triggered.
- **Fraud daily lock**: a choice of *none, foreign, outgoing, incoming, global*. Specifies a lock level which will be used to lock the account and his subscribers when *fraud daily limit* is exceeded.
- **Fraud daily notify**: An email address or comma-separated list of email addresses that will receive notifications when *fraud daily limit* is exceeded.
- **Currency**: The currency symbol for your currency. Any UTF-8 character may be used and will be printed in web interfaces.
- **VAT rate**: The percentage of value added tax for all fees in the billing profile. Currently for informational purpose only and not used further.
- **VAT included**: Whether VAT is included in the fees entered in web forms or uploaded to the platform. Currently for informational purpose only and not used further.

6.1.2 Creating Billing Fees

Each *Billing Profile* holds multiple *Billing Fees*.

To set up billing fees, click on the *Fees* button of the billing profile you want to configure. Billing fees may be uploaded using a configurable CSV file format, or entered directly via the web interface by clicking *Create Fee Entry*. To configure the CSV field order for the file upload, rearrange the entries in the *www_admin→fees_csv→element_order* array in */etc/ngcp-config/config.yml* and execute the command `ngcpcfg apply`. For input via the web interface, just fill in the text fields accordingly.



In both cases, the following information may be specified independently for every destination:

- **Zone**: A zone for a group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. `foreign zone 1`)
- **Source**: The source pattern. This is a POSIX regular expression matching the complete source URI (e.g. `^.*@sip\.example\.org$` or `^someone@sip\.sipwise\.com$` or just `.` to match everything). If you leave this field empty, the default pattern `.` matching everything will be set implicitly. Internally, this pattern will be matched against the `<source_cli>@<source_domain>` fields of the CDR.
- **Destination**: The destination pattern. This is a POSIX regular expression matching the complete destination URI (e.g. `someone@sip\.example\.org` or `^43`). This field must be set.
- **Direction**: `Outbound` for standard origination fees (applies to callers placing a call and getting billed for that) or `Inbound` for termination fees (applies to callees if you want to charge them for receiving various calls, e.g. for 800-numbers). *If in doubt, use Outbound*. If you upload fees via CSV files, use `out` or `in`, respectively.

Important

The {source, destination, direction} combination needs to be unique for a billing profile. The system will return an error if such a set is specified twice, both for the file upload and the input via the web interface.

Important

There are several internal services (vsc, conference, voicebox) which will need a specific destination entry with a domain-based destination. If you don't want to charge the same (or nothing) for those services, add a fee for destination `\.local$` there. If you want to charge different amounts for those services, break it down into separate fee entries for `@vsc\.local$`, `@conference\.local$` and `@voicebox\.local$` with the according fees. **NOT CREATING EITHER THE CATCH-ALL FEE OR THE SEPARATE FEES FOR THE `.local` DOMAIN WILL BREAK YOUR RATING PROCESS!**

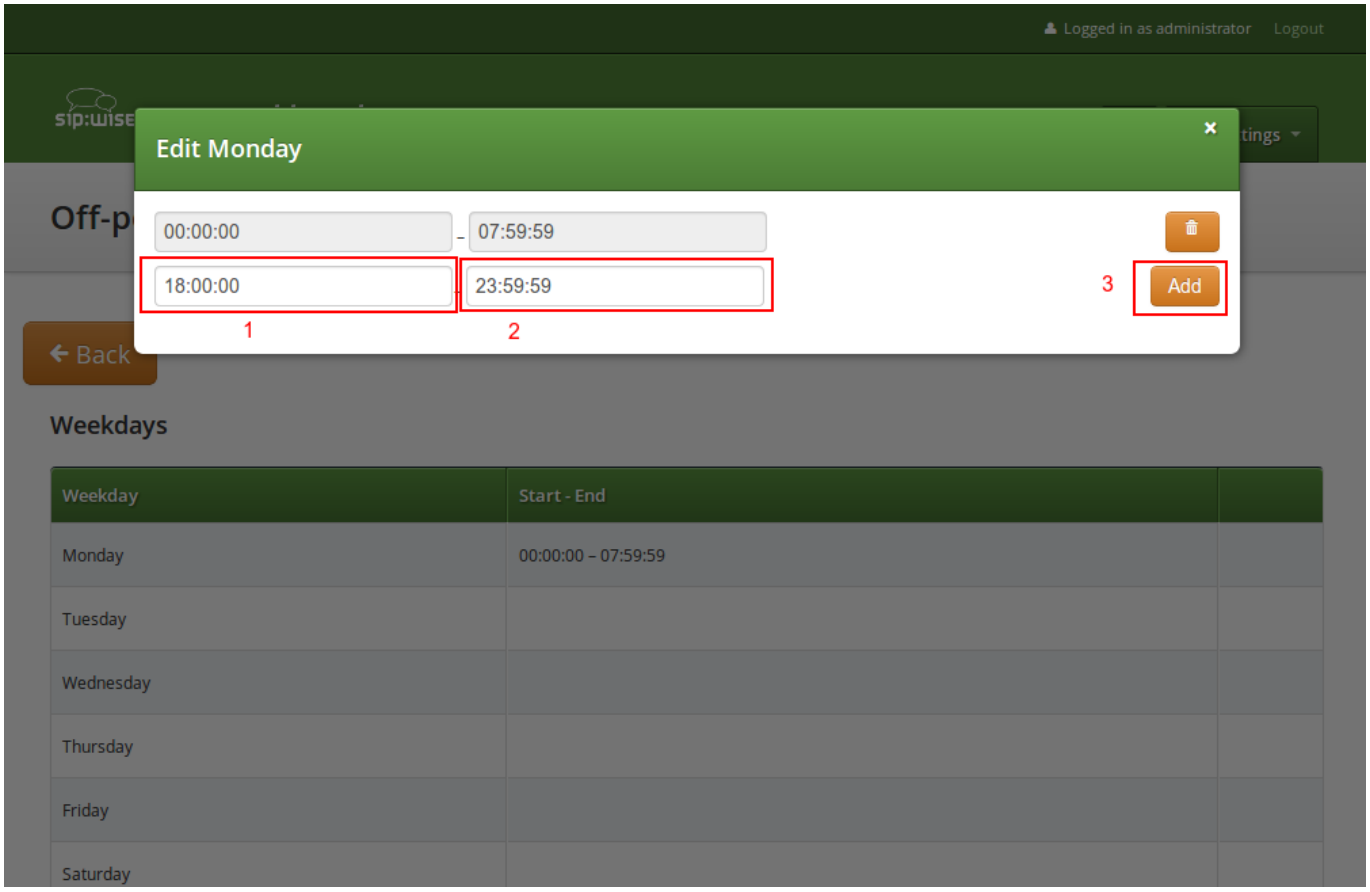
- **Onpeak init rate**: The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours.
- **Onpeak init interval**: The duration of the first billing interval, in seconds. Applicable to calls during onpeak hours.
- **Onpeak follow rate**: The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours. Defaults to *onpeak init rate*.
- **Onpeak follow interval**: The duration of subsequent billing intervals, in seconds. Applicable to calls during onpeak hours. Defaults to *onpeak init interval*.
- **Offpeak init rate**: The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *onpeak init rate*.
- **Offpeak init interval**: The duration of the first billing interval, in seconds. Applicable to calls during off-peak hours. Defaults to *onpeak init interval*.

- **Offpeak follow rate**: The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *offpeak init rate* if that one is specified, or to *onpeak follow rate* otherwise.
- **Offpeak follow interval**: The duration of subsequent billing intervals, in seconds. Applicable to calls during off-peak hours. Defaults to *offpeak init interval* if that one is specified, or to *onpeak follow interval* otherwise.
- **Use free time**: Specifies whether free time minutes may be used when calling this destination. May be specified in the file upload as 0, n[o], f[alse] and 1, y[es], t[rue] respectively.

6.1.3 Creating Off-Peak Times

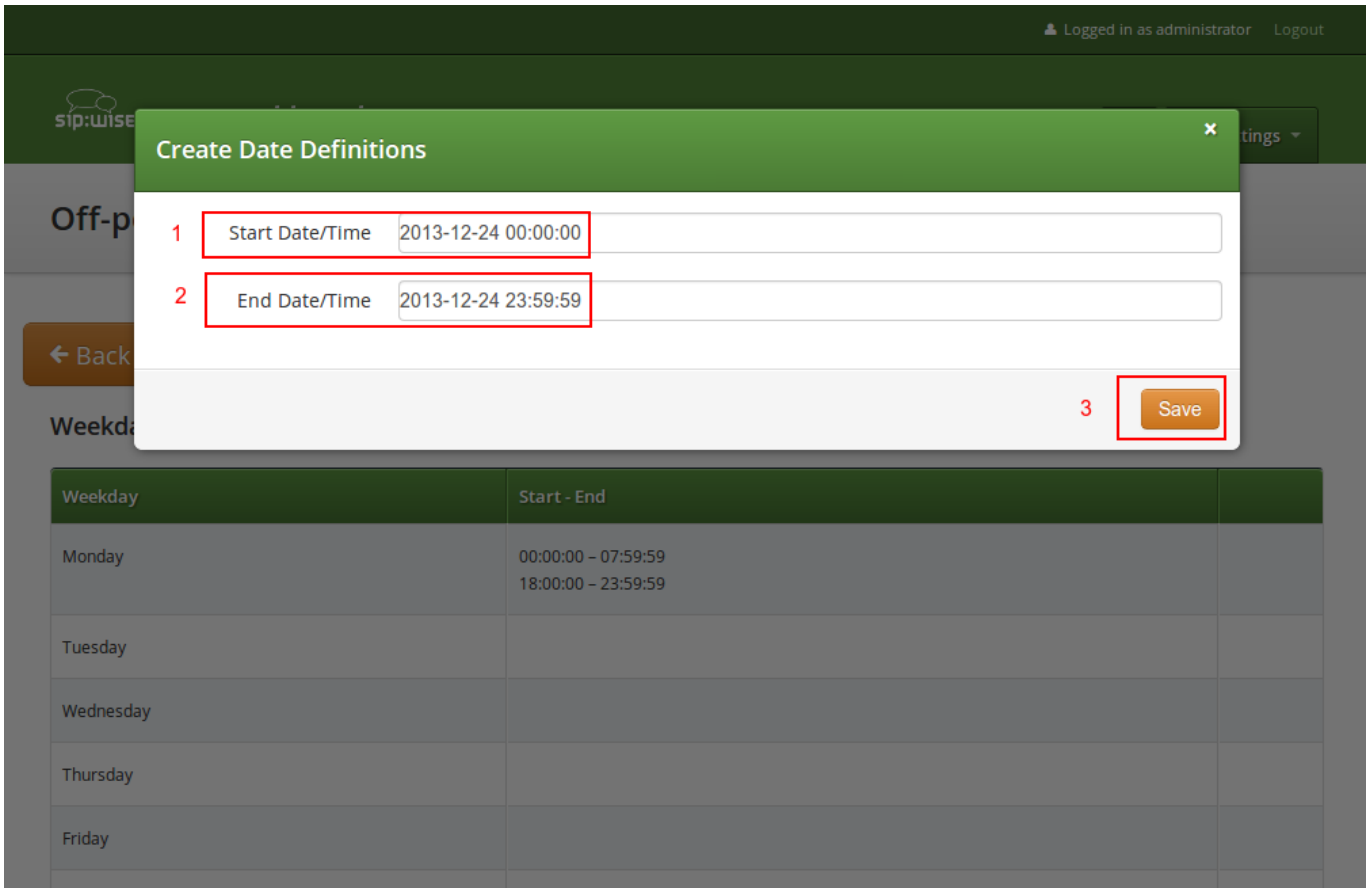
To be able to differentiate between on-peak and off-peak calls, the platform stores off-peak times for every billing profile based on weekdays and/or calendar days. To edit the settings for a billing profile, go to *Settings→Billing* and press the *Peaktimes* button on the billing profile you want to configure.

To set off-peak times for a weekday, click on *Edit* next to the according weekday. You will be presented with two input fields which both receive a timestamp in the form of *hh:mm:ss* specifying a time of day for the start and end of the off-peak period. If any of the fields is left empty, the system will automatically insert `00:00:00` (*start* field) or *23:59:59* (*end* field). Click on *Add* to store the setting in the database. You may create more than one off-peak period per weekday. To delete a range, just click *Delete* next to the entry. Click the *close* icon when done.



To specify off-peak ranges based on calendar dates, click on *Create Special Off-Peak Date*. Enter a date in the form of *YYYY-*

MM-DD hh:mm:ss into the *Start Date/Time* input field and *End Date/Time* input field to define a range for the off-peak period.



6.1.4 Fraud Detection and Locking

The NGCP supports a fraud detection feature, which is designed to detect accounts causing unusually high customer costs, and then to perform one of several actions upon those accounts. This feature can be enabled and configured through two sets of billing profile options described in Section 6.1.1, namely the monthly (*fraud monthly limit*, *fraud monthly lock* and *fraud monthly notify*) and daily limits (*fraud daily limit*, *fraud daily lock* and *fraud daily notify*). Either monthly/daily limits or both of them can be active at the same time.

Once a day, shortly after midnight local time, a background script automatically checks all accounts which are linked to a billing profile enabled for fraud detection, and selects those which have caused a higher cost than the *fraud monthly limit* configured in the billing profile, within the currently active billing interval (e.g. in the current month), or a higher cost than the *fraud daily limit* configured in the billing profile, within the calendar day. It then proceeds to perform at least one of the following actions on those accounts:

- If **fraud lock** is set to anything other than *none*, it will lock the account accordingly (e.g. if **fraud lock** is set to *outgoing*, the account will be locked for all outgoing calls).
- If anything is listed in **fraud notify**, an email will be sent to the email addresses configured. The email will contain information about which account is affected, which subscribers within that account are affected, the current account balance and the configured fraud limit, and also whether or not the account was locked in accordance with the **fraud lock** setting. It should be noted that this email is meant for the administrators or accountants etc., and not for the customer.

Important

You can override these settings on a per-account basis via SOAP or the Admin interface.

Caution

Accounts that were automatically locked by the fraud detection feature will **not** be automatically unlocked when the next billing interval starts. This has to be done manually through the administration panel or through the provisioning interface.

Important

If fraud detection is configured to only send an email and not lock the affected accounts, it will continue to do so for over-limit accounts every day. The accounts must either be locked in order to stop the emails (only currently active accounts are considered when the script looks for over-limit accounts) or some other action to resolve the conflict must be taken, such as disabling fraud detection for those accounts.

6.2 Billing Data Export

Regular billing data export is done using CSV (*comma separated values*) files which may be downloaded from the platform using the *cdrexport* user which has been created during the installation.

There are two types of exports. One is *CDR* (Call Detail Records) used to charge for calls made by subscribers, and the other is *EDR* (Event Detail Records) used to charge for provisioning events like enabling certain features.

6.2.1 File Name Format

In order to be able to easily identify billing files, the file names are constructed by the following fixed-length fields:

```
<prefix><separator><version><separator><timestamp><separator><sequence number>< ↩
    suffix>
```

The definition of the specific fields is as follows:

Table 3: CDR/EDR export file name format

File name element	Length	Description
`<prefix>`	7	A fixed string. Always `sipwise`.
`<separator>`	1	A fixed character. Always _.
`<version>`	3	The format version, a three digit number. Currently `007`.
`<timestamp>`	14	The file creation timestamp in the format `YYYYMMDDhhmmss`.
`<sequence number>`	10	A unique 10-digit zero-padded sequence number for quick identification.
`<suffix>`	4	A fixed string. Always `.cdr` or `.edr`.

A valid example filename for a CDR billing file created at 2012-03-10 14:30:00 and being the 42nd file exported by the system, is:

`sipwise_007_20130310143000_0000000042.cdr`

6.2.2 File Format

Each billing file consists of three parts: one header line, zero to 5000 body lines and one trailer line.

File Header Format

The billing file header is one single line, which is constructed by the following fields:

`<version>,<number of records>`

The definition of the specific fields is as follows:

Table 4: CDR/EDR export file header line format

Body Element	Length	Type	Description
`<version>`	3	zero-padded uint	The format version. Currently `007`.
`<number of records>`	4	zero-padded uint	The number of body lines contained in the file.

A valid example for a Header is:

`007,0738`

File Body Format for Call Detail Records (CDR)

The body of a CDR consists of a minimum of zero and a maximum of 5000 lines. Each line holds one call detail record in CSV format and is constructed by the following fields, all of them enclosed in single quotes:

Table 5: CDR export file body line format

Body Element	Length	Type	Description
`<id>`	1-10	uint	Internal CDR id.

Table 5: (continued)

Body Element	Length	Type	Description
<update_time>	19	timestamp	Timestamp of last modification.
<source_user_id>	36	string	Internal UUID of calling party subscriber.
<source_provider_id>	1-255	string	Internal ID of calling party provider.
<source_ext_subscriber_id>	0-255	string	External ID of calling party subscriber.
<source_subscriber_id>	1-10	uint	Internal ID of calling party subscriber.
<source_ext_account_id>	0-255	string	External ID of calling party VoIP account.
<source_account_id>	1-10	uint	Internal ID of calling party VoIP account.
<source_user>	1-255	string	SIP username of calling party.
<source_domain>	1-255	string	SIP domain of calling party.
<source_cli>	1-64	string	CLI of calling party in E.164 format.
<source_clir>	1	uint	1 for calls with CLIR, 0 otherwise.
<source_ip>	0-64	string	IP Address of the calling party.
<destination_user_id>	1 / 36	string	Internal UUID of called party subscriber or 0 if callee is not local.
<destination_provider_id>	1-255	string	Internal ID of called party provider.
<dest_ext_subscriber_id>	0-255	string	External ID of called party subscriber.
<dest_subscriber_id>	1-10	uint	Internal ID of called party subscriber.
<dest_ext_account_id>	0-255	string	External ID of called party VoIP account.
<destination_account_id>	1-10	uint	Internal ID of called party VoIP account.
<destination_user>	1-255	string	Final SIP username of called party.
<destination_domain>	1-255	string	Final SIP domain of called party.
<destination_user_in>	1-255	string	Incoming SIP username of called party.
<destination_domain_in>	1-255	string	Incoming SIP domain of called party.
<dialed_digits>	1-255	string	The user-part of the SIP Request URI as received by the soft-switch.
<peer_auth_user>	0-255	string	User to authenticate towards peer.
<peer_auth_realm>	0-255	string	Realm to authenticate towards peer.
<call_type>	3-4	string	The type of the call - one of: call: normal call cfu: call forward unconditional cft: call forward timeout cfb: call forward busy cfna: call forward no answer

Table 5: (continued)

Body Element	Length	Type	Description
<call_status>	2-7	string	The final call status - one of: ok: successful call busy: callee busy noanswer: no answer from callee cancel: cancel from caller offline callee offline timeout: no reply from callee other: unspecified, see <call_code> for details
<call_code>	3	uint	The final SIP status code.
<init_time>	23	timestamp	Timestamp of call initiation (invite received from caller). Seconds include fractional part (3 decimals).
<start_time>	23	timestamp	Timestamp of call establishment (final response received from callee). Seconds include fractional part (3 decimals).
<duration>	4-11	fixed precision	Length of call (beginning at start_time) in seconds with 3 decimals.
<call_id>	1-255	string	The SIP call-id.
<rating_status>	2-7	string	The internal rating status - one of: unrated: not rated ok: successfully rated failed: error while rating Currently always ok or unrated, depending on whether rating is enabled or not.
<rated_at>	0 / 19	timestamp	Timestamp of rating or empty if not rated.
<source_carrier_cost>	4-11	fixed precision	The originating carrier cost or empty if not rated. In cent with two decimals. Only available in system exports, not for resellers.
<source_customer_cost>	4-11	fixed precision	The originating customer cost or empty if not rated. In cent with two decimals.
<source_carrier_zone>	0-127	string	The originating carrier billing zone or empty if not rated. Only available in system exports, not for resellers.
<source_customer_zone>	0-127	string	The originating customer billing zone or empty if not rated.
<source_carrier_destination>	0-127	string	The originating carrier billing destination or empty if not rated. Only available in system exports, not for resellers.
<source_customer_destination>	0-127	string	The originating customer billing destination or empty if not rated.

Table 5: (continued)

Body Element	Length	Type	Description
<source_carrier_free_time>	1-10	uint	The number of originating free time seconds used on carrier side or empty if not rated. Only available in system exports, not for resellers.
<source_customer_free_time>	1-10	uint	The number of originating free time seconds used from the customer's account balance or empty if not rated.
<destination_carrier_cost>	4-11	fixed precision	The termination carrier cost or empty if not rated. In cent with two decimals. Only available in system exports, not for resellers.
<destination_customer_cost>	4-11	fixed precision	The termination customer cost or empty if not rated. In cent with two decimals.
<destination_carrier_zone>	0-127	string	The termination carrier billing zone or empty if not rated. Only available in system exports, not for resellers.
<destination_customer_zone>	0-127	string	The termination customer billing zone or empty if not rated.
<destination_carrier_destination>	0-127	string	The termination carrier billing destination or empty if not rated. Only available in system exports, not for resellers.
<destination_customer_destination>	0-127	string	The termination customer billing destination or empty if not rated.
<destination_carrier_free_time>	1-10	uint	The number of termination free time seconds used on carrier side or empty if not rated. Only available in system exports, not for resellers.
<destination_customer_free_time>	1-10	uint	The number of termination free time seconds used from the customer's account balance or empty if not rated.
<source_reseller_cost>	4-11	fixed precision	The originating reseller cost or empty if not rated. In cent with two decimals. Only available in system exports, not for resellers.
<source_reseller_zone>	0-127	string	The originating reseller billing zone or empty if not rated. Only available in system exports, not for resellers.
<source_reseller_destination>	0-127	string	The originating reseller billing destination or empty if not rated. Only available in system exports, not for resellers.
<source_reseller_free_time>	1-10	uint	The number of originating free time seconds used from the reseller's account balance or empty if not rated. Only available in system exports, not for resellers.
<destination_reseller_cost>	4-11	fixed precision	The termination reseller cost or empty if not rated. In cent with two decimals. Only available in system exports, not for resellers.

Table 5: (continued)

Body Element	Length	Type	Description
<destination_reseller_zone>	0-127	string	The termination reseller billing zone or empty if not rated. Only available in system exports, not for resellers.
<destination_reseller_destination>	0-127	string	The termination reseller billing destination or empty if not rated. Only available in system exports, not for resellers.
<destination_reseller_free_time>	1-10	uint	The number of termination free time seconds used from the reseller's account balance or empty if not rated. Only available in system exports, not for resellers.
<line_terminator>	1	string	A fixed character. Always \n (special char LF - ASCII 0x0A).

A valid example of one body line of a rated CDR is (line breaks added for clarity):

```
'15','2013-03-26 22:09:11','a84508a8-d256-4c80-a84e-820099a827b0','1','','1','',
'2','testuser1','192.168.51.133','4311001','0','192.168.51.1',
'94d85b63-8f4b-43f0-b3b0-221c9e3373f2','1','','3','','4','testuser3',
'192.168.51.133','testuser3','192.168.51.133','testuser3','','','call','ok','200',
'2013-03-25 20:24:50.890','2013-03-25 20:24:51.460','10.880','44449842',
'ok','2013-03-25 20:25:27','0.00','24.00','onnet','testzone','platform internal',
'testzone','0','0','0.00','200.00','','foo','','foo','0','0',
'0.00','','','0','0.00','','','0'
```

The format of the CDR export files generated for resellers (as opposed to the complete system-wide export) is identical except for a few missing fields. Reseller CDR CSV files don't contain the fields for *carrier* or *reseller* ratings, neither in *source* nor *destination* direction. Thus, the reseller CSV files have 16 fewer fields.

File Body Format for Event Detail Records (EDR)

The body of a EDR consists of a minimum of zero and a maximum of 5000 lines. Each line holds one call detail record in CSV format and is constructed by the following fields, all of them enclosed in single quotes:

Table 6: EDR export file body line format

Body Element	Length	Type	Description
<event_id>	1-10	uint	Internal EDR id.

Table 6: (continued)

Body Element	Length	Type	Description
<event_type>	1-255	string	The type of the event - one of: start_profile: A subscriber profile has been newly assigned to a subscriber. end_profile: A subscriber profile has been removed from a subscriber. update_profile: A subscriber profile has been changed for a subscriber. start_huntgroup: A subscriber has been provisioned as group. end_huntgroup: A subscriber has been deprovisioned as group. start_ivr: A subscriber has a new call-forward to auto-attendant set. end_ivr: A subscriber has removed a call-forward to auto-attendant.
<customer_external_id>	0-255	string	The external customer ID as provisioned for the subscriber.
<contact_company>	0-255	string	The company name of the customer's contact.
<subscriber_external_id>	0-255	string	The external subscriber ID as provisioned for the subscriber.
<subscriber_number>	0-255	string	The voip number of the subscriber with the highest ID (DID or primary number).
<old_status>	0-255	string	The old status of the event. Depending on the event_type: start_profile: Empty. end_profile: The profile id of the profile which got removed from the subscriber. update_profile: The old profile id which got updated. start_huntgroup: Empty. end_huntgroup: The profile id of the group which got deprovisioned. start_ivr: Empty. end_ivr: Empty.

Table 6: (continued)

Body Element	Length	Type	Description
<new_status>	0-255	string	The new status of the event. Depending on the event_type: start_profile: The profile id which got assigned to the subscriber. end_profile: Empty. update_profile: The new profile id which got updated. start_huntgroup: The current profile id assigned to the group subscriber. end_huntgroup: The current profile id assigned to the group subscriber. start_ivr: Empty. end_ivr: Empty.
<timestamp>	0-255	string	The time when the event occured.
<line_terminator>	1	string	A fixed character. Always \n (special char LF - ASCII 0x0A).

A valid example of one body line of an EDR is (line breaks added for clarity):

```
"1","start_profile","sipwise_ext_customer_id_4","Sipwise GmbH",
"sipwise_ext_subscriber_id_44","436667778","","1","2014-06-19 11:34:31"
```

File Trailer Format

The billing file trailer is one single line, which is constructed by the following fields:

```
<md5 sum>
```

The <md5 sum> is a 32 character hexadecimal MD5 hash of the *Header* and *Body*.

To validate the billing file, one must remove the Trailer before computing the MD5 sum of the file. An example bash script to validate the integrity of the file is given below:

```
#!/bin/sh

error() { echo $@; exit 1; }
test -n "$1" || error "Usage: $0 <cdr-file>"
test -f "$1" || error "File '$1' not found"

TMPFILE="/tmp/$(basename "$1").$$"
```

```
MD5="$(sed -rn '$ s/^([a-z0-9]{32}).*$/\1/i p' "$1")  $TMPFILE"
sed '$d' "$1" > "$TMPFILE"
echo "$MD5" | md5sum -c -
rm -f "$TMPFILE"
```

Given the script is located in `cdr-md5.sh` and the CDR-file is `sipwise_001_20071110123000_0000000004.cdr`, the output of the integrity check for an intact CDR file would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: OK
```

If the file has been altered during transmission, the output of the integrity check would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: FAILED
md5sum: WARNING: 1 of 1 computed checksum did NOT match
```

6.2.3 File Transfer

Billing files are created twice per hour at minutes `25` and `55` and are stored in the home directory of the `cdrexport` user. If the amount of records within the transmission interval exceeds the threshold of 5000 records per file, multiple billing files are created. If no billing records are found for an interval, a billing file without body data is constructed for easy detection of lost billing files on the 3rd party side.

CDR and EDR files are fetched by a 3rd party billing system using SFTP or SCP with either public key or password authentication using the username cdrexport.
If public key authentication is chosen, the public key file has to be stored in the file `~/.ssh/authorized_keys2` below the home directory of the `cdrexport` user. Otherwise, a password has to be set for the user.

The 3rd party billing system is responsible for deleting CDR files after fetching them.

Note

The `cdrexport` user is kept in a jailed environment on the system, so it has only access to a very limited set of commandline utilities.

---

7 Invoices and invoice templates

The sip:carrier allows to generate and send customer invoices for each billing period based on Calls Detailed Records (CDR). Generated invoices can be sent to customers emails using invoice generation script Section 7.3.

Invoices present billing information from the reseller point of view. Recipients of the invoices are customers. Invoices include information related to the calls made by subscribers associated with the customer.

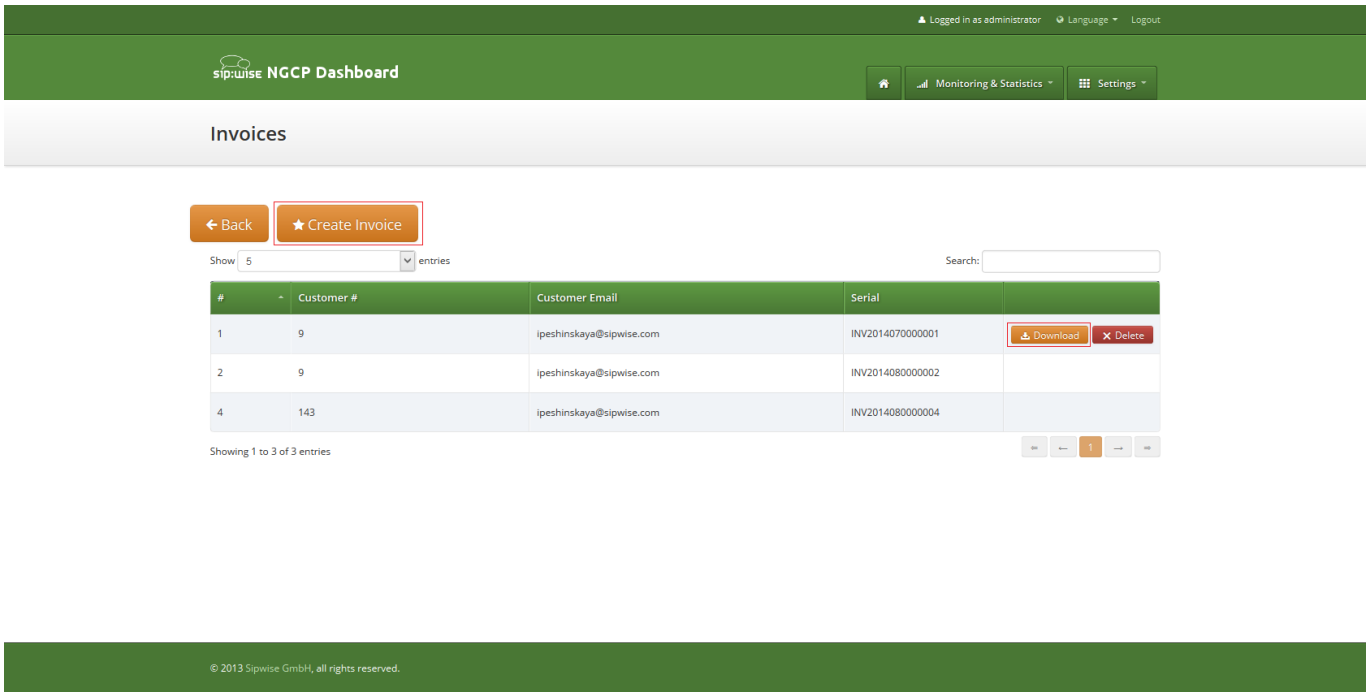
By default invoice contains information about billing plan fixed fee, calls zones fees and calls detailed information.

Content and vision of the invoices are customizable by invoice templates Section 7.2.

Note
The sip:carrier generates invoices in pdf format.

7.1 Invoices management

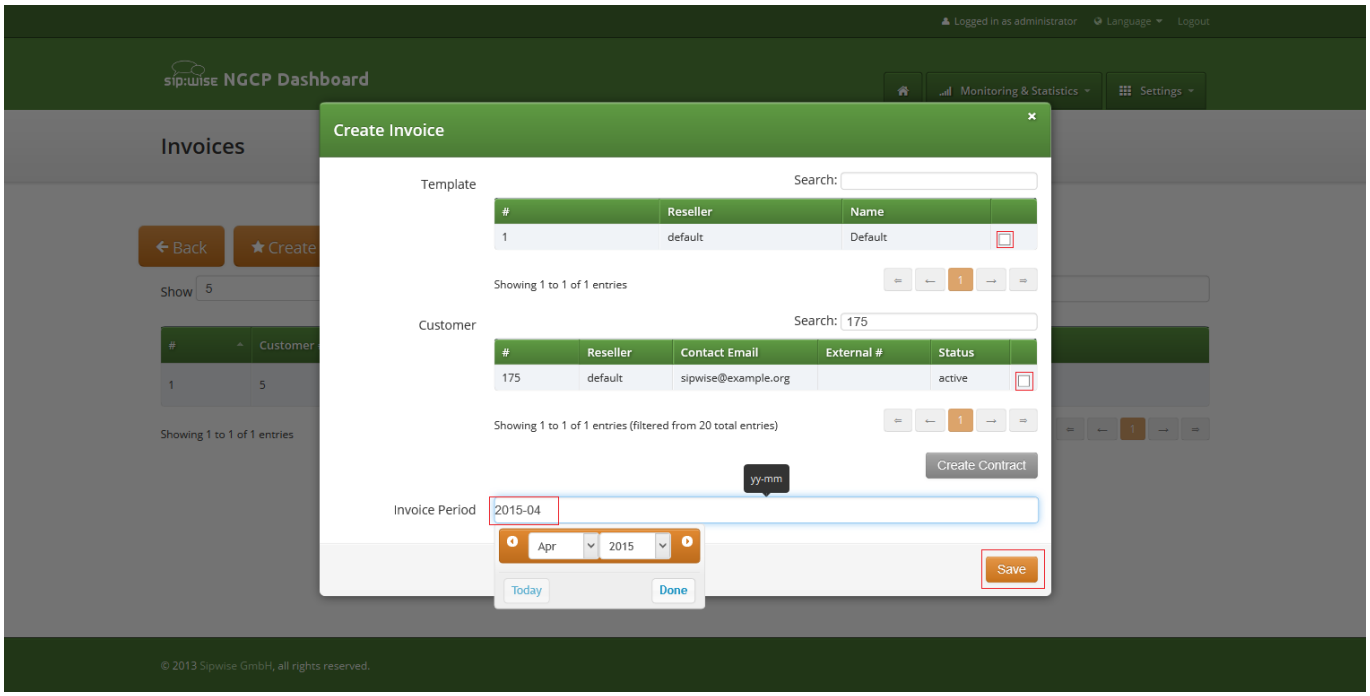
Invoices can be requested for generation, searched, downloaded and deleted in the invoices interface.



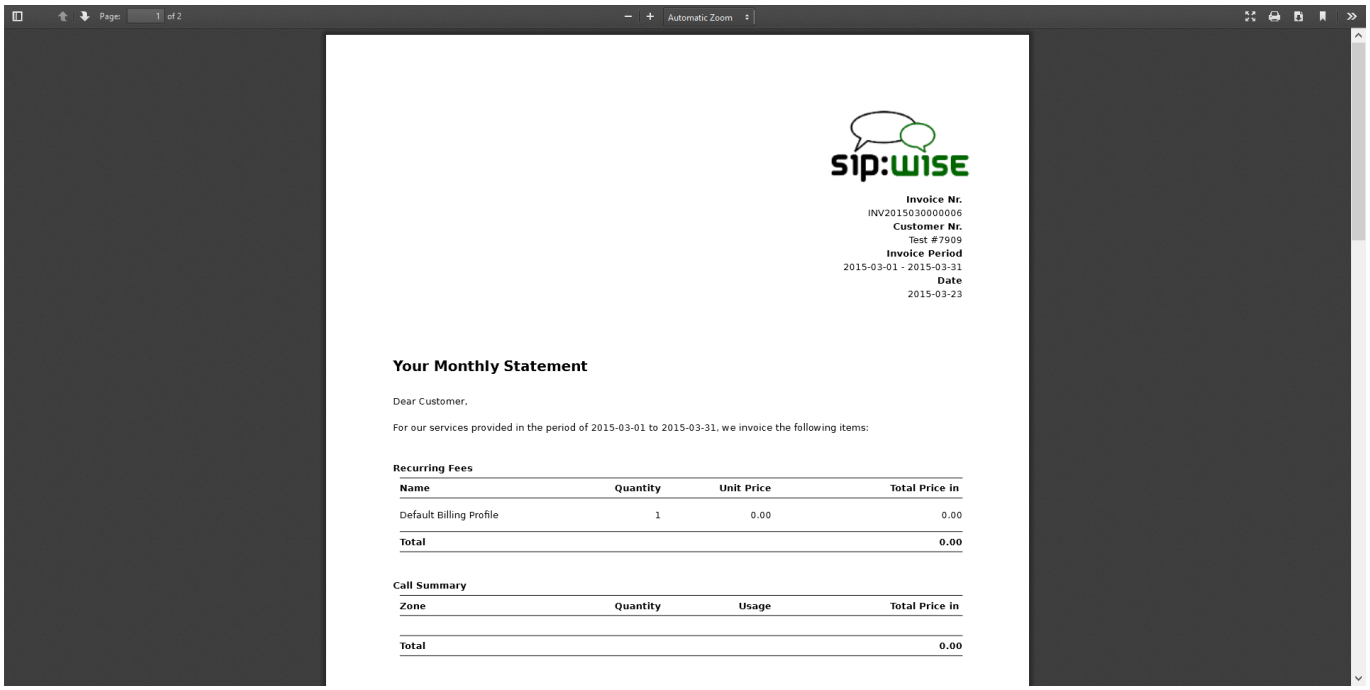
To request invoice generation for the particular customer and period press "Create invoice" button. On the invoice creation form following parameters are available for selection:

- **Template**: any of existent invoice template can be selected for the invoice generation.
- **Customer**: owner of the billing account, recipient of the invoice.
- **Invoice period**: billing period. Can be specified only as one calendar month. Calls with start time between first and last second of the period will be considered for the invoice

All form fields are mandatory.



Generated invoice can be downloaded as pdf file.



To do it press button "Download" against invoice in the invoice management interface.

Respectively press on the button "Delete" to delete invoice.

7.2 Invoice templates

Invoice template defines structure and look of the generated invoices. The sip:carrier allows to create some invoice templates. Multiple invoice templates can be used to send invoices to the different customers using different languages.

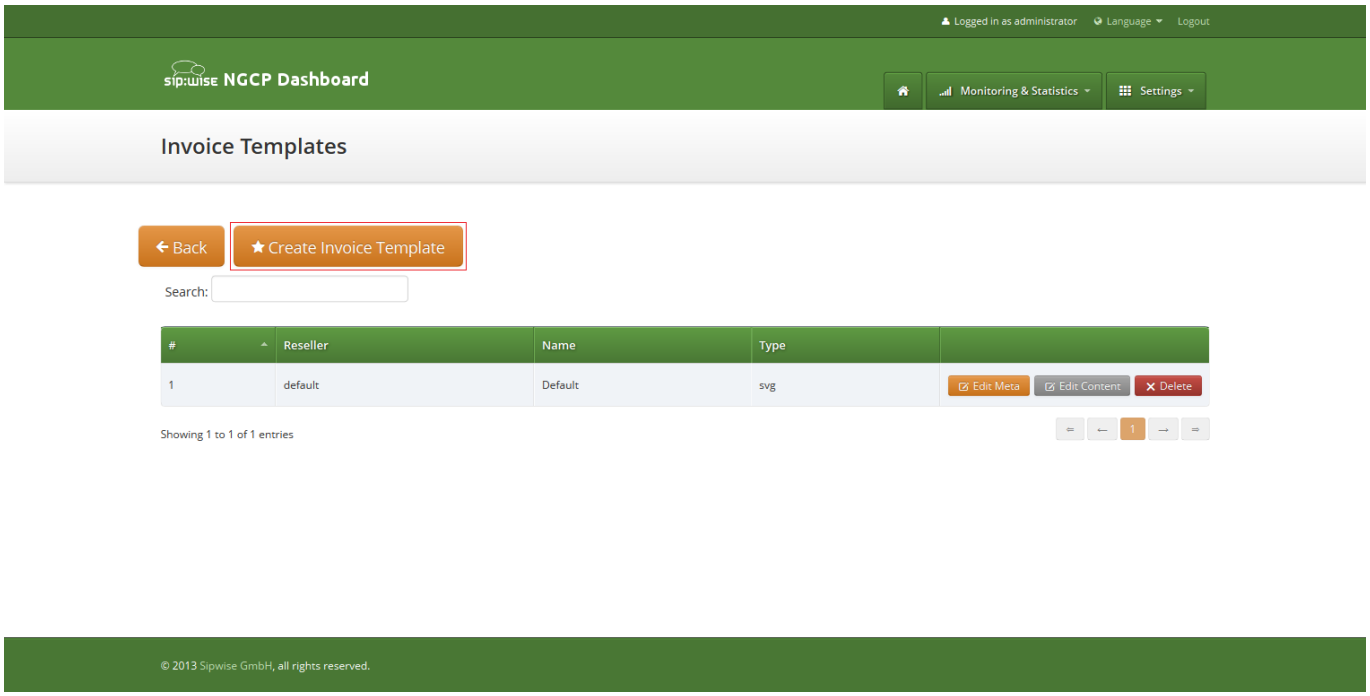
Important

At least one invoice template should be created to enable invoice generation. Each customer has to be associated to one of the existent invoice template, otherwise invoices will be not generated for this customer.

Customer can be linked to the invoice template in the customer interface.

7.2.1 Invoice Templates management

Invoice templates can be searched, created, edited and deleted in the invoice templates management interface.



Invoice template creation is separated on two steps:

- Register new invoice template meta information.
- Edit content (template itself) of the invoice template.

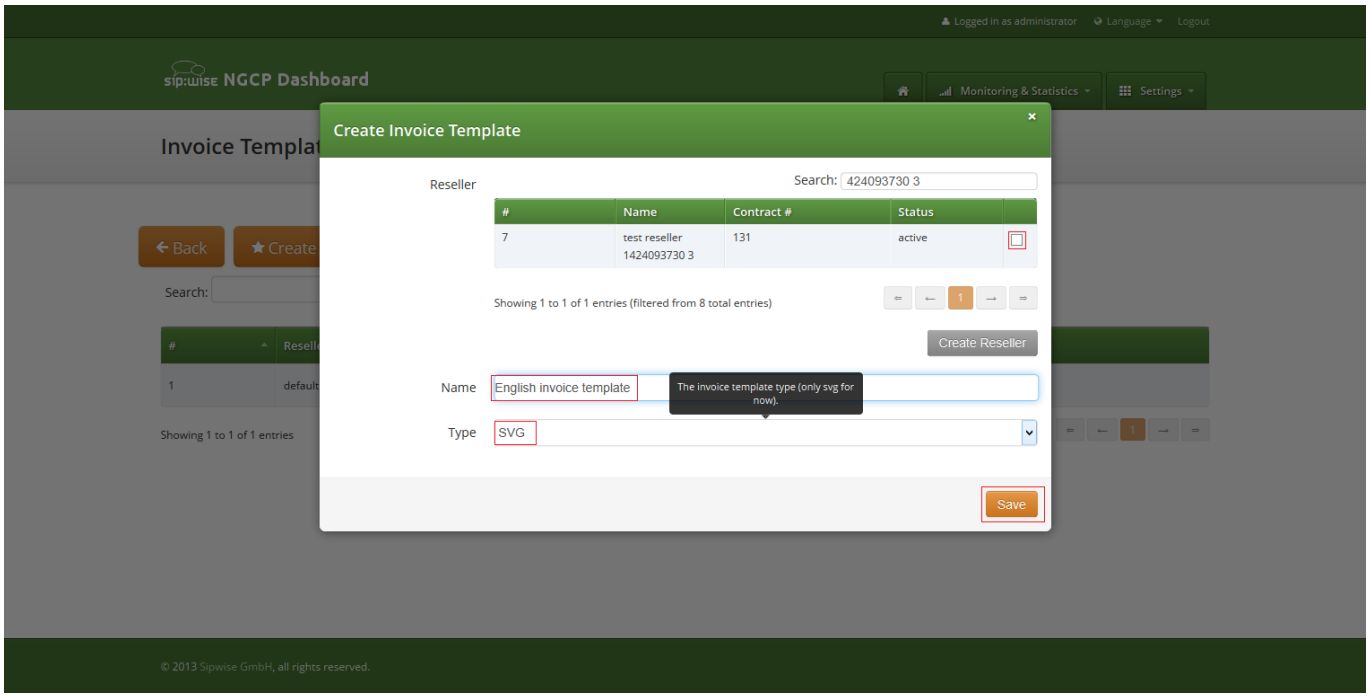
To register new invoice template press "Create Invoice Template" button.

On the invoice template meta information form following parameters can be specified:

- **Reseller**: reseller who owns this invoice template. Please note, that it doesn't mean that the template will be used for the reseller customers by default. After creation, invoice template still need to be linked to the reseller customers.

- **Name**: unique invoice template name to differentiate invoice templates if there are some.
- **Type**: currently sip:carrier supports only svg format of the invoice templates.

All form fields are mandatory.



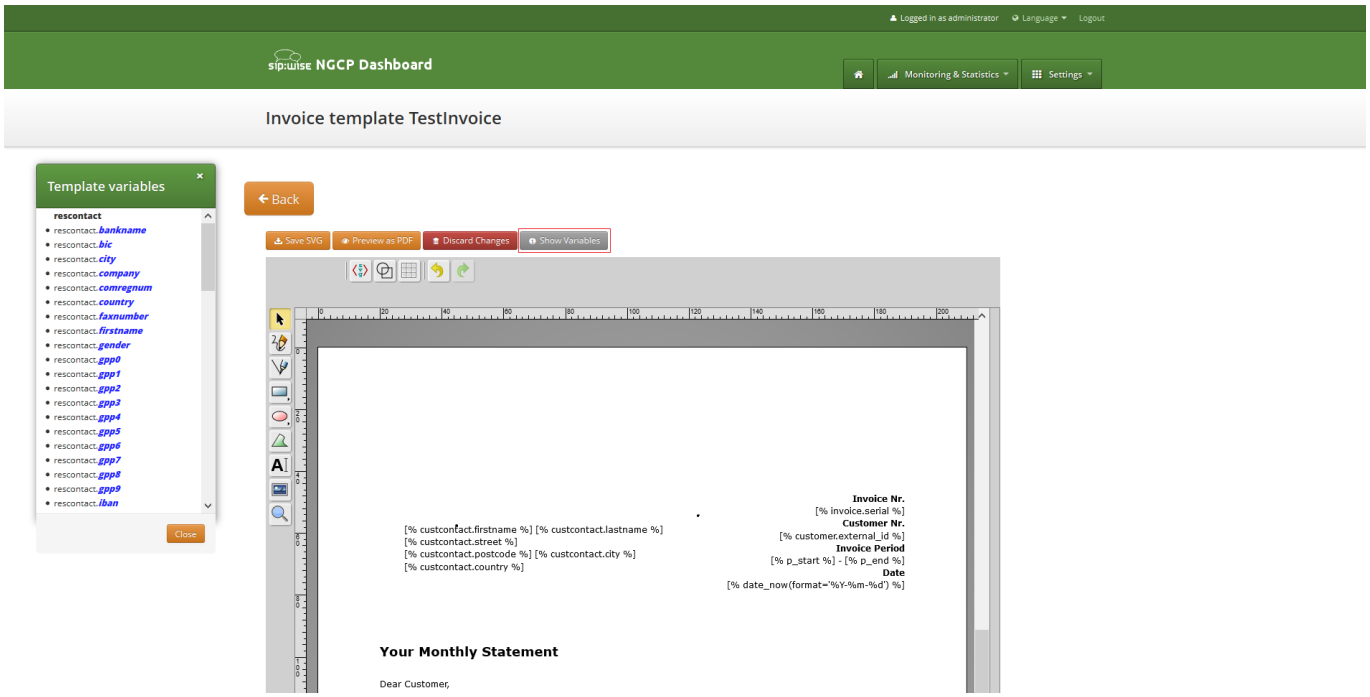
After registering new invoice template you can change invoice template structure in WYSIWYG SVG editor and preview result of the invoice generation based on the template.

7.2.2 Invoice Template content

Invoice template is a XML SVG source, which describes content, look and position of the text lines, images or other invoice template elements. The sip:carrier provides embedded WYSIWYG SVG editor svg-edit 2.6 to customize default template. The sip:carrier svg-edit has some changes in layers management, image edit, user interface, but this basic introduction still may be useful.

Template refers to the owner reseller contact ("rescontact"), customer contract ("customer"), customer contact ("custcontact"), billing profile ("billprof"), invoice ("invoice") data as variables in the "[%%]" mark-up with detailed information accessed as field name after point e.g. [%invoice.serial%]. During invoice generation all variables or other special tokens in the "[% %]" mark-ups will be replaced by their database values.

Press on "Show variables" button on invoice template content page to see full list of variables with the fields:



You can add/change/remove embedded variables references directly in main svg-edit window. To edit text line in svg-edit main window double click on the text and place cursor on desired position in the text.

After implementation of the desired template changes, invoice template should be saved Section 7.2.3.

To return to the sip:carrier invoice template **default** content you can press on the "Discard changes" button.

Important

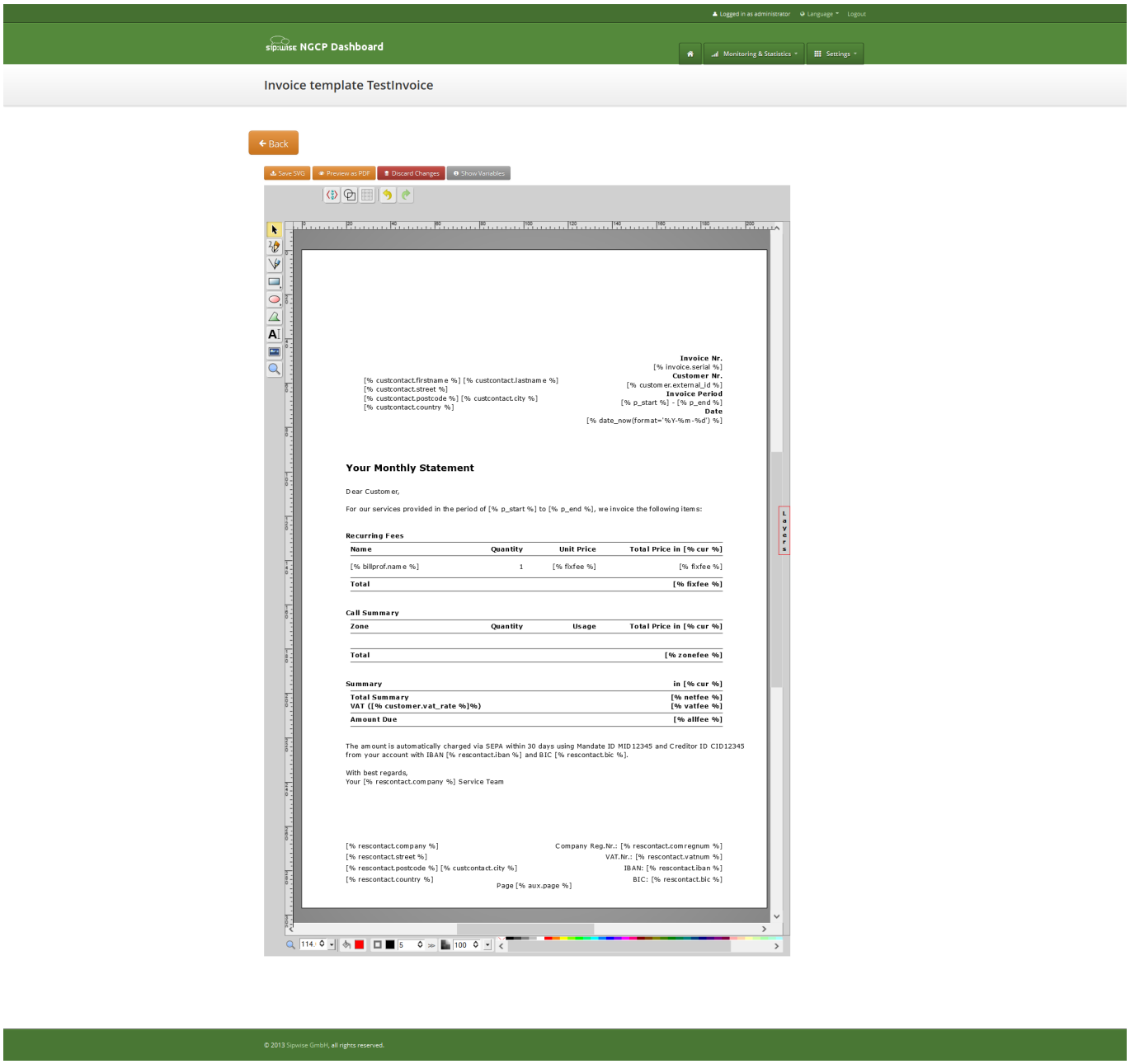
"Discard changes" operation can't be undone.

Layers

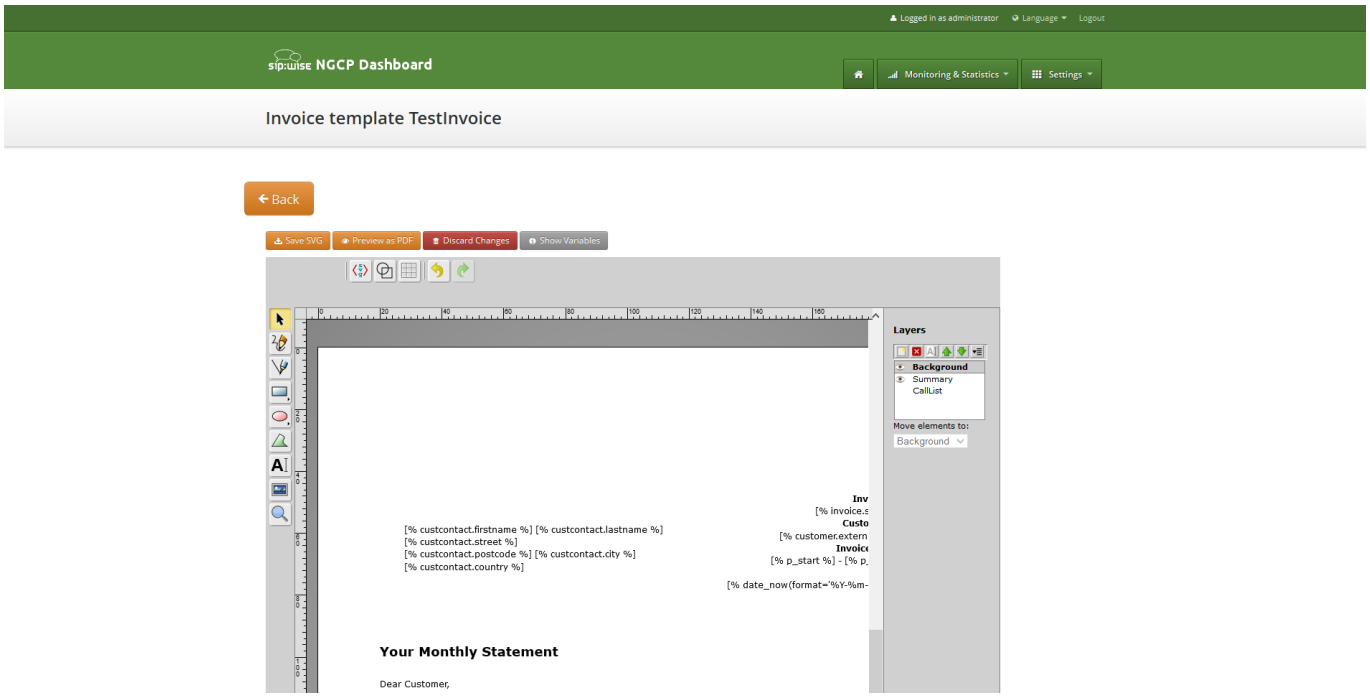
Default template contains three groups elements (<g/>), which can be thinked of as pages, or in terms of svg-edit - layers. Layers are:

- **Background**: special layer, which will be repeated as background for every other page of the invoice.
- **Summary**: page with a invoice summary.
- **CallList**: page with calls made in a invoice period. Is invisible by default.

To see all invoice template layers, press on "Layers" vertical sign on right side of the svg-edit interface:



Side panel with layers list will be shown.

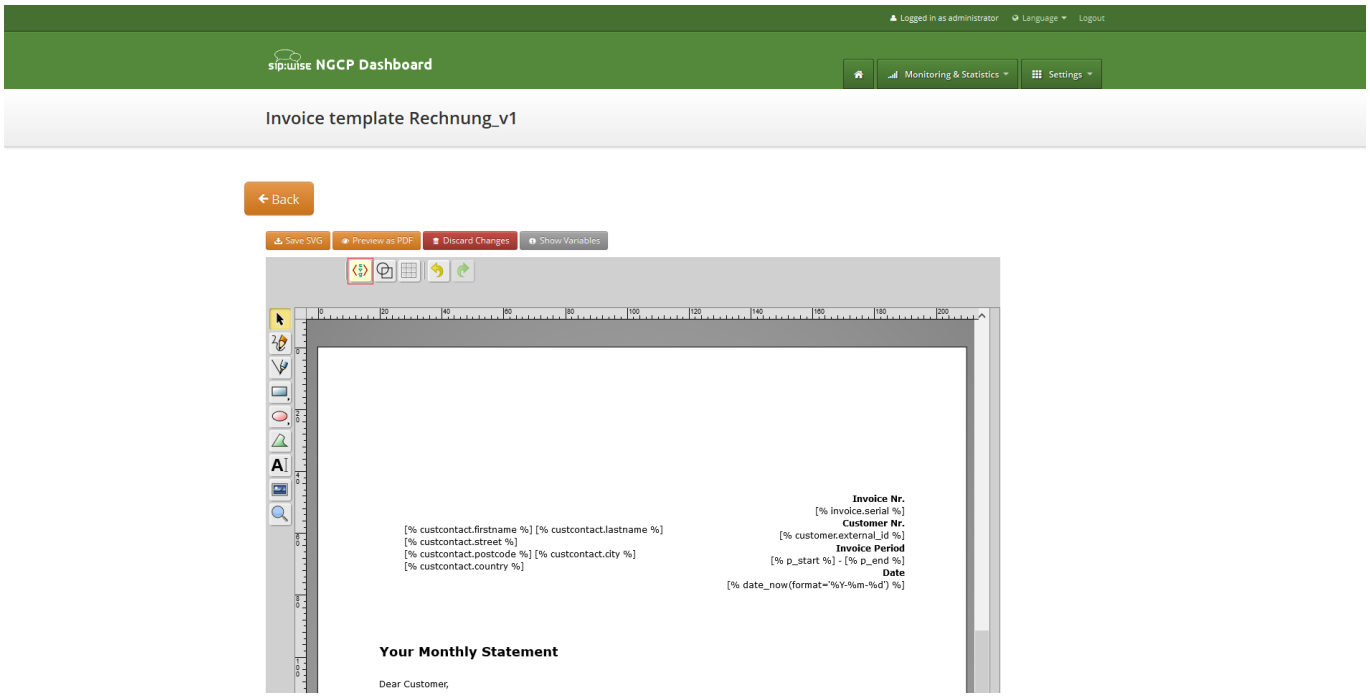


One of the layers is active, and its element can be edited in the main svg-edit window. Currently active layer's name is **bold** in the layers list. The layers may be visible or invisible. Visible layers have "eye" icon left of their names in the layers list.

To make a layer active, click on its name in the layers list. If the layer was invisible, its elements became visible on activation. Thus you can see mixed elements of some layers, then you can switch off visibility of other layers by click on their "eye" icons. It is good idea to keep visibility of the "Background" layer on, so look of the generated page will be seen.

Edit SVG XML source

Sometimes it may be convenient to edit svg source directly and svg-edit allows to do it. After press on the <svg> icon in the top left corner of the svg-edit interface:



SVG XML source of the invoice template will be shown.

SVG source can be edited in place or just copy-pasted as usual text.

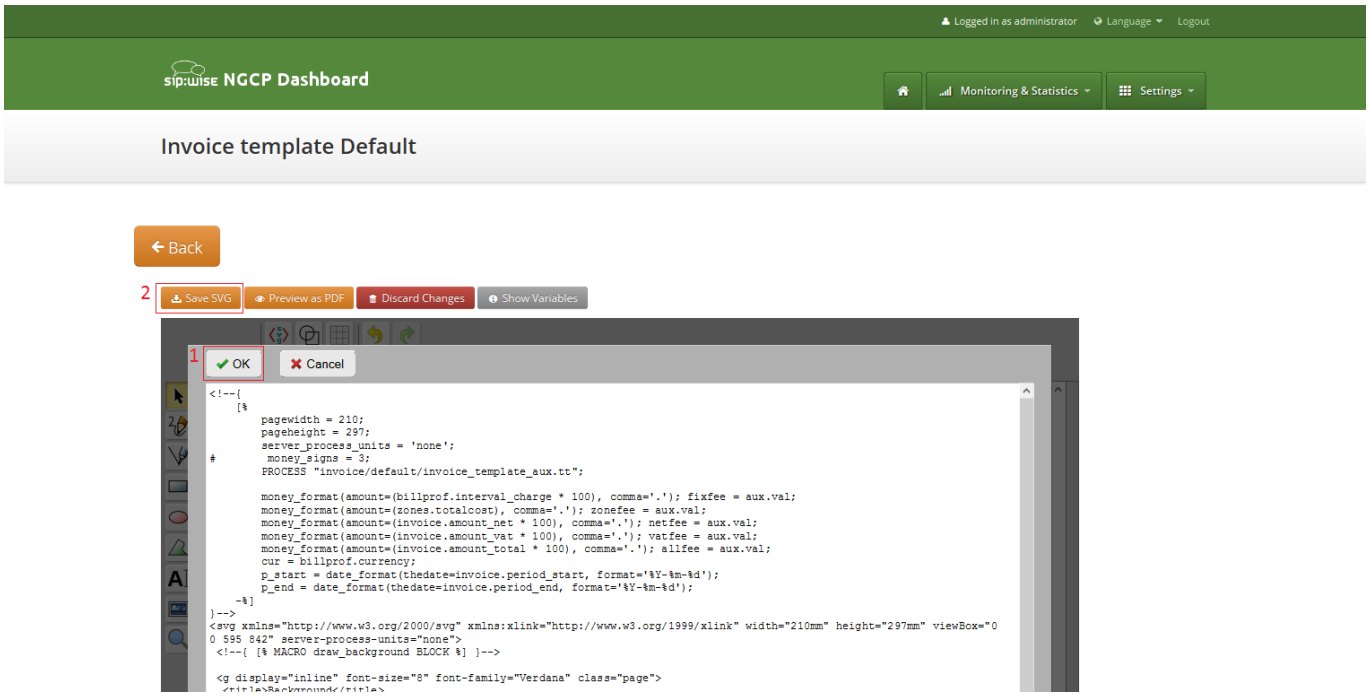
Note

Template keeps sizes and distances in pixels.

Important

When edit svg xml source, please change very carefully and thinkfully things inside special comment mark-up "<!--{ }-→". Otherwise invoice generation may be broken. Please be sure that document structure repeats default invoice template: has the same groups (<g/>) elements on the top level, text inside special comments mark-up "<!--{ }-→" preserved or changed appropriately, svg xml structure is correct.

To save your changes in the svg xml source, first press "OK" button on the top left corner of the source page:



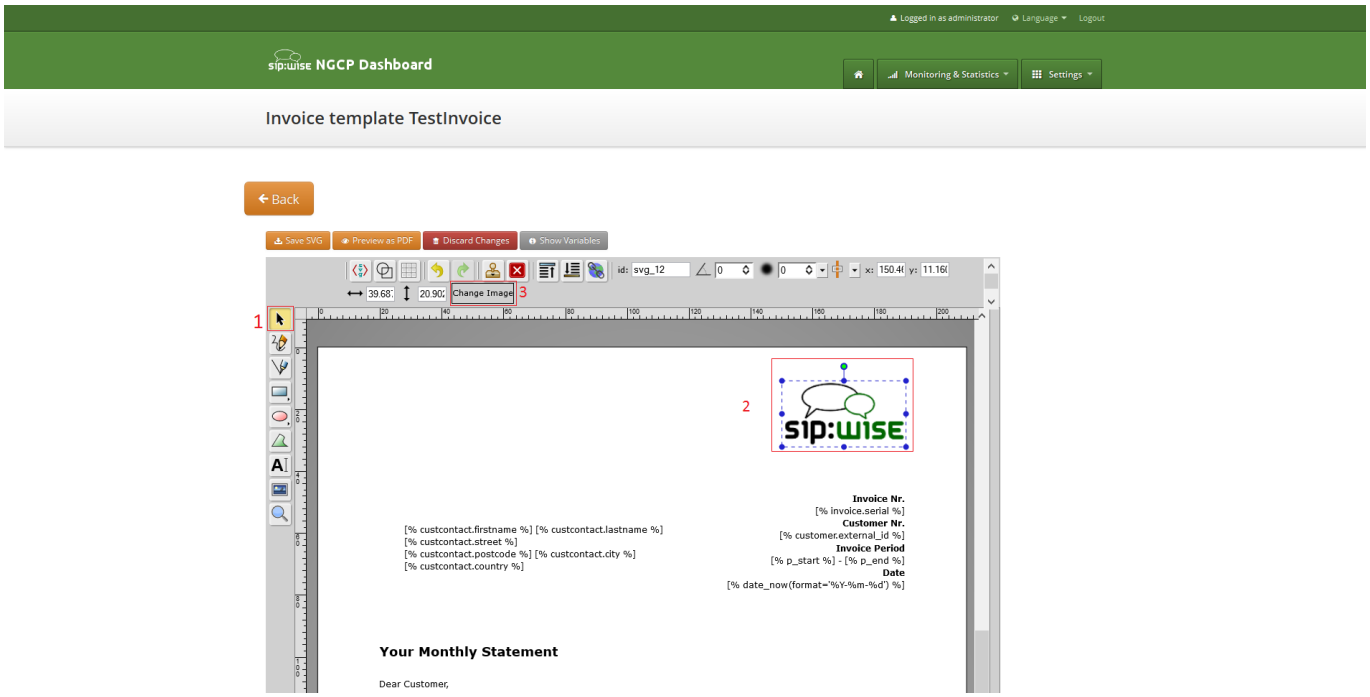
And then save invoice template changes Section 7.2.3.

Note

You can copy and keep the svg source of your template as a file on the disk before start experimenting with the template. Later you will be able to return to this version replacing svg source.

Change logo image

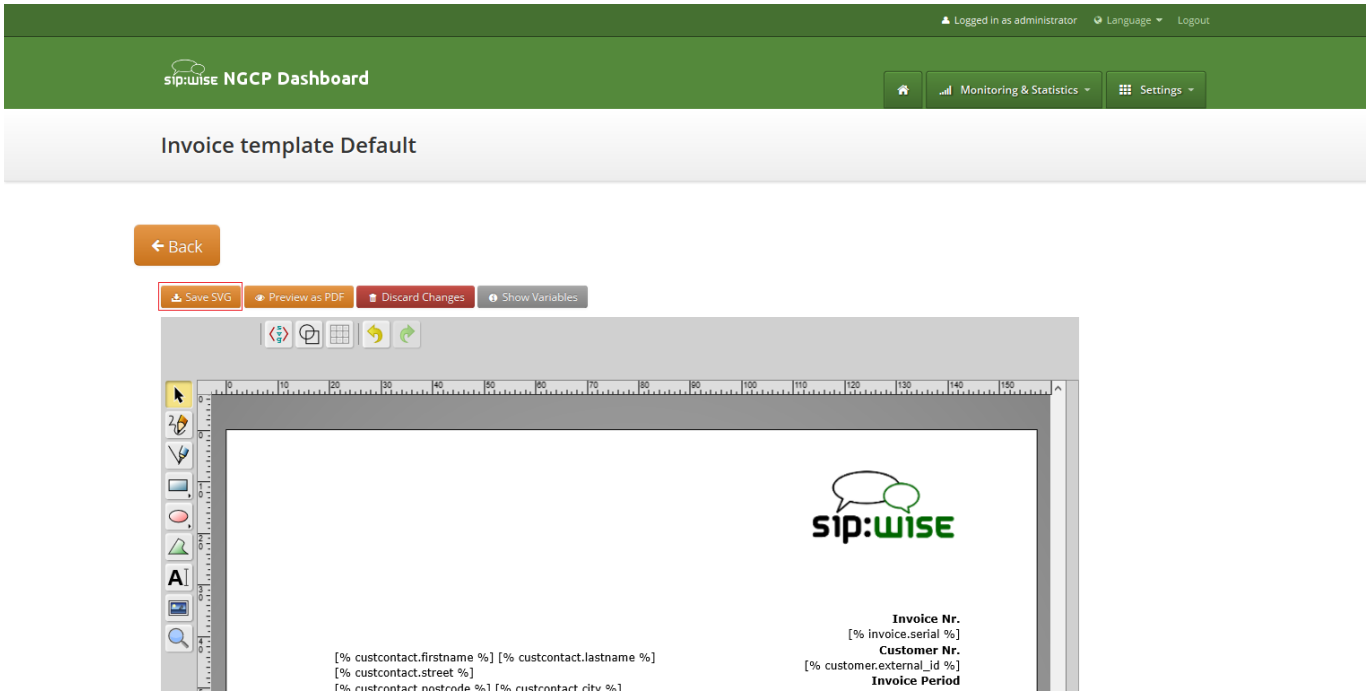
- Make sure that "Select tool" is active.
- Select default logo, clicking on the logo image.
- Press "Change image" button, which should appear on the top toolbar.



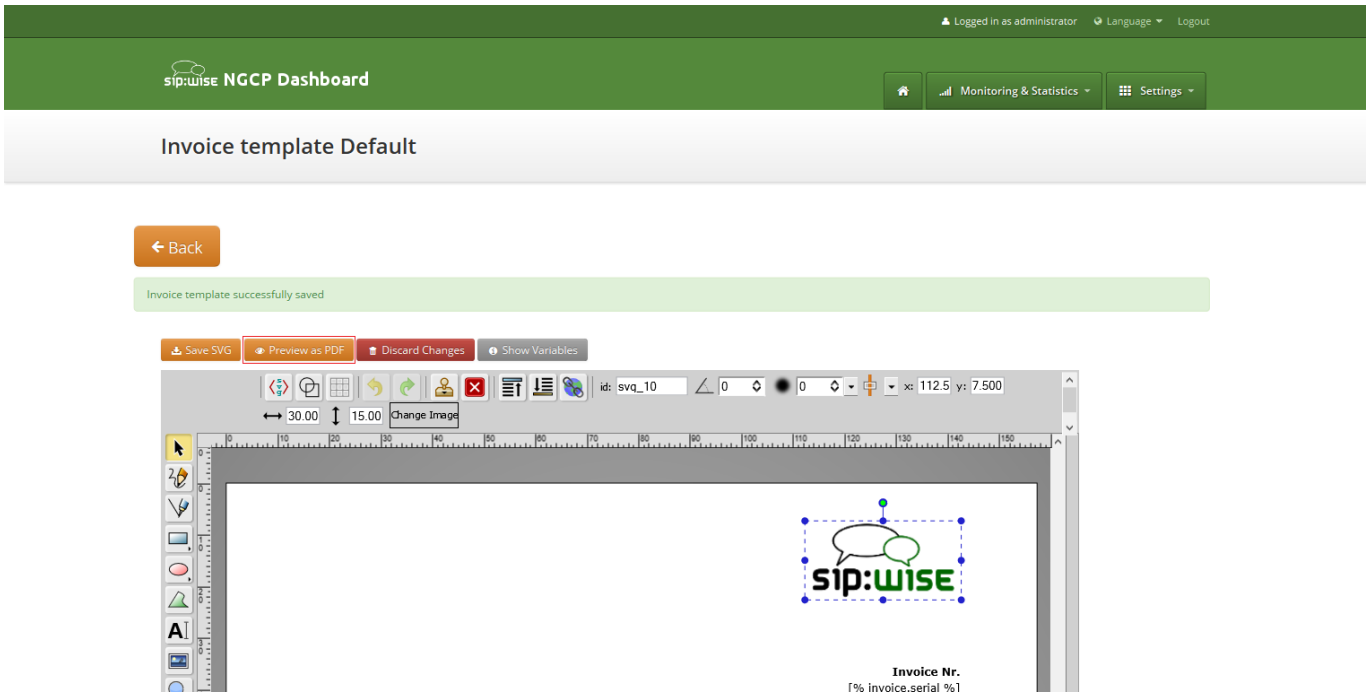
After image uploaded save invoice template changes Section 7.2.3.

7.2.3 Save and preview invoice template content.

To save invoice template content changes press button "Save SVG".



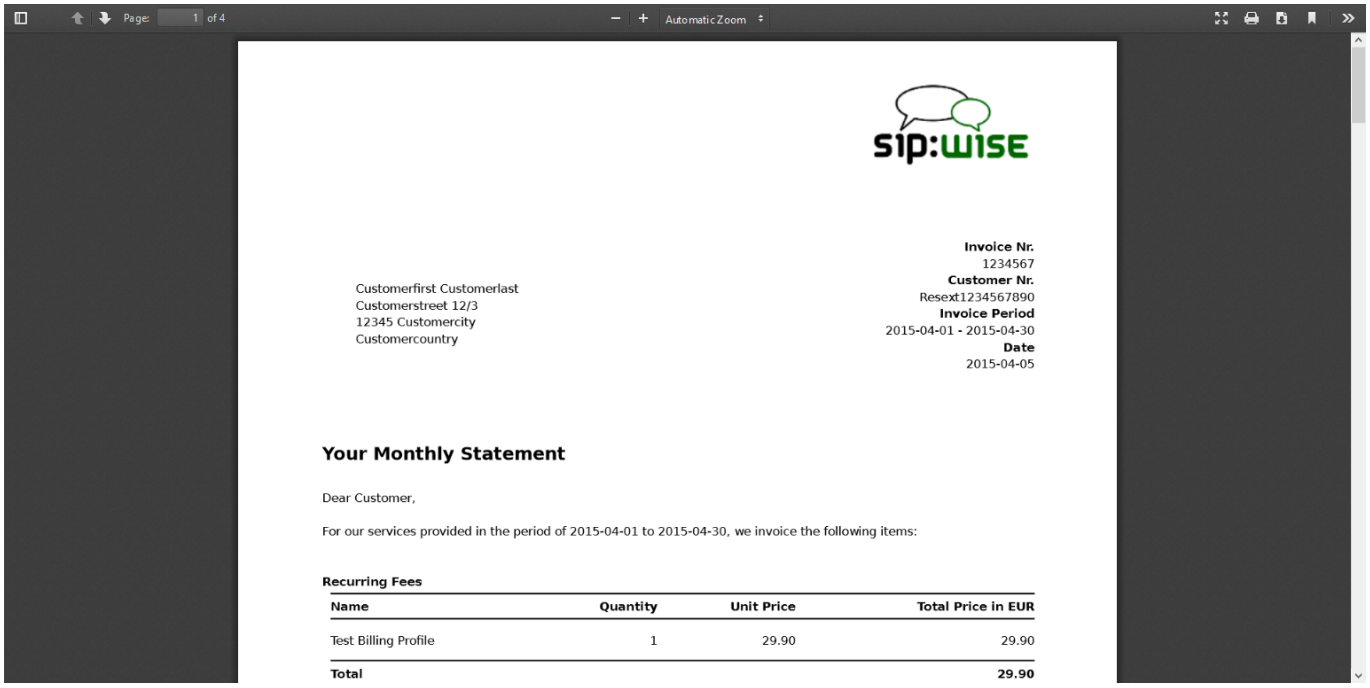
You will see message about successfully saved template. You can preview your invoice look in PDF format. Press on "Preview as PDF" button.



Invoice preview will be opened in the new window.

Note

Example fake data will be used for preview generation.



7.3 Invoices generation

Except invoices generation on demand using web interface, invoices can be generated automatically for all customers using cron and invoice generator script.

Also invoice generation script is responsible for the sending generated invoices to the customers.

Script is located at: /usr/share/ngcp-panel/tools/generate_invoices.pl

In short:

- To generate and immediately send invoices for the previous month:

```
perl /usr/share/ngcp-panel/tools/generate_invoice.pl --send --prevmonth
```

- To generate invoices for the previous month without sending:

```
perl /usr/share/ngcp-panel-tools/generate_invoice.pl --prevmonth
```

- To send already generated invoices for the previous month:

```
perl /usr/share/ngcp-panel/tools/generate_invoice.pl --sendonly --prevmonth
```

- Regenerate invoices for the specified period:

```
perl /usr/share/ngcp-panel/tools/generate_invoice.pl --stime="2015-01-01 00:00:00" ↩
     --etime="2015-01-31 00:00:00" --regenerate
```

Some not obvious options:

- **--allow_terminated** Generates invoices for the terminated contracts too.
- **--force_unrated** Generate invoices despite unrated calls existence in the specified generation period.
- **--no_empty** Skip invoices for the contracts without calls in the specified period and with null permanent fee for the billing profile.

To see all possible script options use --help or --man:

```
/usr/share/ngcp-panel/tools/generate_invoices.pl --man
```

Script will be run periodically as configured by the cron files. Cron files templates can be found at:

- /etc/ngcp-config/templates/etc/cron.d/ngcp-invoice-gen.tt2
- /etc/ngcp-config/templates/etc/cron.d/ngcp-invoice-gen.services

After applying your configuration cron file will be located at:

- /etc/cron.d/ngcp-invoice-gen

Script uses configuration file located at: /etc/ngcp-invoice-gen/invoice-gen.conf

Except common DB connection configuration following specific options can be defined in the config file:

- **RESELLER_ID** *1,2,3,...N*

Comma separated resellers id. Invoice generation will be performed only for the specified resellers.

- **CLIENT_CONTRACT_ID** *1,2,3,...N*

Comma separated customers id. Invoice generation will be performed only for the specified customers.

- **STIME** *YYYY-mm-DD HH:MM:SS*

Usually is not necessary. Script option --prevmonth will define correct start and end time for the previous month billing period. Generated invoices will include all calls with call start time more then STIME value and less the ETIME value.

- **ETIME** *YYYY-mm-DD HH:MM:SS*

Usually is not necessary. Script option --prevmonth will define correct start and end time for the previous month billing period. Generated invoices will include all calls with call start time more then STIME value and less the ETIME value.

- **SEND** *[0|1]*

Generated invoices will be immediately sent to the customers.

- **RESEND** *[0|1]*

Invoices, already sent to the customers, will be sent again.

- **REGENERATE** *[0|1]*

Already presented invoices files will be generated again. Otherwise they will stay intouched.

- **ALLOW_TERMINATED** *[0|1]*

Generate invoices for the already terminated customers too.

- **ADMIN_EMAIL** *your@email.com*

Purposed for notifications about invoices generation fails. Not in use now.

All generated invoices can be seen in the invoice management interface Section 7.1.

On request each invoice will be sent to the proper customer as e-mail with the invoice PDF in the attachment. Letter content is defined by the invoice email template.

8 Email templates

8.1 Email events

The sip:carrier allows to customize content of the emails sent on the following actions:

- Web password reset requested. Email will be sent to the subscriber, whom password was requested for resetting. If the subscriber doesn't have own email, letter will be sent to the customer, who owns the subscriber.
- New subscriber created. Email will be sent to the newly created subscriber or to the customer, who owns new subscriber.
- Letter with the invoice. Letter will be sent to the customer.

8.2 Initial template values and template variables

Default email templates for each of the email events are inserted on the initial sip:carrier database creation. Content of the default template is described in the appropriate sections. Default email templates aren't linked to any reseller and can't be changed through sip:carrier Panel. They will be used to initialize default templates for the newly created reseller.

Each email template refers to the values from the database using special mark-ups "[%" and "%]". Each email template has fixed set of the variables. Variables can't be added or changed without changes in the sip:carrier Panel code.

8.3 Password reset email template

Email will be sent after subscriber or subscriber administrator requested password reset for the subscriber account. Letter will be sent to the subscriber. If subscriber doesn't have own email, letter will be sent to the customer owning the subscriber.

Default content of the password reset email template is:

Template name	**passreset_default_email**
From	default@sipwise.com
Subject	Password reset email
Body	Dear Customer, Please go to [%url%] to set your password and log into your self-care interface. Your faithful Sipwise system -- This is an automatically generated message. Do not reply.

Following variables will be provided to the email template:

- [%**url**%]: specially generated url where subscriber can define his new password.
- [%**subscriber**%]: username@domain of the subscriber, which password was requested for reset.

8.4 New subscriber notification email template

Email will be sent on the new subscriber creation. Letter will be sent to the newly created subscriber if it has an email. Otherwise, letter will be sent to the customer who owns the subscriber.

Note
By default email content template is addressed to the customer. Please consider this when create the subscriber with an email.

Template name	**subscriber_default_email**
From	default@sipwise.com
Subject	Subscriber created
Body	Dear Customer, A new subscriber [%subscriber%] has been created for you. Your faithful Sipwise system -- This is an automatically generated message. Do not reply.

Following variables will be provided to the email template:

- [%**url**%]: specially generated url where subscriber can define his new password.
- [%**subscriber**%]: username@domain of the subscriber, which password was requested for reset.

8.5 Invoice email template

Template name	**invoice_default_email**
From	default@sipwise.com
Subject	Invoice #[%invoice.serial%] from [%invoice.period_start_obj.ymd%] to [%invoice.period_end_obj.ymd%]

Body	Dear Customer, Please find your invoice #[%invoice.serial%] for [%invoice. ↩ period_start_obj.month_name%], [%invoice.period_start_obj.year%] in attachment letter. Your faithful Sipwise system -- This is an automatically generated message. Do not reply.

Variables passed to the email template:

- [%**invoice**%]: container variable for the invoice information.

Invoice fields

- [%invoice.**serial**%]
- [%invoice.**amount_net**%]
- [%invoice.**amount_vat**%]
- [%invoice.**amount_total**%]
- [%invoice.**period_start_obj**%]
- [%invoice.**period_end_obj**%]

The fields [%invoice.period_start_obj%] and [%invoice.period_end_obj%] provide methods of the perl package DateTime for the invoice start date and end date. Further information about DateTime can be obtained from the package documentation: man DateTime

---

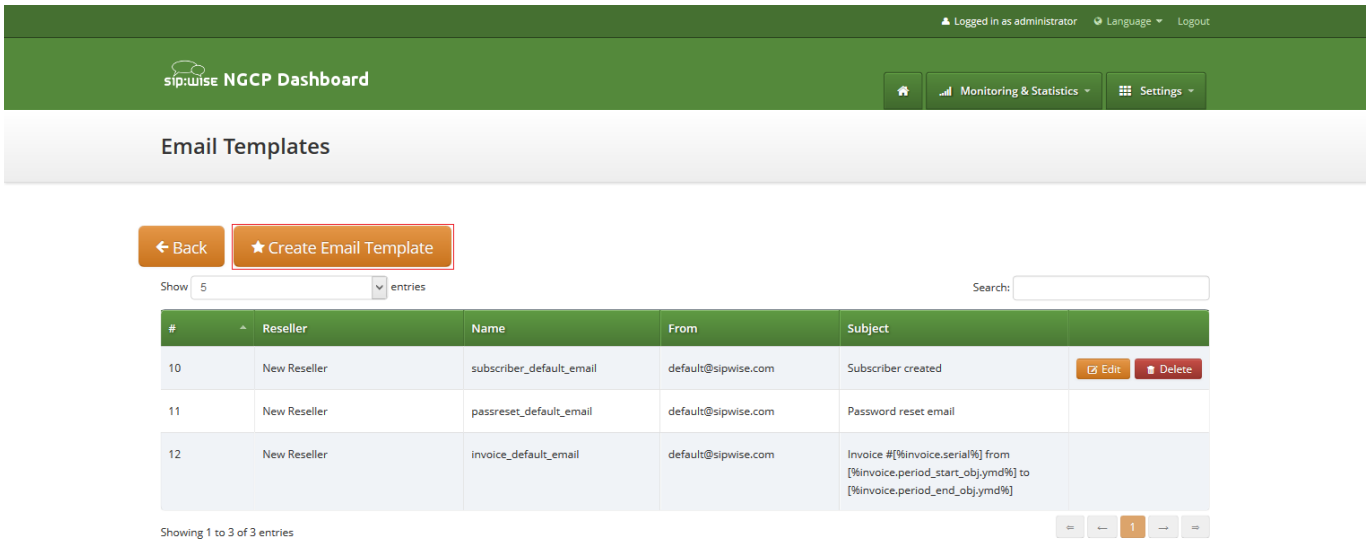
- [%**provider**%]: container variable for the reseller contact. All database contact values will be available.
- [%**client**%]: container variable for the customer contact.

Contact fields example for the "provider". Replace "provider" to client to access proper "customer" contact fields.

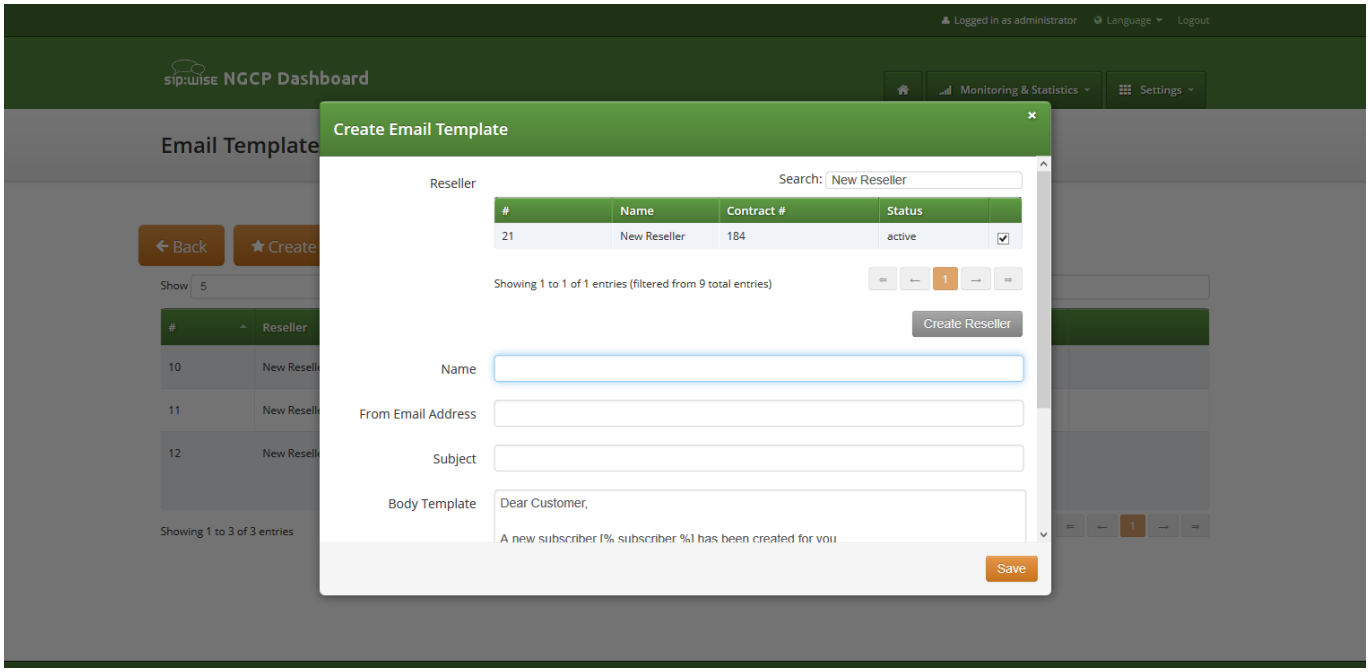
- [%provider.**gender**%]
- [%provider.**firstname**%]
- [%provider.**lastname**%]
- [%provider.**comregnum**%]
- [%provider.**company**%]
- [%provider.**street**%]
- [%provider.**postcode**%]
- [%provider.**city**%]
- [%provider.**country**%]
- [%provider.**phonenumber**%]
- [%provider.**mobilenumber**%]
- [%provider.**email**%]
- [%provider.**newsletter**%]
- [%provider.**faxnumber**%]
- [%provider.**iban**%]
- [%provider.**bic**%]
- [%provider.**vatnum**%]
- [%provider.**bankname**%]
- [%provider.**gpp0** - provider.**gpp9**%]

8.6 Email templates management

Email templates linked to the resellers can be customized in the email templates management interface. For the administrative account email templates of all the resellers will be shown. Respectively for the reseller account only owned email templates will be shown.



To create create new email template press button "Create Email Template".



On the email template form all fields are mandatory:

- **Reseller**: reseller who owns this email template.
- **Name**: currently only email template with the following names will be considered by the sip:carrier on the appropriate event Section 8.1 :
  - passreset_default_email;
  - subscriber_default_email;

  - invoice_default_email;

- **From Email Address**: email address which will be used in the From field in the letter sent by the sip:carrier.
- **Subject**: Template of the email subject. Subject will be processed with the same template variables as the email body.
- **Body**: Email text template. Will be processed with appropriate template variables.

9 Provisioning interfaces

The sip:carrier provides two kinds of provisioning interfaces for easy interconnection with 3rd party tools. The one recommended by Sipwise is the REST API, and the other (soon deprecated) one is SOAP and XMLRPC. Any new functionality is only added to the REST interface, so do not base any new development on SOAP or XMLRPC.

9.1 REST API

The sip:carrier provides a REST API to provision various functionality of the platform. The entry point - and at the same time the official documentation - is at https://<your-ip>:1443/api. It allows both administrators and resellers (in a limited scope) to manage the system.

You can either authenticate via username and password of your administrative account you're using to access the admin panel, or via SSL client certificates. Find out more about client certificate authentication in the online api documentation.

9.2 SOAP and XMLRPC API

Important

SOAP and XMLRPC API are deprecated and disabled by default since mr3.6.1. Please consider using REST API as SOAP and XMLRPC API will be deleted in upcoming release(s). To enable SOAP and XMLRPC change */etc/ngcp-config/config.yml* by setting *ossbss→frontend→fcgi* and execute *ngcpcfg apply*.

The sip:carrier provides two (soon deprecated) XML based provisioning interfaces - SOAP and XMLRPC. The server provides online documentation about all the functions available. To access the online documentation for the first time, you need to follow the following instructions:

- Generate a password for http access to the provisioning interfaces:

```
htpasswd -nbs myuser mypassword
```

Note

Also see `man 1 htpasswd` on how to generate crypt or MD5 passwords if you like. Of course you may use any other process to generate crypt, MD5 or SHA hashed passwords. But using `htpasswd` ensures the hashes are also understood by Nginx. To install htpasswd please run `apt-get install apache2-utils` on your system.

- Edit */etc/ngcp-config/config.yml*. Under section *ossbss→htpasswd*, replace *user* and *pass* with your new values and execute *ngcpcfg apply* as usual.
- Access https://<ip>:2443/SOAP/Provisioning.wsdl and login with your new credentials.

Note

The default port for provisioning interfaces is 2443. You can change it in */etc/ngcp-config/config.yml* by modifying *oss-bss→apache→port* and execute *ngcpcfg apply*.

Important

The displayed online API documentation shows all the currently available functionalities. Enabling or disabling features in */etc/ngcp-config/config.yml* will directly reflect in the functions being available via the APIs.

Important

If your SOAP client throws errors because of the inline <documentation> tags (e.g. Visual Studio and the stock PHP SOAP client complain about this), try to use the WSDL URL https://<ip>:2443/SOAP/Provisioning.wdsl?plain instead, which supresses the output of these tags.

10 Configuration Framework

The sip:carrier provides a configuration framework for consistent and easy to use low level settings management. A basic usage of the configuration framework only needs two actions already used in previous chapters:

- Edit */etc/ngcp-config/config.yml* file.
- Execute *ngcpcfg apply* command.

Low level management of the configuration framework might be required by advanced users though. This chapter explains the architecture and usage of the NGCP configuration framework. If the basic usage explained above fits your needs, feel free to skip this chapter and return to it when your requirements change.

A more detailed workflow of the configuration framework for creating a configuration file consists of 6 steps:

- Generation or editing of configuration templates and/or configuration values.
- Generation of the configuration files based on configuration templates and configuration values defined in config.yml, constants.yml and network.yml files.
- Execution of *prebuild* commands if defined for a particular configuration file or configuration directory.
- Placement of the generated configuration file in the target directory. This step is called *build* in the configuration framework.
- Execution of *postbuild* commands if defined for that configuration file or configuration directory.
- Execution of *services* commands if defined for that configuration file or configuration directory. This step is called *services* in the configuration framework.
- Saving of the generated changes. This step is called *commit* in the configuration framework.

10.1 Configuration templates

The sip:carrier provides configuration file templates for most of the services it runs. These templates are stored in the directory */etc/ngcp-config/templates*.

Example: Template files for */etc/ngcp-sems/sems.conf* are stored in */etc/ngcp-config/templates/etc/ngcp-sems/*.

There are different types of files in this template framework, which are described below.

10.1.1 .tt2 and .customtt.tt2 files

These files are the main template files that will be used to generate the final configuration file for the running service. They contain all the configuration options needed for a running sip:carrier system. The configuration framework will combine these files with the values provided by *config.yml*, *constants.yml* and *network.yml* to generate the appropriate configuration file.

Example: In the installation chapter we've changed the public interface from *lo* to *eth0*. This parameter will for example change kamailio's listen address, when the configuration file is generated. A quick look to the template file under */etc/ngcp-config/templates/etc/kama* will show a line like this:

```
listen=udp:[% ip %]:[% kamailio.lb.port %]
```

After applying the changes with the *ngcpcfg apply* command, a new configuration file will be created under */etc/kamailio/kamailio.cfg* with the proper values taken from the main configuration files (in this case *network.yml*):

```
listen=udp:1.2.3.4:5060
```

All the low-level configuration is provided by these .tt2 template files and the corresponding config.yml file. Anyways, advanced users might require a more particular configuration.

Instead of editing .tt2 files, the configuration framework recognises .customtt.tt2 files. These files are the same as .tt2, but they have higher priority when the configuration framework creates the final configuration files. An advanced user should create a .customtt.tt2 file from a copy of the corresponding .tt2 template and leave the .tt2 template untouched. This way, the user will have his personalized configuration and the system will continue providing a working, updated configuration template in .tt2 format.

Example: We'll create */etc/ngcp-config/templates/etc/kamailio.cfg.customtt.tt2* and use it for our personalized configuration. In this example, we'll just append a comment at the end of the template.

```
cd /etc/ngcp-config/templates/etc/kamailio/lb
cp kamailio.cfg.tt2 kamailio.cfg.customtt.tt2
echo '# This is my last line comment' >> kamailio.cfg.customtt.tt2
ngcpcfg apply
```

The ngcpcfg command will generate */etc/kamailio/kamailio.cfg* from our custom template instead of the general one.

```
tail -1 /etc/kamailio/kamailio.cfg
# This is my last line comment
```

Tip

The tt2 files use the Template Toolkit language. Therefore you can use all the feature this excellent toolkit provides within ngcpcfg's template files (all the ones with the .tt2 suffix).

10.1.2 .prebuild and .postbuild files

After creating the configuration files, the configuration framework can execute some commands before and after placing that file in its target directory. These commands usually are used for changing the file's owner, groups, or any other attributes. There are some rules these commands need to match:

- They have to be placed in a *.prebuild* or *.postbuild* file in the same path as the original *.tt2* file.
- The file name must be the same as the configuration file, but having the mentioned suffixes.
- The commands must be *bash* compatible.
- The commands must return 0 if successful.

- The target configuration file is matched by the environment variable *output_file*.

Example: We need *www-data* as owner of the configuration file */etc/ngcp-ossbss/provisioning.conf*. The configuration framework will by default create the configuration files with root:root as owner:group and with the same permissions (rwx) as the original template. For this particular example, we will change the owner of the generated file using the *.postbuild* mechanism.

```
echo 'chgrp www-data ${output_file}' \
  > /etc/ngcp-config/templates/etc/ngcp-ossbss/provisioning.conf.postbuild
```

10.1.3 .services files

.services files are pretty similar and might contain commands that will be executed after the *build* process. There are two types of *.services* files:

- The particular one, with the same name as the configuration file it is associated to.
  Example: */etc/ngcp-config/templates/etc/asterisk/sip.conf.services* is associated to */etc/asterisk/sip.conf*
- The general one, named *ngcpcfg.services* wich is associated to every file in its target directory.
  Example: */etc/ngcp-config/templates/etc/asterisk/ngcpcfg.services* is associated to every file under */etc/asterisk/*

When the *services* step is triggered all *.services* files associated to a changed configuration file will be executed. In case of the general file, any change to any of the configuration files in the directory will trigger the execution of the commands.

Tip

If the service script has the execute flags set (*chmod +x $file*) it will be invoked directly. If it doesn't have execute flags set it will be invoked under bash. Make sure the script is bash compatible if you do not set execute permissions on the service file.

These commands are usually service reload/restarts to ensure the new configuration has been loaded by running services.

Note

The configuration files mentioned in the following example usually already exist on the platform. Please make sure you don't overwrite any existing files if following this example.

Example:

```
echo '/etc/init.d/mysql restart' \
  > /etc/ngcpcfg-config/templates/etc/mysql/my.cnf.services
echo '/etc/init.d/asterisk restart' \
  > /etc/ngcpcfg-config/templates/etc/asterisk/ngcpcfg.services
```

In this example we created two *.services* files. Now, each time we trigger a change to */etc/mysql.my.cnf* or to */etc/asterisk/** we'll see that MySQL or Asterisk services will be restarted by the ngcpcfg system.

10.2 config.yml, constants.yml and network.yml files

The */etc/ngcp-config/config.yml* file contains all the user-configurable options, using the YAML (YAML Ain't Markup Language) syntax.

The */etc/ngcp-config/constants.yml* file provides configuration options for the platform that aren't supposed to be edited by the user. Do not manually edit this file unless you really know what you're doing.

The */etc/ngcp-config/network.yml* file provides configuration options for all interfaces and IP addresses on those interfaces. You can use the *ngcp-network* tool for conveniently change settings without having to manually edit this file.

The */etc/ngcp-config/ngcpcfg.cfg* file is the main configuration file for ngcpcfg itself. Do not manually edit this file unless you really know what you're doing.

10.3 ngcpcfg and its command line options

The shared storage used by all nodes is the shared storage of the mgmt pair.

The ngcpcfg utility supports the following command line options:

10.3.1 apply

The *apply* option is a short-cut for the options "build && services && commit" and also executes *etckeeper* to record any modified files inside /etc. It is the recommended option to use the ngcpcfg framework unless you want to execute any specific commands as documented below.

10.3.2 build

The *build* option generates (and therefore also updates) configuration files based on their configuration (config.yml) and template files (.tt2). Before the configuration file is generated a present .prebuild will be executed, after generation of the configuration file the according .postbuild script (if present) will be executed. If a *file* or *directory* is specified as argument the build will generate only the specified configuration file/directory instead of running through all present templates.

Example: to generate only the file /etc/nginx/sites-available/ngcp-panel you can execute:

```
ngcpcfg build /etc/nginx/sites-available/ngcp-panel
```

Example: to generate all the files located inside the directory /etc/nginx/ you can execute:

```
ngcpcfg build /etc/nginx/
```

10.3.3 commit

The *commit* option records any changes done to the configuration tree inside */etc/ngcp-config*. The commit option should be executed when you've modified anything inside the configuration tree.

10.3.4 decrypt

Decrypt /etc/ngcp-config-crypted.tgz.gpg and restore configuration files, doing the reverse operation of the *encrypt* option. Note: This feature is only available if the ngcp-ngcpcfg-locker package is installed.

10.3.5 diff

Show uncommited changes between ngcpcfg's Git repository and the working tree inside */etc/ngcp-config*. Iff the tool doesn't report anything it means that there are no uncommited changes. If the *--addremove* option is specified then new and removed files (iff present) that are not yet (un)registered to the repository will be reported, no further diff actions will be executed then. Note: This option is available since ngcp-ngcpcfg version 0.11.0.

10.3.6 encrypt

Encrypt /etc/ngcp-config and all resulting configuration files with a user defined password and save the result as /etc/ngcp-config-crypted.tgz.gpg. Note: This feature is only available if the ngcp-ngcpcfg-locker package is installed.

10.3.7 help

The *help* options displays ngcpcfg's help screen and then exits without any further actions.

10.3.8 initialise

The *initialise* option sets up the ngcpcfg framework. This option is automatically executed by the installer for you, so you shouldn't have to use this option in normal operations mode.

10.3.9 pull

Retrieve modifications from shared storage. Note: This option is available in the High Availability setup only.

10.3.10 push

Push modifications to shared storage and remote systems. After changes have been pushed to the nodes the *build* option will be executed on each remote system to rebuild the configuration files (unless the --nobuild has been specified, then the build step will be skipped). If hostname(s) or IP address(es) is given as argument then the changes will be pushed to the shared storage and to the given hosts only. You can use *all* as a shortcut to push to the other nodes. If no host has been specified then the hosts specified in */etc/ngcp-config/systems.cfg* are used. Note: This option is available in the High Availability setup only.

10.3.11 services

The *services* option executes the service handlers for any modified configuration file(s)/directory.

10.3.12 status

The *status* option provides a human readable interface to check the state of the configuration tree. If you are unsure what should be done as next step or if want to check the current state of the configuration tree just invoke *ngcpcfg status*.

If everything is OK and nothing needs to be done the output should look like:

```
# ngcpcfg status
Checking state of ngcpcfg:
OK:   has been initialised already (without shared storage)
Checking state of configuration files:
OK:   nothing to commit.
Checking state of /etc files
OK:   nothing to commit.
```

If the output doesn't say "OK" just follow the instructions provided by the output of *ngcpcfg status*.

Further details regarding the ngcpcfg tool are available through *man ngcpcfg* on the Sipwise Next Generation Platform.

11 Network Configuration

Starting with version 2.7, the sip:carrier uses a dedicated *network.yml* file to configure the IP addresses of the system. The reason for this is to be able to access all IPs of all nodes for all services from any particular node in case of a distributed system on one hand, and in order to be able the generate */etc/network/interfaces* automatically for all nodes based on this central configuration file.

11.1 General Structure

The basic structure of the file looks like this:

```
hosts:
  self:
    role:
      - proxy
      - lb
      - mgmt
    interfaces:
      - eth0
      - lo
    eth0:
      ip: 192.168.51.213
      netmask: 255.255.255.0
      type:
        - sip_ext
        - rtp_ext
        - web_ext
        - web_int
    lo:
      ip: 127.0.0.1
      netmask: 255.255.255.0
      type:
        - sip_int
        - ha_int
```

11.2 Available Host Options

There are three different main sections for a host in the config file, which are *role*, *interfaces* and the actual interface definitions.

- *role*: The role setting is an array defining which logical roles a node will act as. Possible entries for this setting are:
  - *mgmt*: This entry means the host is acting as management node for the platform. In a sip:carrier, this option must always been set. The management node exposes the admin and csc panels to the users and the APIs to external applications and is used to export CDRs.

  - *lb*: This entry means the host is acting as SIP load-balancer for the platform. In a sip:carrier, this option must always been set. The SIP load-balancer acts as an ingress and egress point for all SIP traffic to and from the platform.
  - *proxy*: This entry means the host is acting as SIP proxy for the platform. In a sip:carrier, this option must always been set. The SIP proxy acts as registrar, proxy and application server and media relay, and is responsible for providing the features for all subscribers provisioned on it.

- *interfaces*: The interfaces setting is an array defining all interface names in the system. The actual interface details are set in the actual interface settings below.
- *<interface name>*: After the interfaces are defined in the *interfaces* setting, each of those interfaces needs to be specified as a separate setting with the following options:
  - *ip*
  - *netmask*
  - *advertised_ip*
  - *type*

There are different *interface types*, which define the services on a particular *interface*. For example the type *ssh_ext* set for a specific interface defines that the SSH daemon will listen on that interface for incoming connections. The list of possible types is as follows (note that you can assign a type only once per node):

- *mon_ext*: interface for monitoring purposes, e.g. for snmpd
- *rtp_ext*: interface for external RTP relay
- *sip_ext*: interface for external SIP communication between the sip:carrier and the end points
- *sip_ext_incoming*: extra listen interface for external SIP traffic (optional)
- *sip_int*: interface for internal SIP communication, e.g. between load-balancer, proxy and application servers
- *ssh_ext*: interface for SSH remote login
- *web_ext*: interface for the subscriber web panel and the subscriber's SOAP/REST APIs
- *web_int*: interface for the administrator web panel, his SOAP/REST APIs and internal API communication
- *aux_ext*: interface for potentially insecure external components like rsyslogd service; e.g. the CloudPBX module can use those services to provide time services and remote logging facilities to end customer devices. The type *aux_ext* is assigned to *lo* interface by default. If it is needed to expose this type to the public, it is recommended to assign the type *aux_ext* to a separate VLAN interface to be able to limit or even block the incoming traffic easily via firewalling in case of emergency, like a (D)DOS attack on rsyslog services.

12 Advanced Network Configuration

You have a typical deployment now and you are good to go, however you may need to do extra configuration depending on the devices you are using and functionality you want to achieve.

12.1 Extra SIP Sockets

By default, the load-balancer listens on the UDP and TCP ports 5060 (*kamailio→lb→port*) and TLS port 5061 (*kamailio→lb→tls→port*). If you need to setup one or more extra SIP listening ports or IP addresses in addition to those standard ports, please edit the *kamailio→lb→extra_sockets* option in your */etc/ngcp-config/config.yml* file.

The correct format consists of a label and value like this:

```
    extra_sockets:
      port_5064: udp:10.15.20.108:5064
      test: udp:10.15.20.108:6060
```

The label is shown in the `outbound_socket` peer preference (if you want to route calls to the specific peer out via specific socket); the value must contain a transport specification as in example above (udp, tcp or tls). After adding execute `ngcpcfg apply`:

```
ngcpcfg apply 'added extra socket' && ngcpcfg push all
```

The direction of communication through this SIP extra socket is incoming+outgoing. The sip:carrier will answer the incoming client registrations and other methods sent to the extra socket. For such incoming communication no configuration is needed. For the outgoing communication the new socket must be selected in the `outbound_socket` peer preference. For more details read until the end of next chapter Section 12.2 that covers peer configuration for SIP and RTP in greater detail.

Important

In this section you have just added an extra SIP socket. RTP traffic will still use your *rtp_ext* IP address.

12.2 Extra SIP and RTP Sockets

If you want to use an additional interface (with a different IP address) for SIP signalling and RTP traffic you need to add your new interface in the */etc/network/interfaces* file. Also the interface must be declared in */etc/ngcp-config/network.yml*.

Suppose we need to add a new SIP socket and a new RTP socket on VLAN 100. You can use the *ngcp-network* tool for adding interfaces without having to manually edit this file:

```
ngcp-network --set-interface=eth0.100 --host=slb01a --ip=auto --netmask=auto --type= ↩
    sip_ext_incoming
ngcp-network --set-interface=eth0.100 --host=slb01b --ip=auto --netmask=auto --type= ↩
    sip_ext_incoming
ngcp-network --set-interface=eth0.100 --host=prx01a --ip=auto --netmask=auto --type= ↩
    rtp_int_100
ngcp-network --set-interface=eth0.100 --host=prx01b --ip=auto --netmask=auto --type= ↩
    rtp_int_100
```

The generated file should look like the following:

```
slb01a:
..
..
    eth0.100:
      hwaddr: ff:ff:ff:ff:ff:ff
      ip: 192.168.1.2
      netmask: 255.255.255.0
      shared_ip:
        - 192.168.1.3
      shared_v6ip: ~
      type:
        - sip_ext_incoming
..
..
    interfaces:
      - lo
      - eth0
      - eth0.100
      - eth1
..
..
prx01a:
..
..
    eth0.100:
      hwaddr: ff:ff:ff:ff:ff:ff
      ip: 192.168.1.20
      netmask: 255.255.255.0
      shared_ip:
        - 192.168.1.30
      shared_v6ip: ~
      type:
        - rtp_int_100
..
..
    interfaces:
      - lo
      - eth0
      - eth0.100
      - eth1
..
..
slb01b:
..
..
    eth0.100:
      hwaddr: ff:ff:ff:ff:ff:ff
```

```
      ip: 192.168.1.4
      netmask: 255.255.255.0
      shared_ip:
        - 192.168.1.3
      shared_v6ip: ~
      type:
        - sip_ext_incoming
..
..
    interfaces:
      - lo
      - eth0
      - eth0.100
      - eth1
..
..
prx01b:
..
..
    eth0.100:
      hwaddr: ff:ff:ff:ff:ff:ff
      ip: 192.168.1.40
      netmask: 255.255.255.0
      shared_ip:
        - 192.168.1.30
      shared_v6ip: ~
      type:
        - rtp_int_100
..
..
    interfaces:
      - lo
      - eth0
      - eth0.100
      - eth1
```

As you can see from the above example, extra SIP interfaces must have type *sip_ext_incoming*. While *sip_ext* should be listed only once per host, there can be multiple *sip_ext_incoming* interfaces. The direction of communication through this SIP interface is incoming only. The sip:carrier will answer the incoming client registrations and other methods sent to this address and remember the interfaces used for clients' registrations to be able to send incoming calls to him from the same interface.

In order to use the interface for the outbound SIP communication it is necessary to add it to extra_sockets section in */etc/ngcp-config/config.yml* and select in the `outbound_socket` peer preference. So if using the above example we want to use the vlan100 IP as source interface towards a peer, the corresponding section may look like the following:

```
    extra_sockets:
      port_5064: udp:10.15.20.108:5064
      test: udp:10.15.20.108:6060
```

```
      int_100: udp:192.168.1.3:5060
```

The changes have to be applied:

```
ngcpcfg apply 'added extra SIP and RTP socket' && ngcpcfg push all
```

After applying the changes, a new SIP socket will listen on IP `192.168.1.3` on slb01 node and this socket can now be used as source socket to send SIP messages to your peer for example. In above example we used label *int_100*. So the new label "int_100" is now shown in the `outbound_socket` peer preference.

Also, RTP socket is now listening on `192.168.1.30` on prx01 node and you can choose the new RTP socket to use by setting parameter `rtp_interface` to the Label "int_100" in your Domain/Subscriber/Peer preferences.

13 Security and Maintenance

Once the sip:carrier is in production, security and maintenance becomes really important. In this chapter, we'll go through a set of best practices for any production system.

13.1 Sipwise SSH access to sip:carrier

The sip:carrier provides SSH access to the system for Sipwise operational team for debugging and final tuning. Operational team uses user *sipwise* which can be logged in through SSH key only (password access is disabled) from dedicated access server *jump.sipwise.com* only.

To completely remove Sipwise access to your system, please execute as user root:

```
root@myserver:~# ngcp-support-access --disable && apt-get install ngcp-support-noaccess
```

Note

you have to execute the command above on each node of your sip:carrier system!

Warning

please ensure that the script complete successfully:

```
 * Support access successfully disabled.
```

If you need to restore Sipwise access to the system, please execute as user root:

```
root@myserver:~# apt-get install ngcp-support-access && ngcp-support-access --enable
```

Warning

please ensure that the script complete successfully:

```
 * Support access successfully enabled.
```

13.2 Firewalling

The sip:carrier runs a wide range of services. Some of them need to interact with the user, while some others need to interact with the administrator or with nobody at all. Assuming that we trust the sip:carrier server for outgoing connections, we'll focus only on incoming traffic to define the services that need to be open for interaction.

Table 7: Subscribers

Service	Default port	Config option
Customer self care interface	443 TCP	www_admin→http_csc→port
SIP	5060 UDP, TCP	kamailio→lb→port
SIP over TLS	5061 TCP	kamailio→lb→tls→port + kamailio→lb→tls→enable
RTP	30000-40000 UDP	rtpproxy→minport + rtpproxy→maxport
XCAP	1080 TCP	kamailio→proxy→presence→enable + nginx→xcap_port
XMPP	5222 and 5269 TCP	None, standard XMPP ports for clients (5222) and federation (5269)

Table 8: Administrators

Service	Default port	Config option
SSH/SFTP	22 TCP	NA
Administrator interface	1443 TCP	www_admin→http_admin→port
Provisioning interfaces	2443 TCP	ossbss→apache→port

Caution

To function correctly, the *rtpengine* requires an additional *iptables* rule installed. This rule (with a target of `RTPENGINE`) is automatically installed and removed when the rtpengine starts and stops, so normally you don't need to worry about it. However, any 3rd party firewall solution can potentially flush out all existing iptables rules before installing its own, which would leave the system without the required `RTPENGINE` rule and this would lead to decreased performance. It is imperative that any 3rd party firewall solution either leaves this rule untouched, or installs it back into place after flushing all rules out. The complete parameters to install this rule (which needs to go into the `INPUT` chain of the `filter` table) are: `-p udp -j RTPENGINE --id 0`

13.3 Password management

The sip:carrier comes with some default passwords the user should change during the deployment of the system. They have been explained in the previous chapters of this document.

- The login for the system account *cdrexport* is disabled by default. Although this is a jailed account, it has access to sensitive information, namely the Call Detail Records of all calls. SSH keys should be used to login this user, or alternatively a really strong password should be used when setting the password via *passwd cdrexport*.

- The *root* user in MySQL has no default password. A password should be set using the *mysqladmin password* command.
- The administrative web interface has a default user *administrator* with password *administrator*. It should be changed within this interface.
- Generate new password for user *ngcpsoap* to access the provisioning interfaces, see the details in Section 9.

Important
Many NGCP services use MySQL backend. Users and passwords for these services are created during the installation. These passwords are unique for each installation, and the connections are restricted to localhost. You should not change these users and passwords.

13.4 SSL certificates.

The sip:carrier provides default, self-signed SSL certificates for SSL connections. These certificates are common for every installation. Before going to production state, the system administrator should provide SSL certificates for the web services. These certificates can either be shared by all web interfaces (*provisioning*, *administrator interface* and *customer self care interface*), or separate ones for each them can be used.

- Generate the certificates. The *customer self care interface* certificate should be signed by a certification authority to avoid browser warnings.
- Upload the certificates to the system
- Set the path to the new certificates in */etc/ngcp-config/config.yml*:
  - *ossbss→apache→autoprov→sslcertfile* and *ossbss→apache→autoprov→sslcertkeyfile* for the *provisioning interface*.
  - *ossbss→apache→restapi→sslcertfile* and *ossbss→apache→restapi→sslcertkeyfile* for the *REST interface*.
  - *www_admin→http_admin→sslcertfile* and *www_admin→http_admin→sslcertkeyfile* for the *admin interface*.
  - *www_admin→http_csc→sslcertfile* and *www_csc→http_csc→sslcertkeyfile* for the *customer self care interface*.
- Apply the configuration changes with *ngcpcfg apply*.

The sip:carrier also provides the self-signed SSL certificates for SIP over TLS services. The system administrator should replace them with certificates signed by a trusted certificate authority if he is going to enable it for the production usage (*kamailio→lb→tls→enable* (disabled by default)).

- Generate the certificates.
- Upload the certificates to the system
- Set the path to the new certificates in */etc/ngcp-config/config.yml*:
  - *kamailio→lb→tls→sslcertfile* and *kamailio→lb→tls→sslcertkeyfile* .
- Apply the configuration changes with *ngcpcfg apply*.

13.5 sip:carrier Backup

For any service provider it is important to maintain a reliable backup policy as it enables prompt services restoration after any force majeure event. Although the design of sip:carrier implies data duplication and high availability of services, we still strongly suggest you to configure a backup procedure. The sip:carrier has a built-in solution that can help you back up the most crucial data. Alternatively, it can be integrated with any Debian compatible backup software.

13.5.1 What data to back up

- The database

This is the most important data in the system. All subscriber and billing information, CDRs, user preferences, etc. are stored in the MySQL server. It is strongly recommended to have up-to-date dumps of all the databases on corresponding NGCP nodes.

- System configuration

The system configuration files such as */etc/mysql/sipwise.cnf* and the */etc/ngcp-config/* directory should be included in the backup as well. We suggest backing up the whole */etc* folder.

- Exported CDRs (optional)

The */home/jail/home/cdrexport* directory contains the exported CDRs. It depends on your call data retention policy whether or not to remove these files after exporting them to an external system.

13.5.2 The built-in backup solution

The sip:carrier comes with an easy-to-use solution that creates everyday backups of the most important data:

- The system configuration files. The whole /etc directory is backed up.
- Exported CDRs. The */home/jail/home/cdrexport* directory with csv files.
- All required databases on corresponding servers.

This functionality is disabled by default and can be enabled and configured in the *backuptools* subsection in the *config.yml* file. Please, refer to the NGCP configuration overview - backuptools Section C.1.2 section for the backup configuration options.

Once you set the required configuration options, apply the changes:

```
ngcpcfg apply 'enabled the backup feature'
ngcpcfg push all
```

Once you activate the feature, the sip:carrier will create backups in the off-peak time on the standby nodes and put them to the */var/backup/ngcp_backup* directory. You can copy these files to your backup server using scp or ftp.

Note
make sure that you have enough free disk space to store the backups for the specified number of days.

13.6 Recovery

In the worst case scenario, when the system needs to be recovered from a total loss, you only need 4 steps to get the services back online:

- Install the sip:carrier as explained in chapter 2.
- Restore the */etc/ngcp-config/* directory and the */etc/mysql/sipwise.cnf* file from the backup, overwriting your local files.
- Restore the database from the latest MySQL dump.
- Apply the changes to bring the original configuration into effect:

```
ngcpcfg apply 'restored the system from the backup'
ngcpcfg push all
```

13.7 Reset Database

Important
All existing data will be wiped out! Use this script only if you want to clear all previously configured services and start configuration from scratch.

To reset database to its original state you can execute a script: `ngcp-reset-db`. It will assign new unique passwords for the NGCP services and reset all services. The script will also create dumps for all NGCP databases.

13.8 System requirements and performance

The sip:carrier is a very flexible system, capable of serving from hundreds to several tens of thousands of subscribers in a single node. The system comes with a default configuration, capable of serving up to 50.000 subscribers in a *normal* environment. But there is no such thing as a *normal* environment. And the sip:carrier has sometimes to be tunned for special environments, special hardware requirements or just growing traffic.

Note
If you have performance issues with regards to disk I/O please consider enabling the *noatime* mount option for the root filesystem. Sipwise recommends the usage of *noatime*, though remove it if you use software which conflicts with its presence.

In this section some parameters will be explained to allow the sip:carrier administrator tune the system requirements for optimum performance.

Table 9: Requirement_options

Option	Default value	Requirement impact
cleanuptools→binlog_days	15	Heavy impact on the harddisk storage needed for mysql logs. It can help to restore the database from backups or restore broken replication.
database→bufferpoolsize	64MB	For test systems or low RAM systems, lowering this setting is one of the most effective ways of releasing RAM. The administrator can check the innodb buffer hit rate on production systems; a hit rate over 99% is desired to avoid bottlenecks.
kamailio→lb→pkg_mem	16	This setting affects the amount of RAM the system will use. Each kamailio-lb worker will have this amount of RAM reserved. Lowering this setting up to 8 will help to release some memory depending on the number of kamailio-lb workers running. This can be a dangerous setting as the lb process could run out of memory. Use with caution.
kamailio→lb→shm_mem	1/16 * Total System RAM	The installer will set this value to 1/16 of the total system RAM. This setting does not change even if the system RAM does so it's up to the administrator to tune it. It has been calculated that 1024 (1GB) is a good value for 50K subscriber environment. For a test environment, setting the value to 64 should be enough. "Out of memory" messages in the kamailio log can indicate that this value needs to be raised.
kamailio→lb→tcp_children	8	Number of TCP workers kamailio-lb will spawn per listening socket. The value should be fine for a mixed UDP-TCP 50K subscriber system. Lowering this setting can free some RAM as the number of kamailio processess would decrease. For a test system or a pure UDP subscriber system 2 is a good value. 1 or 2 TCP workers are always needed.
kamailio→lb→tls→enable	yes	Enable or not TLS signaling on the system. Setting this value to "no" will prevent kamailio to spawn TLS listening workers and free some RAM.
kamailio→lb→udp_children	8	See *kamailio→lb→tcp_children* explanation
kamailio→proxy→children	8	See *kamailio→lb→tcp_children* explanation. In this case the proxy only listens udp so these children should be enough to handle all the traffic. It could be set to 2 for test systems to lower the requirements.
kamailio→proxy→*_expires		Set the default and the max and min registration interval. The lower it is more REGISTER requests will be handled by the lb and the proxy. It can impact in the network traffic, RAM and CPU usage.
kamailio→proxy→natping_interval	30	Interval for the proxy to send a NAT keepalive OPTIONS message to the nated subscriber. If decreased, this setting will increase the number of OPTIONS requests the proxy needs to send and can impact in the network traffic and the number of natping processes the system needs to run. See *kamailio→proxy→natping_processes* explanation.

Table 9: (continued)

Option	Default value	Requirement impact
kamailio→proxy→natping_processes	7	Kamailio-proxy will spawn this number of processes to send keepalive OPTIONS to the nated subscribers. Each worker can handle about 250 messages/second (depends on the hardware). Depending the number of nated subscribers and the *kamailio→proxy→natping_interval* parameter the number of workers may need to be adjusted. The number can be calculated like nated_subscribers/natping_interval/pings_per_second_per_process. For the default options, asuming 50K nated subscribers in the system the parameter value would be 50.000/30/250 = (6,66) 7 workers. 7 is the maximum number of processes kamailio will accept. Raising this value will cause kamailio not to start.
kamailio→proxy→shm_mem	1/16 * Total System RAM	See *kamailio→lb→shm_mem* explanation.
rateomat→enable	yes	Set this to no if the system shouldn't perform rating on the CDRs. This will save CPU usage.
rsyslog→external_log	0	If enabled, the system will send the log messages to an external server. Depending on the *rsyslog→external_loglevel* parameter this can increase dramatically the network traffic.
rsyslog→ngcp_logs_preserve_days	93	This setting will set the number of days ngcp logs under */var/log/ngcp* will be kept in disk. Lowering this setting will free a high amount of disk space.

Tip

In case of using virtualized environment with limited amount of hardware resources, you can use the script *ngcp-toggle-performance-config* to adjust sip:carrier configuration for high/low performance:

```
root@spce:~# /usr/sbin/ngcp-toggle-performance-config
/usr/sbin/ngcp-toggle-performance-config - tool to adjust sip:provider configuration for ↩
    low/high performance

  --help                Display this usage information
  --high-performance    Adjust configuration for system with normal/high performance
  --low-performance     Adjust configuration for system with low performance (e.g. VMs)

root@spce:~#
```

13.9 Troubleshooting

The sip:carrier platform provides detailed logging and log files for each component included in the system via rsyslog. The main folder for log files is */var/log/ngcp/*, it contains a list of self explanatory log files named by component name.

The sip:carrier is a high performance system which requires compromise between traceability (maximum amount of debug information being written to hard drive) and productivity (minimum load on IO subsystem). This is the reason why different log levels are configured for the provided components by default.

Most log files are designed for debugging sip:carrier by Sipwise operational team while main log files for daily routine usage are:

Log file	Content	Estimated size
/var/log/ngcp/api.log	API logs providing type and content of API requests and responses as well as potential errors	medium
/var/log/ngcp/panel.log	Admin Web UI logs when performing operational tasks on the ngcp-panel	medium
/var/log/ngcp/cdr.log	mediation and rating logs, e.g. how many CDRs have been generated and potential errors in case of CDR generation or rating fails for particular accounting data	medium

Log file	Content	Estimated size
/var/log/ngcp/ha.log	fail-over related logs in case a node in a pair loses connection to the other side, when a standby node takes over or an active node goes standby due to intra-node communication issues or external ping node connection issues	small
/var/log/ngcp/kamailio-lb.log	Overview of SIP requests and replies along with network source and destination information flowing through the platform	huge

Warning

it is highly NOT recommended to change default log levels as it can cause system IO overloading which will affect call processing.

Note

the exact size of log files depend on system type, system load, system health status and system configuration, so cannot be estimated with high precision. Additionally operational network parameters like ASR and ALOC may impact the log files' size significantly.

14 Monitoring and Alerting

14.1 Internal Monitoring

The platform uses the *monit* daemon internally to monitor all essential services. Since the sip:carrier runs in an active/standby mode, not all services are always running on both nodes, some of them will only run on the active node and be stopped on the standby node. At any time, you can use the command `monit summary` to get a list of all services and their current status, or `monit status` for the same list with more detail.

Important

sip:carrier has a monit services dependencies since mr3.5.1. Services specified in a depend statement will be checked during stop/start/monitor/unmonitor operations. If a service is stopped or unmonitored it will stop/unmonitor any services that depends on itself. Which means that kamailio/sbc/asterisk/prosody/... will be stopped on `monit stop mysql` operation.

The *monit* daemon takes care of quickly restarting a service should it ever fail for whatever reason. When that happens, the deamon will send a notification email to the address specified in the `config.yml` file under the key `general.adminmail`. It will also send warning emails to this address under certain abnormal conditions, such as when the system is low on memory (> 75% used) or under high-load conditions.

Important

In order for *monit* to be able to send email to the specified address, the local MTA (*exim4*) must be configured correctly. If you haven't done so already, run `dpkg-reconfigure exim4-config` to do this. The CE edition's handbook contains more information about this in the *Installation* chapter.

14.2 Statistics Dashboard

The platform's administration interface (described in Section 3) provides a simple graphical overview of the most important system health data points, such as memory usage, load averages and disk usage, as well as statistics about the VoIP system itself, such as the number of concurrent active calls, number of provisioned and registered subscribers, etc.

14.3 External Monitoring Using SNMP

14.3.1 Overview and Initial Setup

The sip:carrier exports a variety of system health data and statistics over standard SNMP. By default, the SNMP interface can only be accessed locally. To make it possible to poll the SNMP data from an external system, the `config.yml` file needs to be edited and the list of allowed community names and allowed hosts/IP ranges must be populated. This list can be found under the `checktools.snmpd.communities` key and consists of one or more `community`/`source` value pairs. The `community` is the SNMP community string to be allowed, while `source` is the IP address or IP block to allow this community from. A `source`

of `default` equals the IP address `127.0.0.1`. Other legal values are single IP addresses or IP blocks in IP/prefix notation, for example `192.168.115.0/24`. It is recommended that you leave the default entry (`public` and `default`) in place for local testing of SNMP functionality.

Tip

To locally check if SNMP is working correctly, execute the command `snmpwalk -v2c -cpublic localhost .` (note the trailing dot), assuming the default SNMP community entry has been left in place. This will generate a long list of raw SNMP OIDs and their values.

Tip

SNMP version 1 and version 2c are supported.

14.3.2 Details

All basic system health variables (such as memory, disk, swap, CPU usage, network statistics, process lists, etc) can be found in standard OID slots from standard MIBs. For example, memory statistics can be found through the *UCD-SNMP-MIB* in OIDs such as `memTotalSwap.0`, `memAvailSwap.0`, `memTotalReal.0`, `memAvailReal.0+, etc., which translate to numeric OIDs` .1.3.6.1.4.1.2021.4.*`.In fact,` UCD-SNMP-MIB++ is the most useful MIB for overall system health checks.

Additionally, there's a list of specially monitored processes, also found through the `UCD-SNMP-MIB.UCD-SNMP-MIB::prNames` (`.1.3.6.1.4.1.2021.2.1.2`) gives the list of monitored processes, `prCount` (`.1.3.6.1.4.1.2021.2.1.5`) is how many of each process are running and `prErrorFlag` (`.1.3.6.1.4.1.2021.2.1.100`) gives a 0/1 error indication (with `prErrMessage` (`.1.3.6.1.4.1.2021.2.1.101`) providing an explanation of any error).

Tip

Some of these processes are not supposed to be running on the standby node, so you'll see the error flag raised there. A possible solution is to run these SNMP checks against the shared service IP of the cluster. Furthermore, `UCD-SNMP-MIB` provides a list of custom, external checks. The names of these can be found under the `UCD-SNMP-MIB::extNames` (`.2`) tree, with `extOutput` (`.101`) providing the output (one line) from each check and `extResult` (`.100`) the exit code from each check.

The first of these external checks called `collective_check` provides a combined and overall system health status indicator. It gathers information from both nodes and returns *0* in `extResult.1` (`.100.1`) if everything is OK and running as it should. If it finds a problem somewhere, but with the system still operational (e.g. a service is stopped on the inactive node), `extResult.1` will return *1* and `extOutput.1` will be set to a string that can be used to diagnose the problem. In case the system is found in a critical and non-operational state, `extResult.1` will return *2*, again with an error message set. If you want to keep it really simple, you can just monitor this one OID and raise an alarm if it ever goes to non-zero.

Tip

The *0/1/2* status codes allow for easy integration with *Nagios*.

The remaining external checks simply return statistics about the system, they all return a number in `extOutput` and have `extResult` always set to zero.

The full list of such checks is below. All of these checks exist in three flavors: the first returns the statistics from `sp1` (the first node in the sip:carrier pair), the second from `sp2`, and the third from whichever node is being queried (which is useful when querying the shared service IP). For example, the local SIP response time from `sp1` is in `sip_check_sp1`, from `sp2` is in `sip_check_sp2` and from the host itself in `sip_check_self`.

The base OID of the Result and Output OID is always `.1.3.6.1.4.1.2021.8.1`, so if you read `.100.1`, the full OID is `.1.3.6.1.4.1.2021.8.1.100.1`.

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.1	.100.1	.101.1	collective_check	Summarized platform check
UCD-SNMP-MIB::extNames.2	.100.2	.101.2	sip_check_sp1	SIP response time in seconds on sp1
UCD-SNMP-MIB::extNames.3	.100.3	.101.3	sip_check_sp2	SIP response time in seconds on sp2
UCD-SNMP-MIB::extNames.4	.100.4	.101.4	mysql_check_sp1	Average number of MySQL queries per second on sp1
UCD-SNMP-MIB::extNames.5	.100.5	.101.5	mysql_check_sp2	Average number of MySQL queries per second on sp2
UCD-SNMP-MIB::extNames.6	.100.6	.101.6	mysql_replication_check_sp1	MySQL replication delay in seconds on sp1
UCD-SNMP-MIB::extNames.7	.100.7	.101.7	mysql_replication_check_sp2	MySQL replication delay in seconds on sp2
UCD-SNMP-MIB::extNames.8	.100.8	.101.8	mpt_check_sp1	RAID status on sp1
UCD-SNMP-MIB::extNames.9	.100.9	.101.9	mpt_check_sp2	RAID status on sp2
UCD-SNMP-MIB::extNames.10	.100.10	.101.10	exim_queue_check_sp1	Number of mails undelivered in MTA queue on sp1
UCD-SNMP-MIB::extNames.11	.100.11	.101.11	exim_queue_check_sp2	Number of mails undelivered in MTA queue on sp2
UCD-SNMP-MIB::extNames.12	.100.12	.101.12	provisioned_subscribers_check_sp1	Number of subscribers provisioned on sp1

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.13	.100.13	.101.13	provisioned_subscribers_check_sp2	Number of subscribers provisioned on sp2
UCD-SNMP-MIB::extNames.14	.100.14	.101.14	kam_dialog_active_check_sp1	Number of active calls on sp1
UCD-SNMP-MIB::extNames.15	.100.15	.101.15	kam_dialog_active_check_sp2	Number of active calls on sp2
UCD-SNMP-MIB::extNames.16	.100.16	.101.16	kam_dialog_early_check_sp1	Number of calls in Early Media state on sp1
UCD-SNMP-MIB::extNames.17	.100.17	.101.17	kam_dialog_early_check_sp2	Number of calls in Early Media state on sp2
UCD-SNMP-MIB::extNames.18	.100.18	.101.18	kam_dialog_type_local_check_sp1	Number of active calls local on sp1
UCD-SNMP-MIB::extNames.19	.100.19	.101.19	kam_dialog_type_local_check_sp2	Number of active calls local on sp2
UCD-SNMP-MIB::extNames.20	.100.20	.101.20	kam_dialog_type_relay_check_sp1	Number of active calls routed via peers on sp1
UCD-SNMP-MIB::extNames.21	.100.21	.101.21	kam_dialog_type_relay_check_sp2	Number of active calls routed via peers on sp2
UCD-SNMP-MIB::extNames.22	.100.22	.101.22	kam_dialog_type_incoming_check_sp1	Number of incoming calls on sp1
UCD-SNMP-MIB::extNames.23	.100.23	.101.23	kam_dialog_type_incoming_check_sp2	Number of incoming calls on sp2
UCD-SNMP-MIB::extNames.24	.100.24	.101.24	kam_dialog_type_outgoing_check_sp1	Number of outgoing calls on sp1
UCD-SNMP-MIB::extNames.25	.100.25	.101.25	kam_dialog_type_outgoing_check_sp2	Number of outgoing calls on sp2
UCD-SNMP-MIB::extNames.26	.100.26	.101.26	kam_usrloc_regusers_check_sp1	Number of subscribers with at least one active registration on sp1
UCD-SNMP-MIB::extNames.27	.100.27	.101.27	kam_usrloc_regusers_check_sp2	Number of subscribers with at least one active registration on sp2
UCD-SNMP-MIB::extNames.28	.100.28	.101.28	kam_usrloc_regdevices_check_sp1	Total number of registered end devices on sp1

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.29	.100.29	.101.29	kam_usrloc_regdevices_check_sp2	Total number of registered end devices on sp2
UCD-SNMP-MIB::extNames.30	.100.30	.101.30	mysql_replication_discrepancies_check_sp1	Number of MySQL tables not in sync between sp1 and sp2
UCD-SNMP-MIB::extNames.31	.100.31	.101.31	mysql_replication_discrepancies_check_sp2	Number of MySQL tables not in sync between sp1 and sp2
UCD-SNMP-MIB::extNames.32	.100.32	.101.32	sip_check_self	Summarized platform check on active node
UCD-SNMP-MIB::extNames.33	.100.33	.101.33	mysql_check_self	Average number of MySQL queries per second on active node
UCD-SNMP-MIB::extNames.34	.100.34	.101.34	mysql_replication_check_self	MySQL replication delay in seconds on active node
UCD-SNMP-MIB::extNames.35	.100.35	.101.35	mpt_check_self	RAID status on active node
UCD-SNMP-MIB::extNames.36	.100.36	.101.36	exim_queue_check_self	Number of mails undelivered in MTA queue on active node
UCD-SNMP-MIB::extNames.37	.100.37	.101.37	provisioned_subscribers_check_self	Number of subscribers provisioned on active node
UCD-SNMP-MIB::extNames.38	.100.38	.101.38	kam_dialog_active_check_self	Number of active calls on active node
UCD-SNMP-MIB::extNames.39	.100.39	.101.39	kam_dialog_early_check_self	Number of calls in Early Media state on active node
UCD-SNMP-MIB::extNames.40	.100.40	.101.40	kam_dialog_type_local_check_self	Number of active calls local on active node
UCD-SNMP-MIB::extNames.41	.100.41	.101.41	kam_dialog_type_relay_check_self	Number of active calls routed via peers on active node
UCD-SNMP-MIB::extNames.42	.100.42	.101.42	kam_dialog_type_incoming_check_self	Number of incoming calls on active node
UCD-SNMP-MIB::extNames.43	.100.43	.101.43	kam_dialog_type_outgoing_check_self	Number of outgoing calls on active node

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.44	.100.44	.101.44	kam_usrloc_regusers_check_self	Number of subscribers with at least one active registration on active node
UCD-SNMP-MIB::extNames.45	.100.45	.101.45	kam_usrloc_regdevices_check_self	Total number of registered end devices on active node
UCD-SNMP-MIB::extNames.46	.100.46	.101.46	mysql_replication_discrepancies_check_self	Number of MySQL tables not in sync between sp1 and sp2

Tip

Some of the checks can be disabled (and some are disabled by default) through the `config.yml` file, and those checks will then return an error message or an empty string in their `extOutput`. Enable those checks in the config file to get their output in the SNMP OID tree. The enable/disable flags can be found in the `checktools` section.

A Cloud PBX

The sip:carrier comes with a commercial Cloud PBX module to provide B2B features for small and medium sized enterprises. The following chapter describes the configuration of the PBX features.

A.1 Configuring the Device Management

The *Device Management* is used by admins and resellers to define the list of device models, firmwares and configurations available for end customer usage. These settings are pre-configured for the default reseller up-front by Sipwise and have to be set up for every reseller separately, so a reseller can choose the devices he'd like to serve and potentially tweak the configuration for them.

End customers choose from a list of *Device Profiles*, which are defined by a specific *Device Model*, a list of *Device Firmwares* and a *Device Configuration*. The following sub-chapters describe the setup of these components.

To do so, go to *Settings→Device Management*.

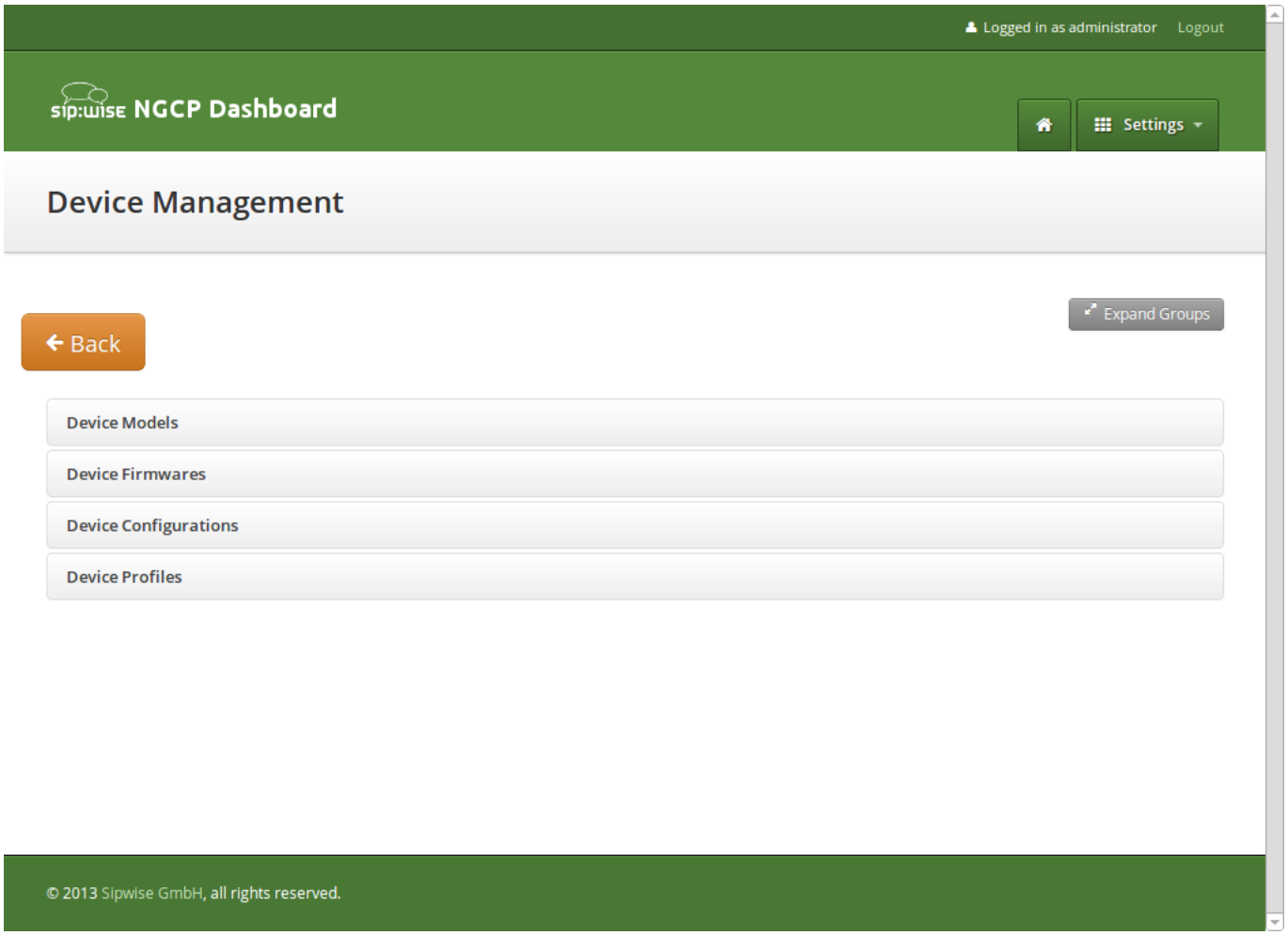


Figure 12: Device Management

A.1.1 Setting up Device Models

A *Device Model* defines a specific hardware device, like the vendor, model name, the number of keys and their capabilities. For example a Cisco SPA504G has 4 keys, which can be used for private lines, shared lines (SLA) and busy lamp field (BLF). If you have an additional attendant console, you get 32 more buttons, which can only do BLF.

In this example, we will create a Cisco SPA504G with an additional Attendant Console.

Expand the *Device Models* row and click *Create Device Model*.

First, you have to select the reseller this device model belongs to, and define the vendor and model name.

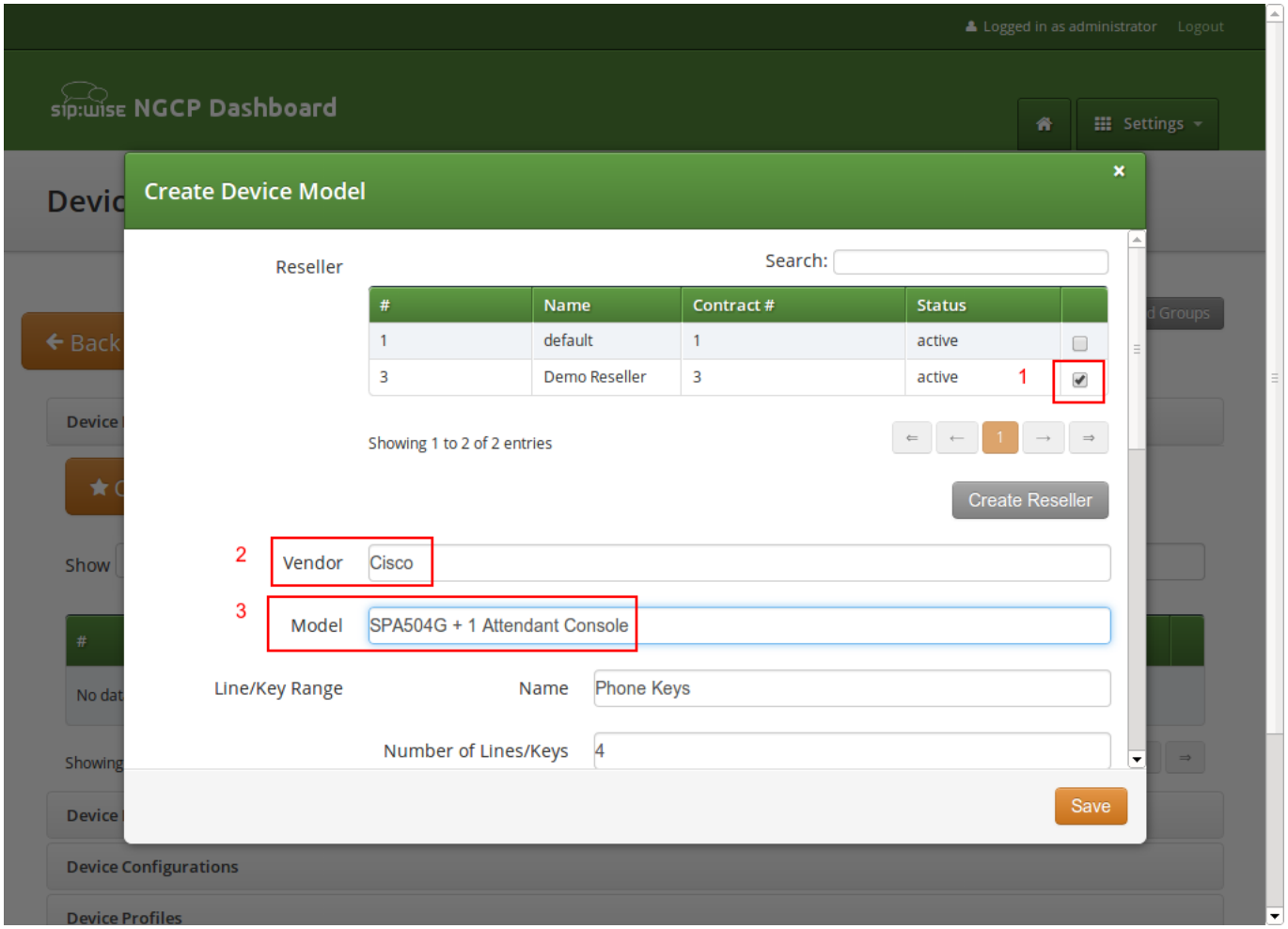


Figure 13: Create Device Model Part 1

In the *Line/Key Range* section, you can define the first set of keys, which we will label `Phone Keys`. The name is important, because it is referenced in the configuration file template, which we will look into in the next sections. The SPA504G internal phone keys support private lines (where the customer can assign a normal subscriber, which is used to place and receive standard phone calls), shared lines (where the customer can assign a subscriber which is shared across multiple people) and busy lamp field (where the customer can assign other subscribers to be monitored when they get a call, and which also acts as speed dial button to the subscriber assigned for BLF), so we enable all 3 of them.

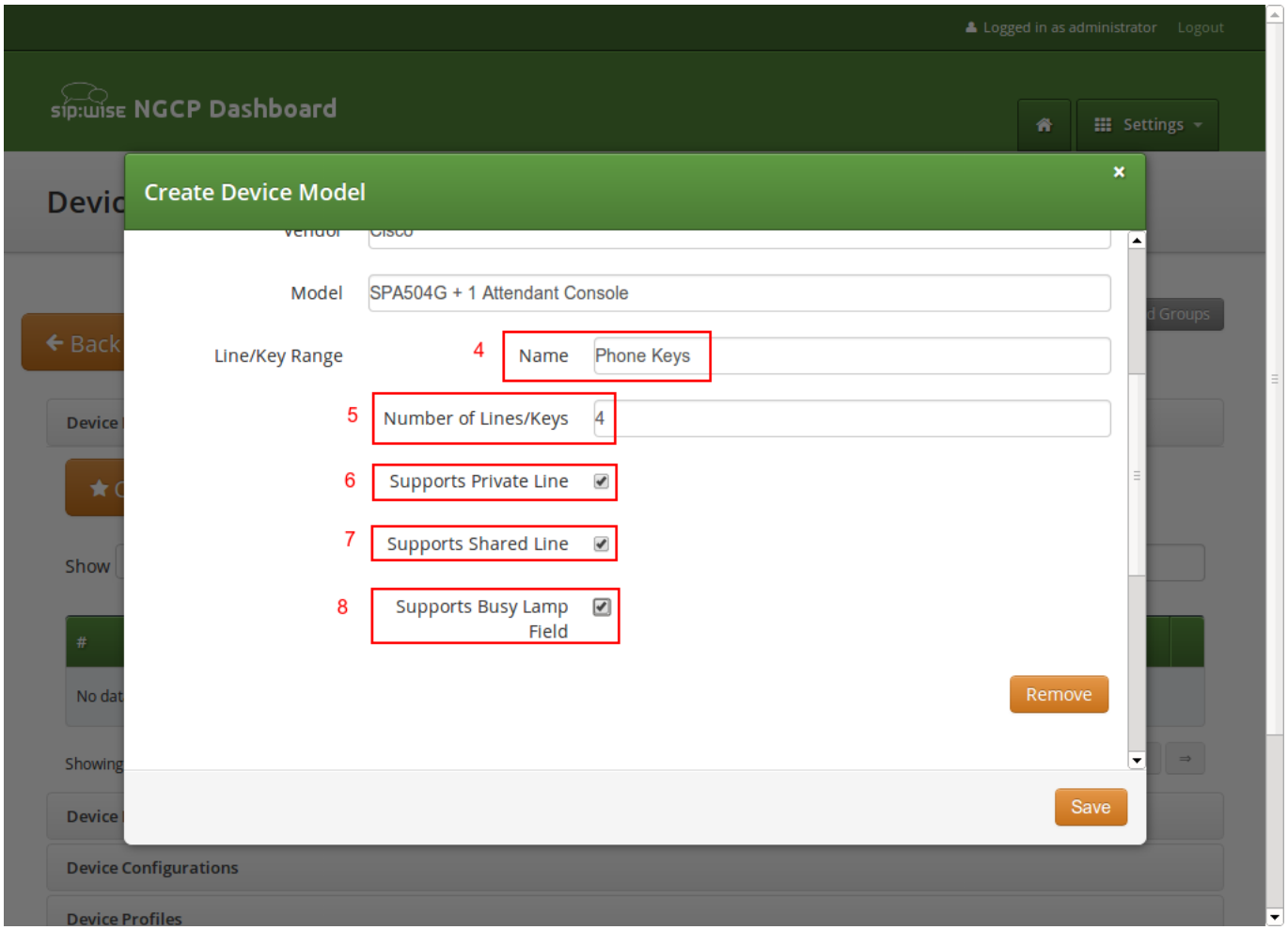


Figure 14: Create Device Model Part 2

In order to also configure the attendant console, press the *Add another Line/Key Range* button to specify the attendant console keys.

Again provide a name for this range, which will be `Attendant Console 1` to match our configuration defined later. There are 32 buttons on the attendant console, so set the number accordingly. Those 32 buttons only support BLF, so make sure to **uncheck** the private and shared line options, and only check the `busy lamp field` option.

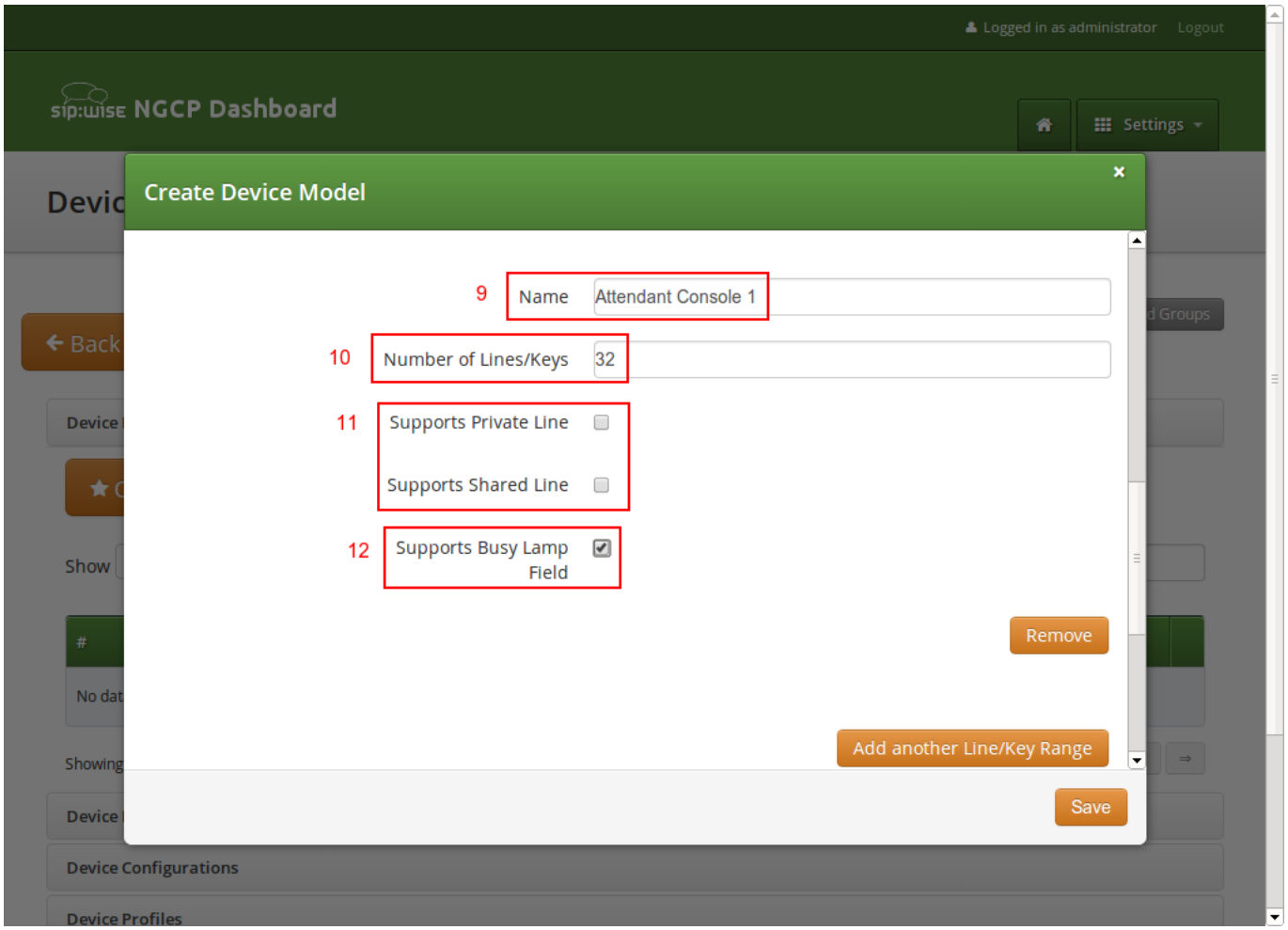


Figure 15: Create Device Model Part 3

The last two settings to configure are the *Front Image* and *MAC Address Image* fields. Upload a picture of the phone here in the first field, which is shown to the customer for him to recognize easily how the phone looks like. The MAC image is used to tell the customer where he can read the MAC address from. This could be a picture of the back of the phone with the label where the MAC is printed, or an instruction image how to get the MAC from the phone menu.

The rest of the fields are left at their default values, which are set to work with Cisco SPAs. Their meaning is as follows:

- *Bootstrap Sync URI*: If a stock phone is plugged in for the first time, it needs to be provisioned somehow to let it know where to fetch its configuration file from. Since the stock phone doesn't know about your server, you have to define an HTTP URI here, where the customer is connected with his web browser to set the according field.
- *Bootstrap Sync HTTP Method*: This setting defines whether an HTTP GET or POST is sent to the Sync URI.
- *Bootstrap Sync Params*: This setting defines the parameters appended to the Sync URI in case of a GET, or posted in the request body in case of POST, when the customer presses the *Sync* button later on.

Finally press *Save* to create the new device model.

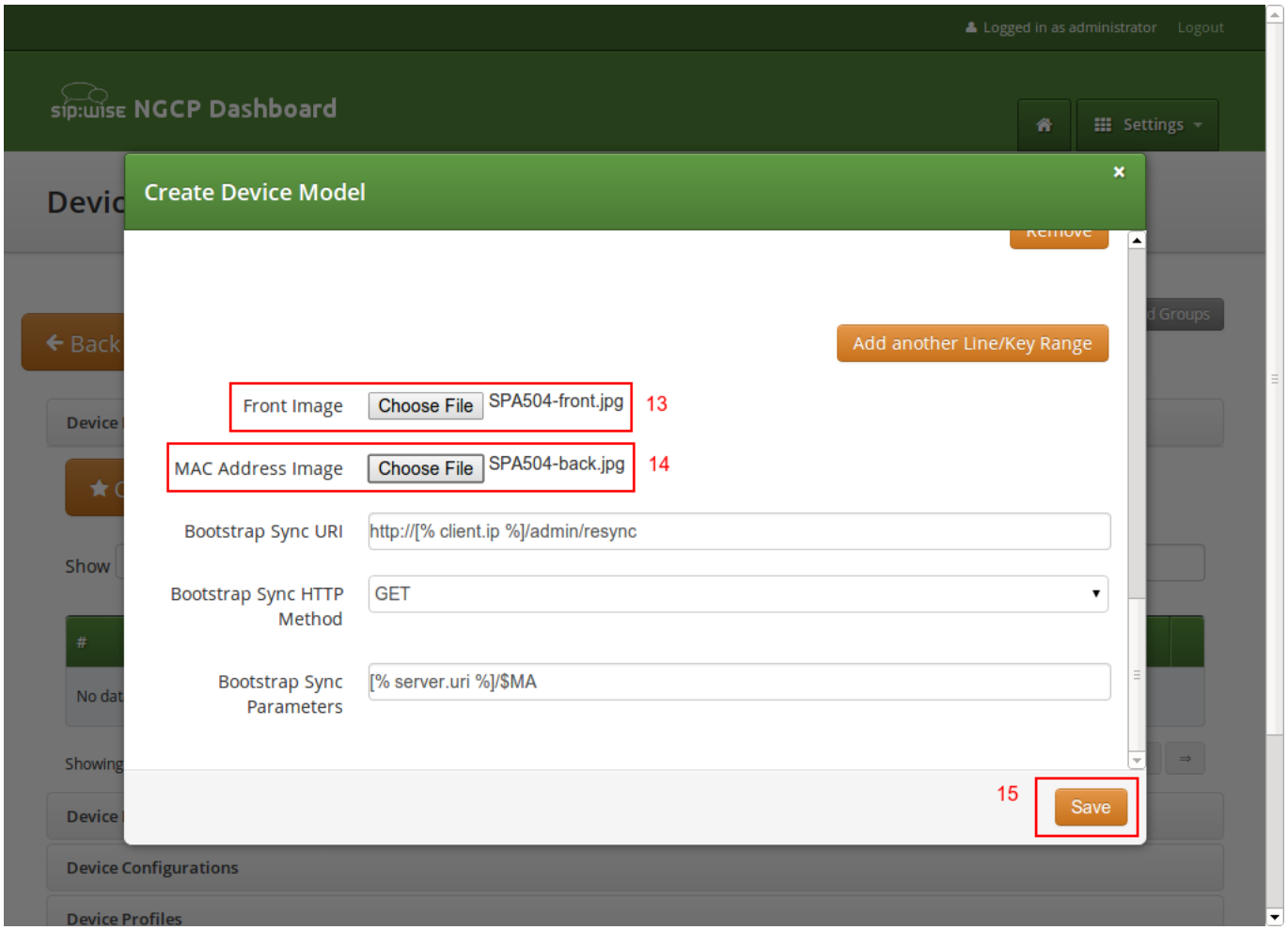


Figure 16: Create Device Model Part 4

A.1.2 Uploading Device Firmwares

A device model can optionally have one or more device firmware(s). Some devices like the Cisco SPA series don't support direct firmware updates from an arbitrary to the latest one, but need to go over specific firmware steps. In the device configuration discussed next, you can return the *next* supported firmware version, if the phone passes the current version in the firmware URL.

Since a stock phone purchased from any shop can have an arbitrary firmware version, we need to upload all firmwares needed to get from any old one to the latest one. In case of the Cisco SPA3x/SPA5x series, that would be the following versions, if the phone starts off with version 7.4.x:

- spa50x-30x-7-5-1a.bin
- spa50x-30x-7-5-2b.bin
- spa50x-30x-7-5-5.bin

So to get an SPA504G with a firmware version 7.4.x to the latest version 7.5.5, we need to upload each firmware file as follows.

Open the *Device Firmware* row in the *Device Management* section and press *Upload Device Firmware*.

Select the device model we're going to upload the firmware for, then specify the firmware version and choose the firmware file, then press *Save*.

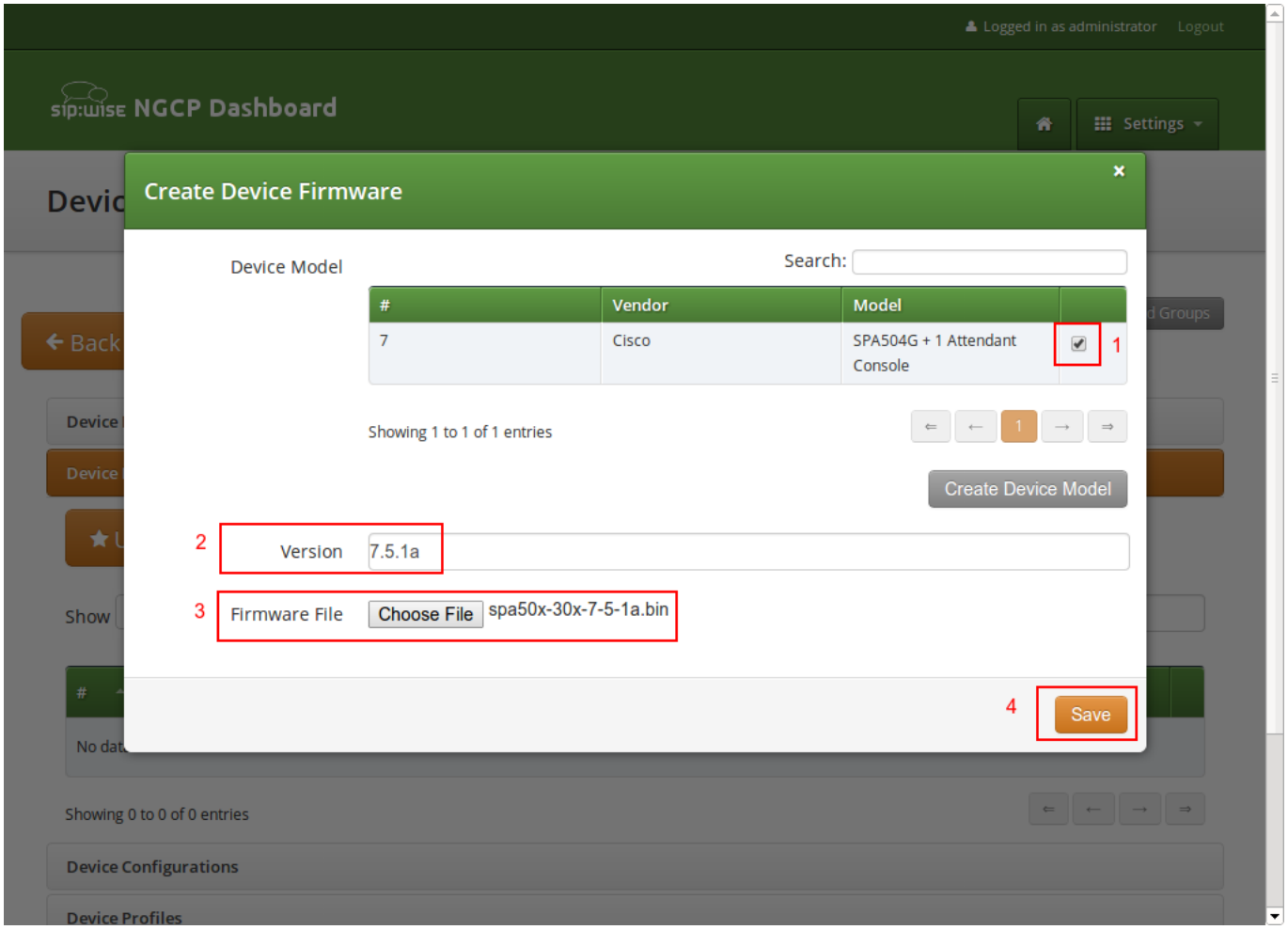


Figure 17: Upload Device Firmware

Repeat this step for every firmware in the list above (and any new firmware you want to support when it's available).

A.1.3 Creating Device Configurations

Each customer device needs a configuration file, which defines the URL to perform firmware updates, and most importantly, which defines the subscribers and features configured on each of the lines and keys. Since these settings are different for each physical phone at all the customers, the Cloud PBX module provides a template system to specify the configurations. That way, template variables can be used in the generic configuration, which are filled in by the system individually when a physical device fetches its configuration file.

To upload a configuration template, open the *Device Configuration* row and press *Create Device Configuration*.

Select the device model and specify a version number for this configuration (it is only for your reference to keep track of different

versions). For Cisco SPA phones, keep the *Content Type* field to `text/xml`, since the configuration content will be served to the phone as XML file.

For devices other than the Cisco SPA, you might set `text/plain` if the configuration file is plain text, or `application/octet-stream` if the configuration is compiled into some binary form.

Finally paste the configuration template into the *Content* area and press *Save*.

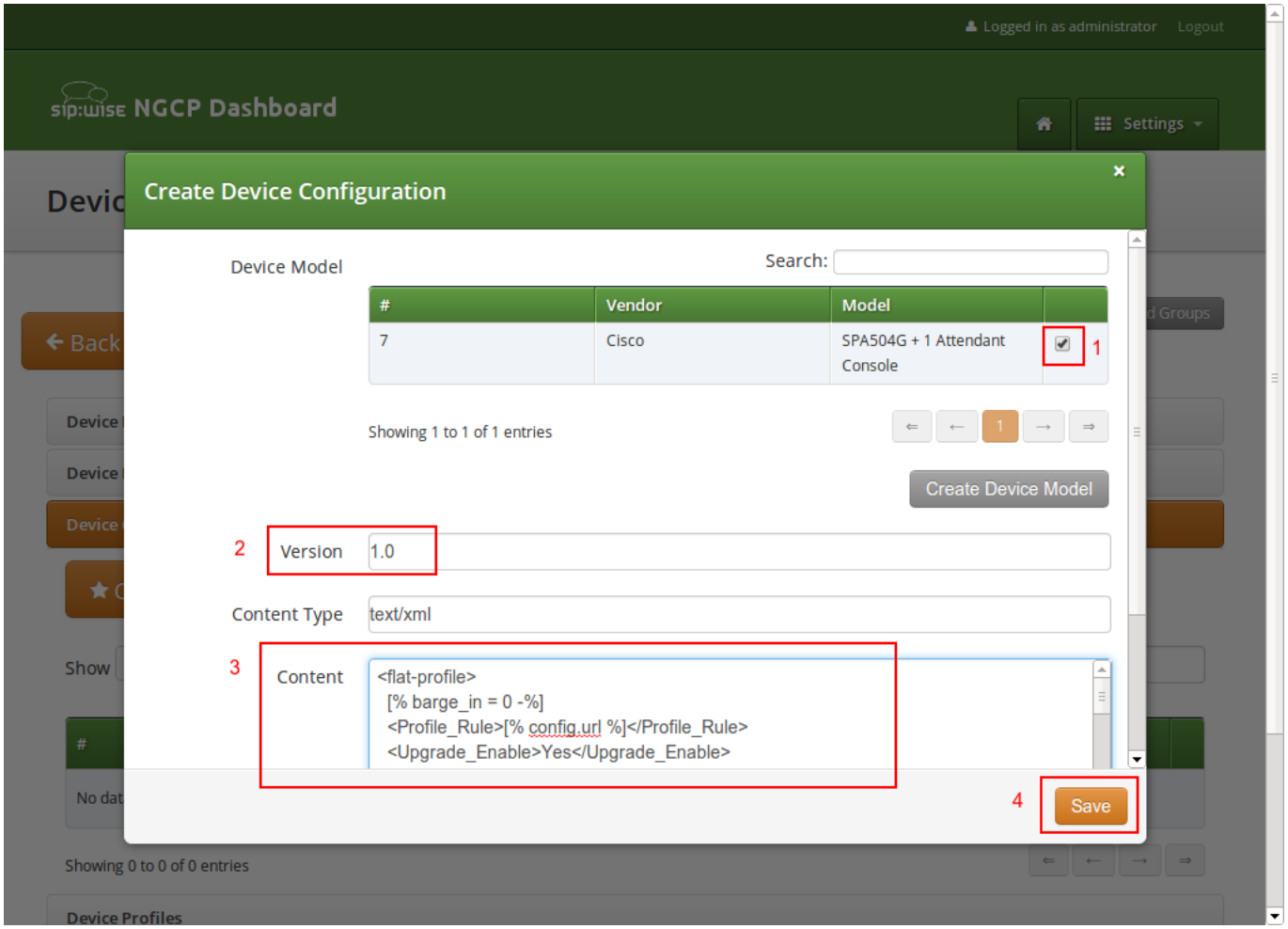


Figure 18: Upload Device Configuration

The templates for certified device models are provided by Sipwise, but you can also write your own. The following variables can be used in the template:

- `config.url`: The URL to the config file, including the device identifier (e.g. `http://sip.example.org:1444/device/autoprov/config/001122334455`).
- `firmware.maxversion`: The latest firmware version available on the system for the specific device.
- `firmware.baseurl`: The base URL to download firmwares (e.g. `http://sip.example.org:1444/device/autoprov/firmware`). To fetch the next newer firmware for a Cisco SPA, you can use the template line `[% firmware.baseurl %]/$MA/from/$SWVER/next`.

- `phone.stationname`: The name of the station (physical device) the customer specifies for this phone. Can be used to show on the display of the phone.
- `phone.lineranges`: An array of lines/keys as specified for the device model. Each entry in the array has the following keys:
  - `name`: The name of the line/key range as specified in the *Device Model* section (e.g. `Phone Keys`).
  - `num_lines`: The number of lines/keys in the line range (e.g. `4` in our `Phone Keys` example, or `32` in our `Attendant Console 1` example).
  - `lines`: An array of lines (e.g. subscriber definitions) for this line range. Each entry in the array has the following keys:
    - `keynum`: The index of the key in the line range, starting from 0 (e.g. `keynum` will be 3 for the 4th key of our `Phone Keys` range).
    - `rangenum`: The index of the line range, starting from 0. The order of line ranges is as you have specified them (e.g. `Phone Keys` was specified first, so it gets rangenum 0, `Auto Attendant 1` gets rangenum 1).
    - `type`: The type of the line/key, on of `private`, `shared` or `blf`.
    - `username`: The SIP username of the line.
    - `domain`: The SIP domain of the line.
    - `password`: The SIP password of the line.
    - `displayname`: The SIP Display Name of the line.

Within the configuration template itself, you can use any Template Toolkit directive and any own variables you like (just make sure to not override any of the ones specified above). For documentation on the syntax, please refer to the Template Toolkit Manual.

A.1.4 Creating Device Profiles

When the customer configures his own device, he doesn't select a *Device Model* directly, but a *Device Profile*. A device profile specifies which model is going to be used with which configuration version. This allows the operator to create new configuration files and assign them to a profile, while still keeping older configuration files for reference or roll-back scenarios. It also allows to test new firmwares by creating a test device model with the new firmware and a specific configuration, without impacting any existing customer devices.

To create a *Device Profile* for our phone, open the *Device Profile* row in the *Device Management* section and press *Create Device Profile*.

Select the device configuration (which implicitly identifies a device model) and specify a *Profile Name*. This name is what the customer sees when he is selecting a device he wants to provision, so pick a descriptive name which clearly identifies a device. Press *Save* to create the profile.

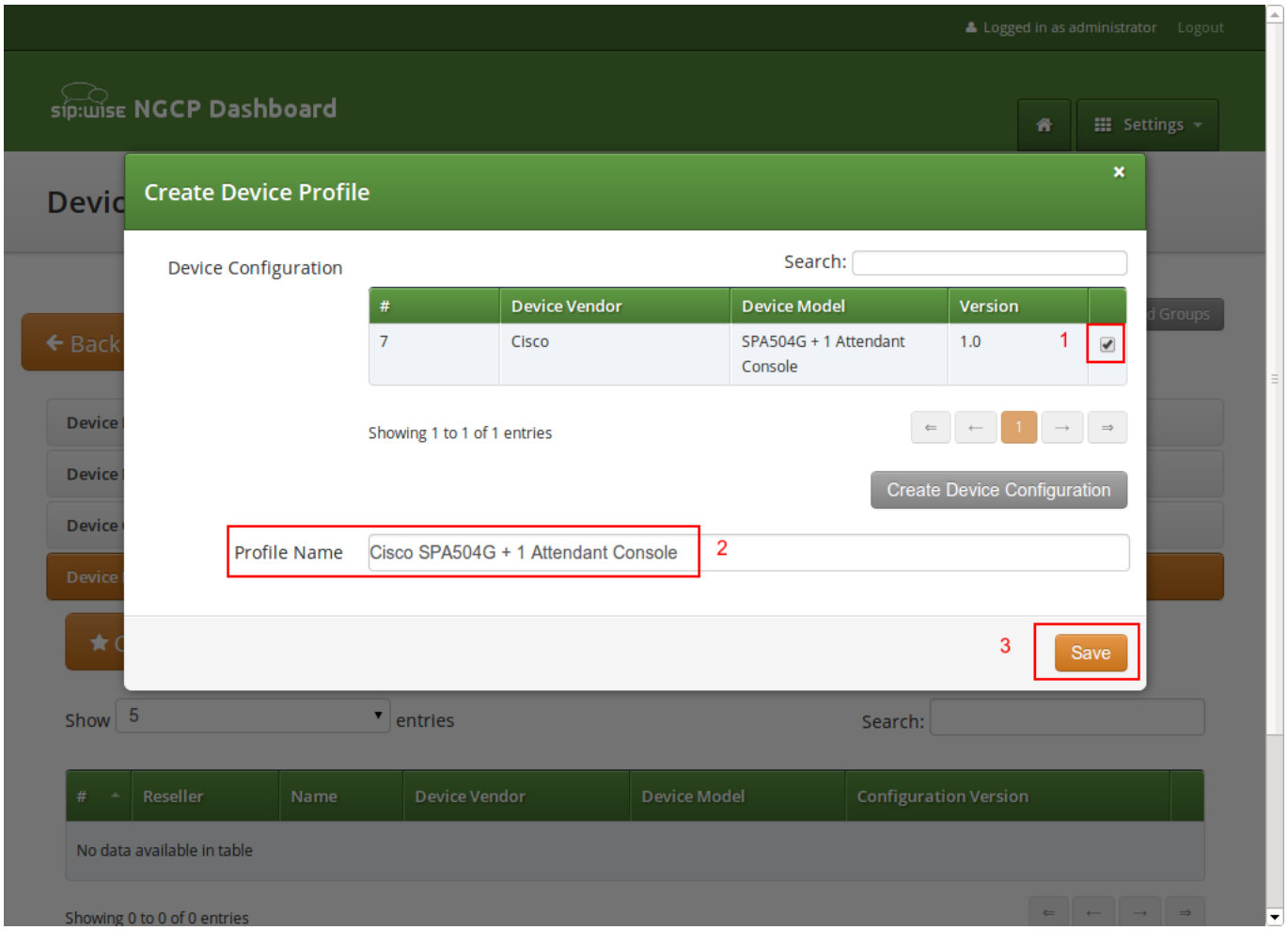


Figure 19: Create Device Profile

Repeat the steps as needed for every device you want to make available to customers.

A.2 Preparing PBX Rewrite Rules

In a PBX environment, the dial-plans usually looks different than for normal SIP subscribers. PBX subscribers should be able to directly dial internal extensions (e.g. 100) instead of the full number to reach another PBX subscriber in the same PBX segment. Therefore, we need to define specific *Rewrite Rules* to make this work.

The PBX dial plans are different from country to country. In the Central European area, you can directly dial an extension (e.g. `100`), and if you want to dial an international number like `0049 1 23456`, you have to dial a break-out digit first (e.g. `0`), so the number to be dialed is `0 0049 1 23456`. Other countries are used to other break-out codes (e.g. `9`), which then results in `9 0049 1 23456`. If you dial a national number like `01 23456`, then the number to actually be dialled is `9 01 23456`.

Since all numbers must be normalized to E.164 format via inbound rewrite rules, the rules need to be set up accordingly.

Let's assume that the break-out code for the example customers created below is `0`, so we have to create a *Rewrite Rule Set* with the following rules.

A.2.1 Inbound Rewrite Rules for Caller

- **Match Pattern**: `^([1-9][0-9]+)$`
- **Replacement Pattern**: `${caller_cloud_pbx_base_cli}\1`
- **Description**: `extension to e164`
- **Direction**: `Inbound`
- **Field**: `Caller`

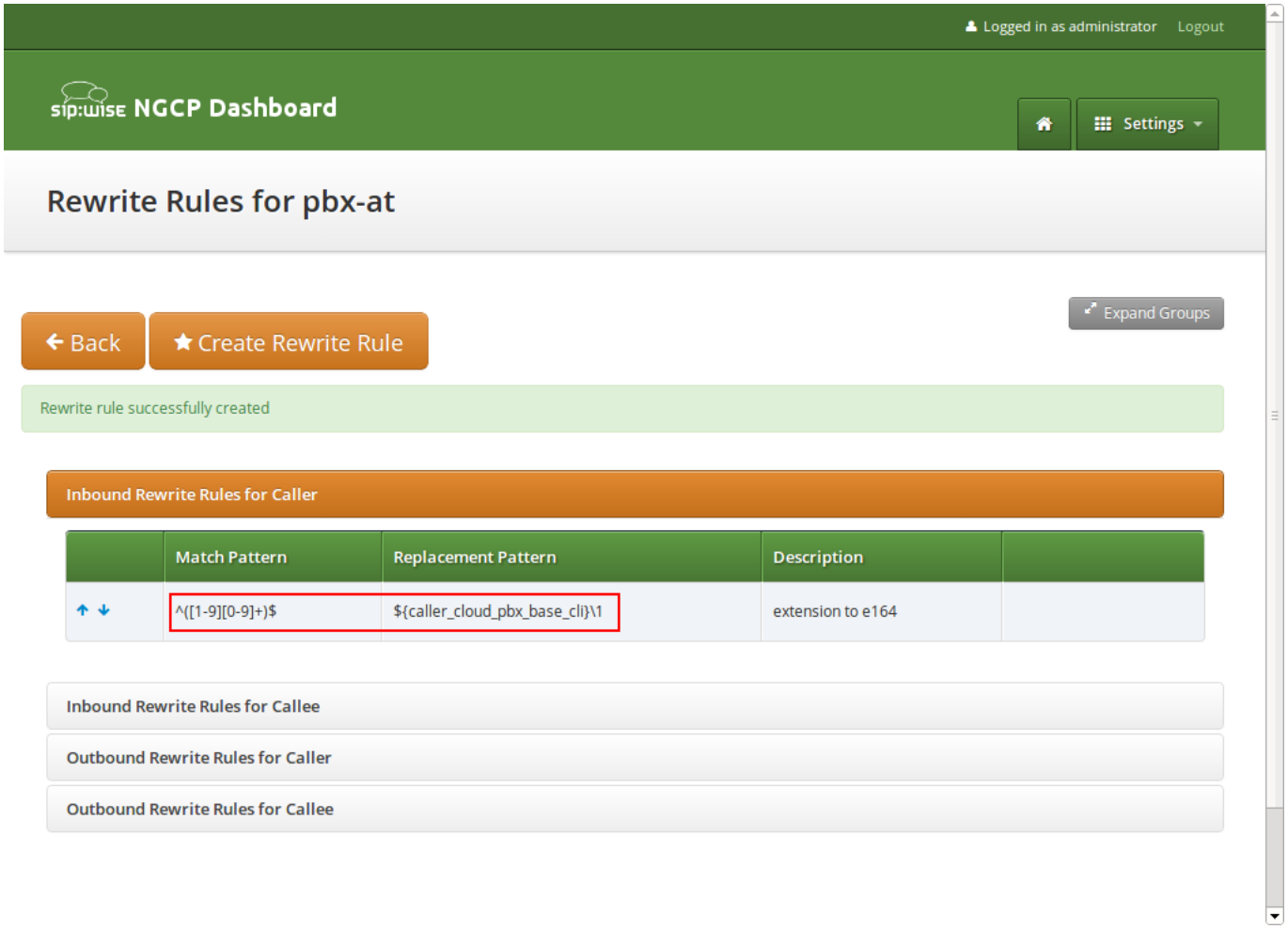


Figure 20: Inbound Rewrite Rule for Caller

A.2.2 Inbound Rewrite Rules for Callee

These rules are the most important ones, as they define which number formats the PBX subscribers can dial. For the break-out code of `0`, the following rules are necessary e.g. for German dialplans to allow pbx internal extension dialing, local area calls without area codes, national calls with area code, and international calls with country codes.

PBX INTERNAL EXTENSION DIALIN

- **Match Pattern**: `^([1-9][0-9]+)$`
- **Replacement Pattern**: `${caller_cloud_pbx_base_cli}\1`
- **Description**: `extension to e164`
- **Direction**: `Inbound`
- **Field**: `Callee`

LOCAL DIALING WITHOUT AREA CODE (USE BREAK-OUT CODE 0)

- **Match Pattern**: `^0([1-9][0-9]+)$`
- **Replacement Pattern**: `${caller_cc}${caller_ac}\1`
- **Description**: `local to e164`
- **Direction**: `Inbound`
- **Field**: `Callee`

NATIONAL DIALING (USE BREAK-OUT CODE 0 AND PREFIX AREA CODE BY 0)

- **Match Pattern**: `^00([1-9][0-9]+)$`
- **Replacement Pattern**: `${caller_cc}\1`
- **Description**: `national to e164`
- **Direction**: `Inbound`
- **Field**: `Callee`

INTERNATIONAL DIALING (USE BREAK-OUT CODE 0 AND PREFIX COUNTRY CODE BY 00)

- **Match Pattern**: `^000([1-9][0-9]+)$`
- **Replacement Pattern**: `\1`
- **Description**: `international to e164`
- **Direction**: `Inbound`
- **Field**: `Callee`

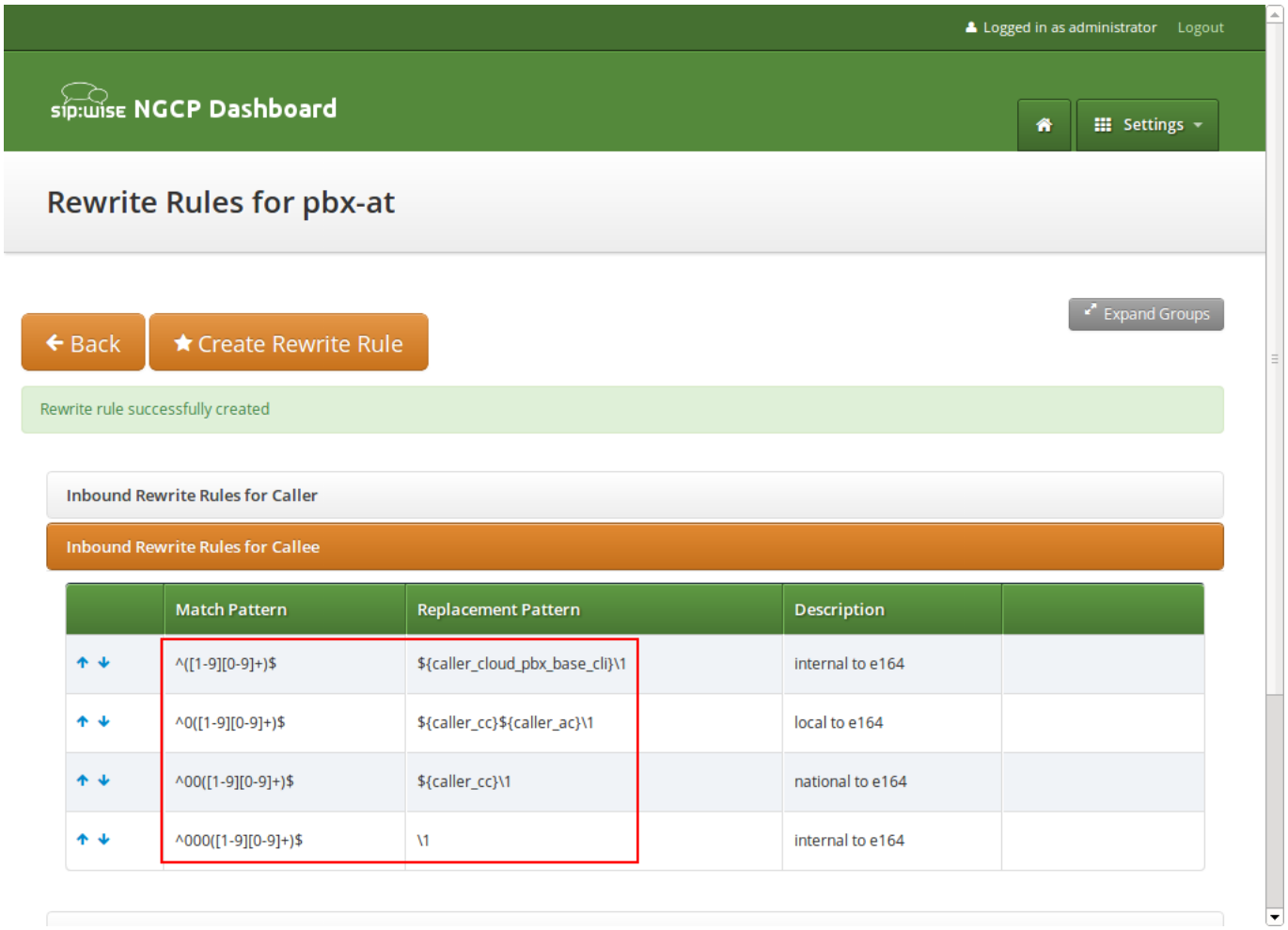


Figure 21: Inbound Rewrite Rule for Callee

A.2.3 Outbound Rewrite Rules for Caller

When a call goes to a PBX subscriber, it needs to be normalized in a way that it's call-back-able, which means that it needs to have the break-out code prefixed. We create a rule to show the calling number in international format including the break-out code. For PBX-internal calls, the caller name will be shown (this is handled by implicitly setting domain preferences accordingly, so you don't have to worry about that in rewrite rules).

- **Match Pattern**: `^([1-9][0-9]+)$`
- **Replacement Pattern**: `000\1`
- **Description**: `e164 to full international`
- **Direction**: `Outbound`
- **Field**: `Caller`

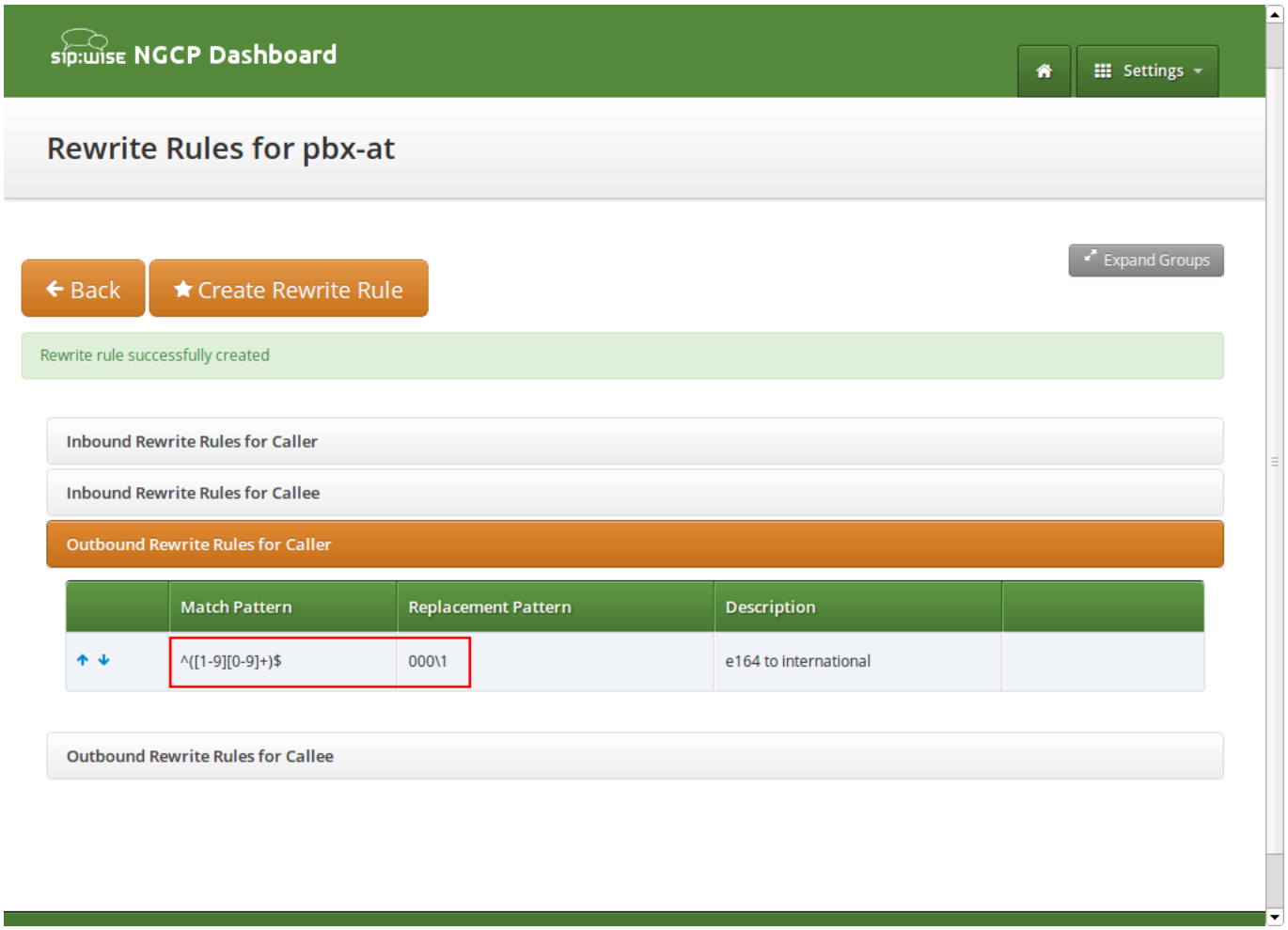


Figure 22: Outbound Rewrite Rule for Caller

Create a new *Rewrite Rule Set* for each dial plan you'd like to support. You can later assign it to customer domains and even to subscribers, if a specific subscriber of a PBX customer would like to have his own dial plan.

A.3 Creating Customers and Pilot Subscribers

As with a normal SIP Account, you have to create a *Customer* contract per customer, and one *Subscriber*, which the customer can use to log into the web interface and manage his PBX environment.

A.3.1 Creating a PBX Customer

Go to *Settings→Customers* and click *Create Customer*. We need a *Contact* for the customer, so press *Create Contact*.

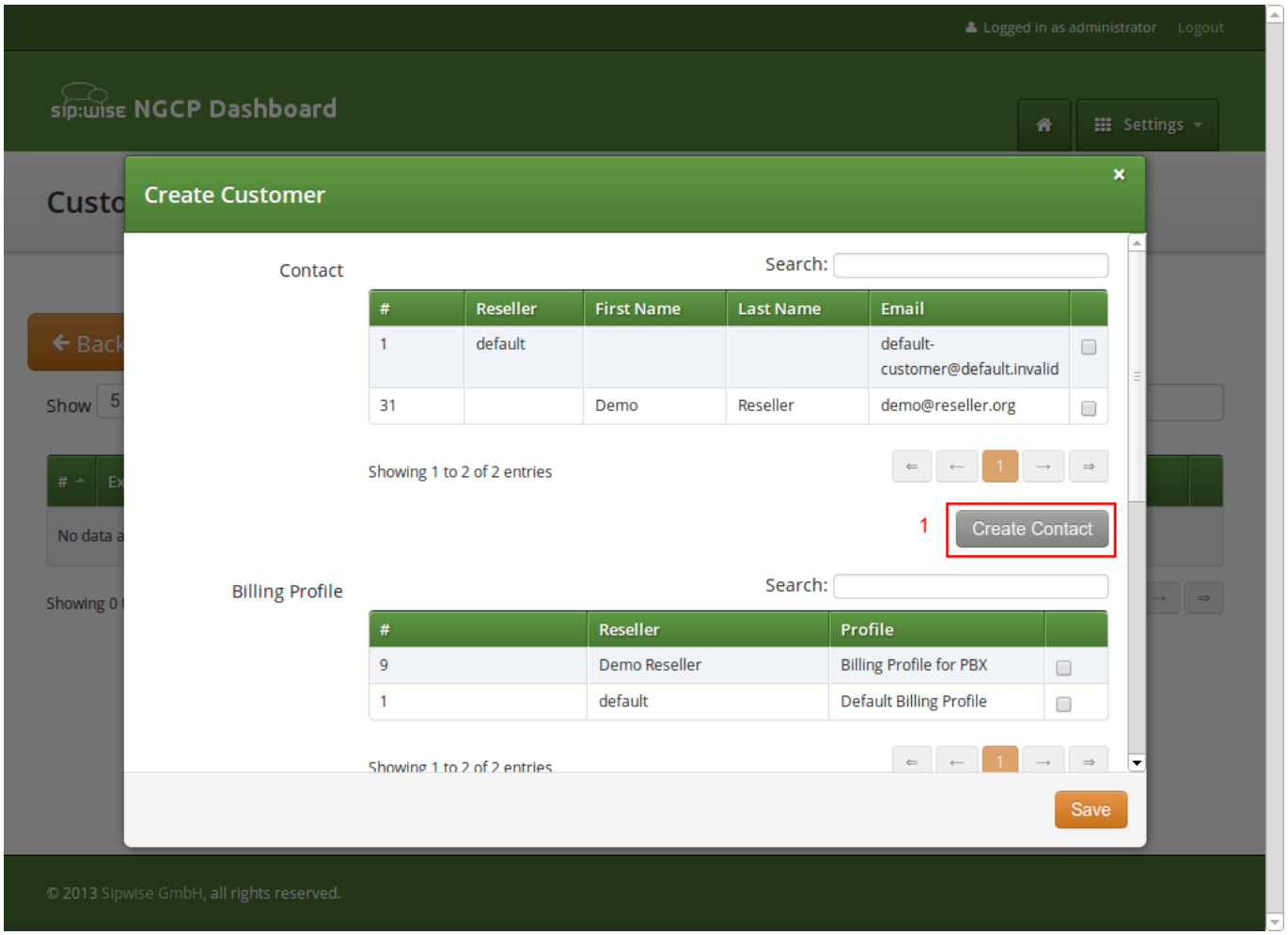


Figure 23: Create PBX Customer Part 1

Fill in the desired fields (you need to provide at least the *Email Address*) and press *Save*.

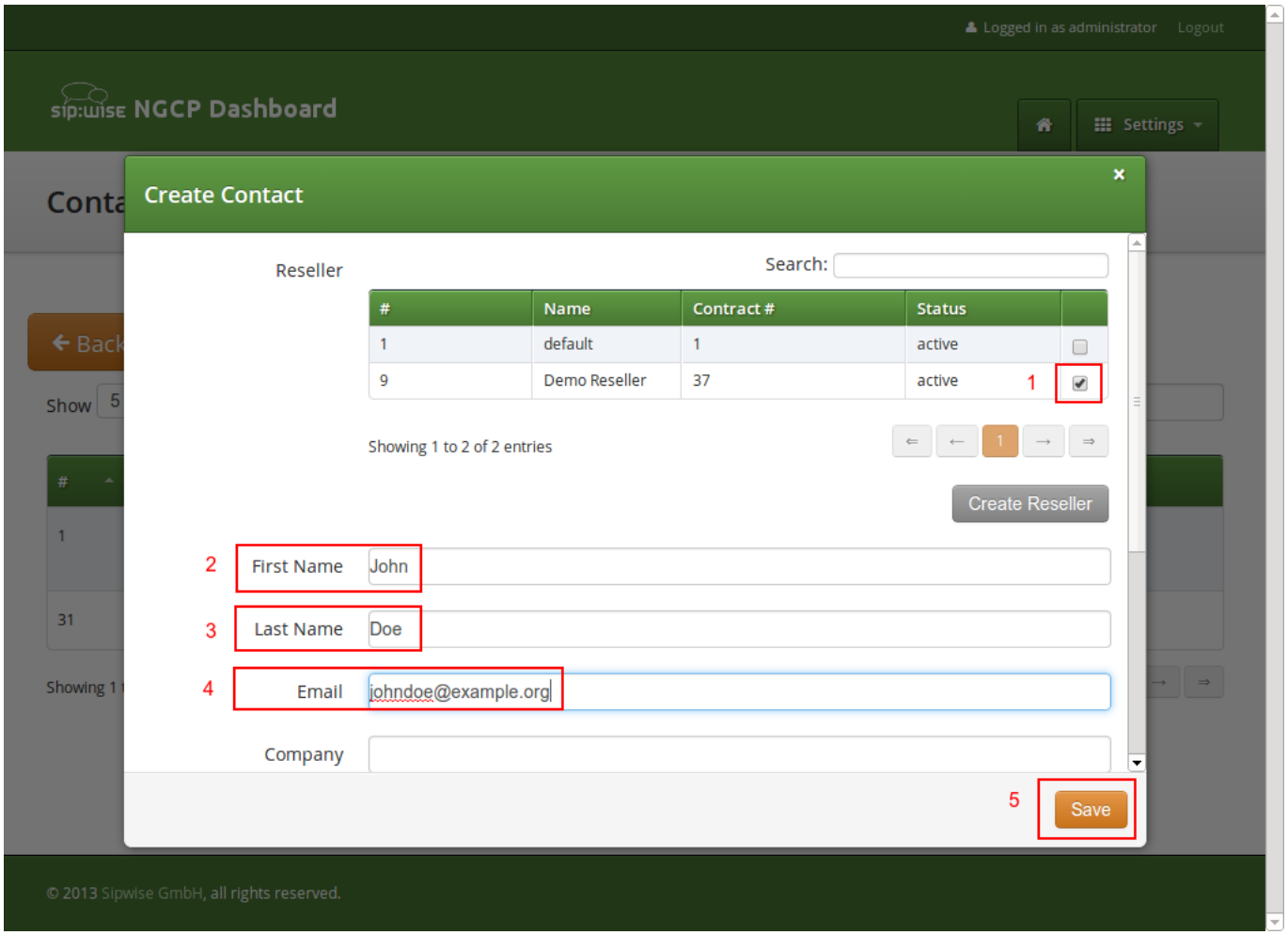


Figure 24: Create PBX Customer Contact

The new *Contact* will be automatically selected now. Also select a *Billing Profile* you want to use for this customer. If you don't have one defined yet, press *Create Billing Profile*, otherwise select the one you want to use.

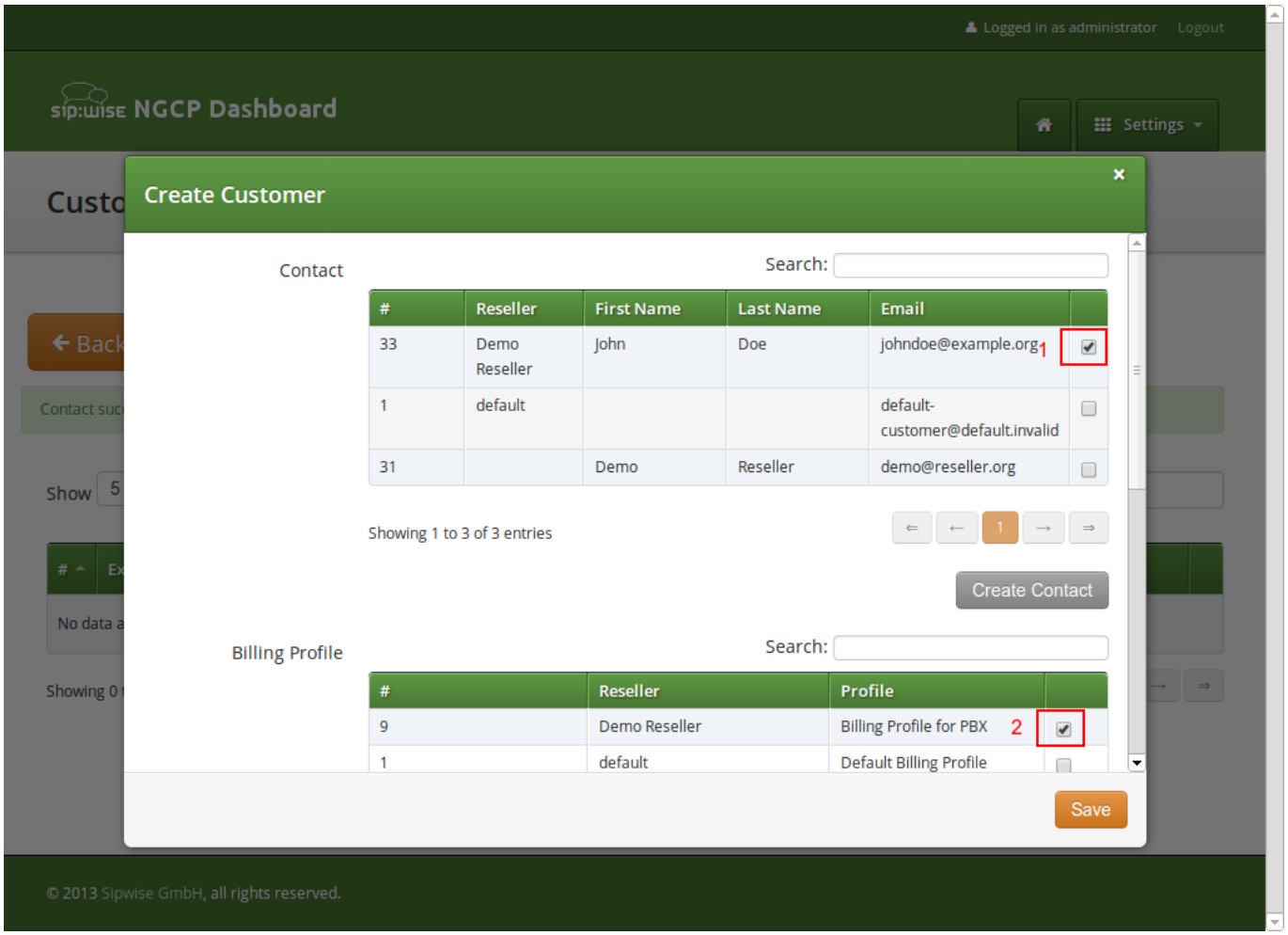


Figure 25: Create PBX Customer Part 2

Next, you need to select the *Product* for the PBX customer. Since it's going to be a PBX customer, select the product *Cloud PBX Account*.

Since PBX customers are supposed to manage their subscribers by themselves, they are able to create them via the web interface. To set an upper limit of subscribers a customer can create, define the value in the *Max Subscribers* field.

Important

As you will see later, both PBX subscribers and PBX groups are normal subscribers, so the value defined here limits the overall amount of subscribers **and** groups. A customer can create an unlimited amount of subscribers if you leave this field empty.

Press *Save* to create the customer.

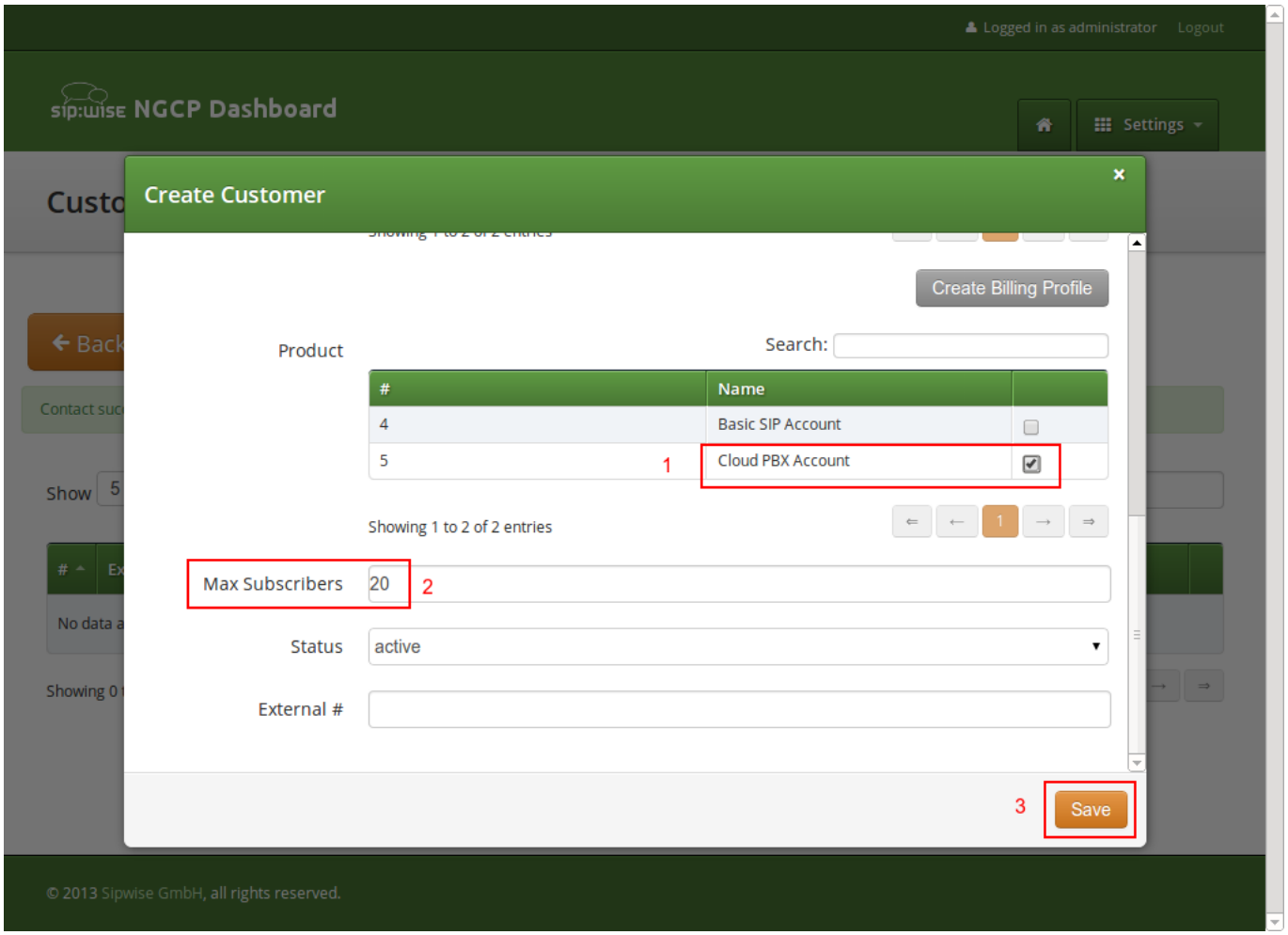


Figure 26: Create PBX Customer Part 3

A.3.2 Creating a PBX Pilot Subscriber

Once the customer is created, you need to create at least one *Subscriber* for the customer, so he can log into the web interface and manage the rest by himself.

Click the *Details* button on the newly created customer to enter the detailed view.

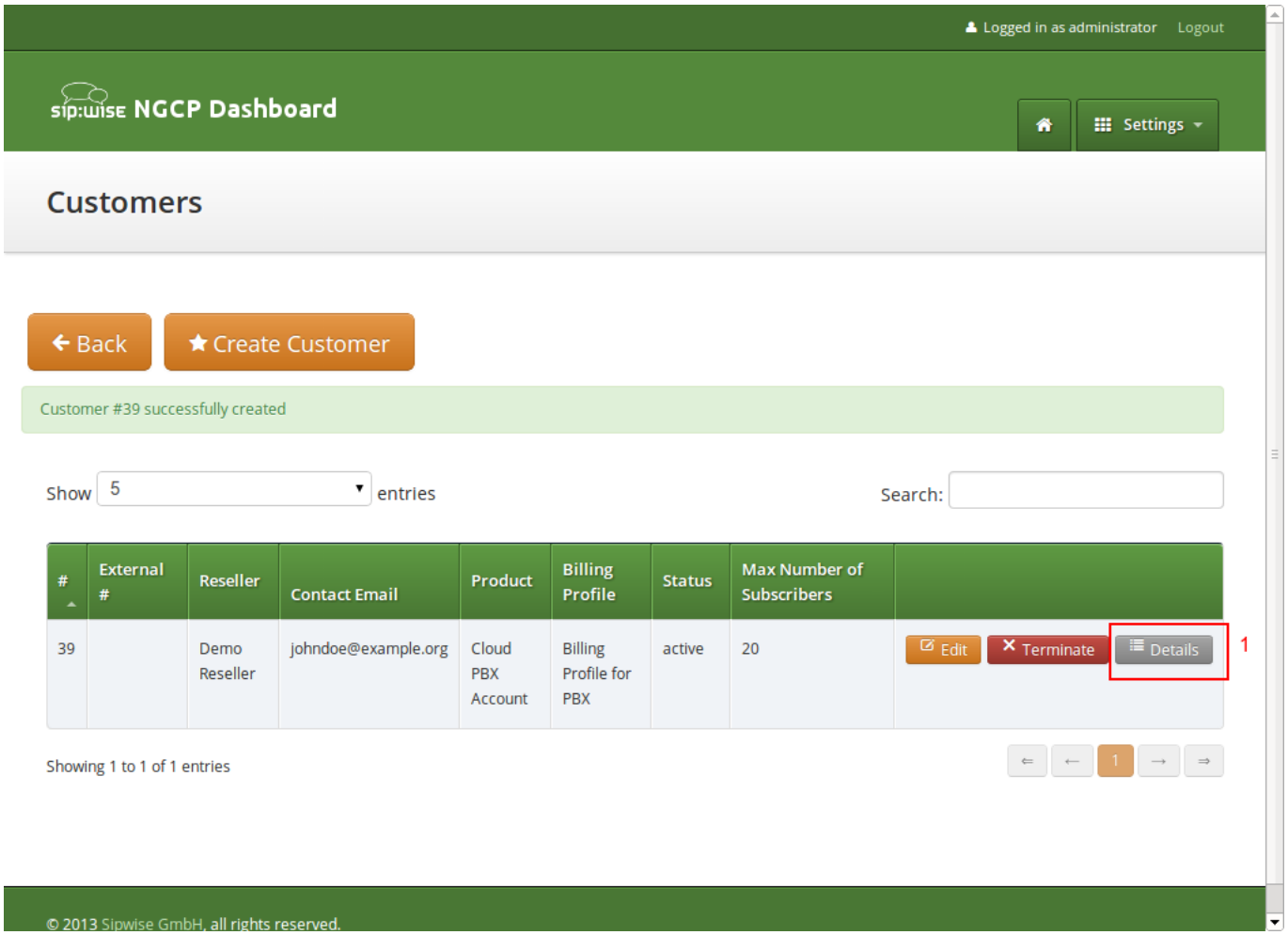


Figure 27: Go to Customer Details

To create the subscriber, open the *Subscribers* row and click *Create Subscriber*.

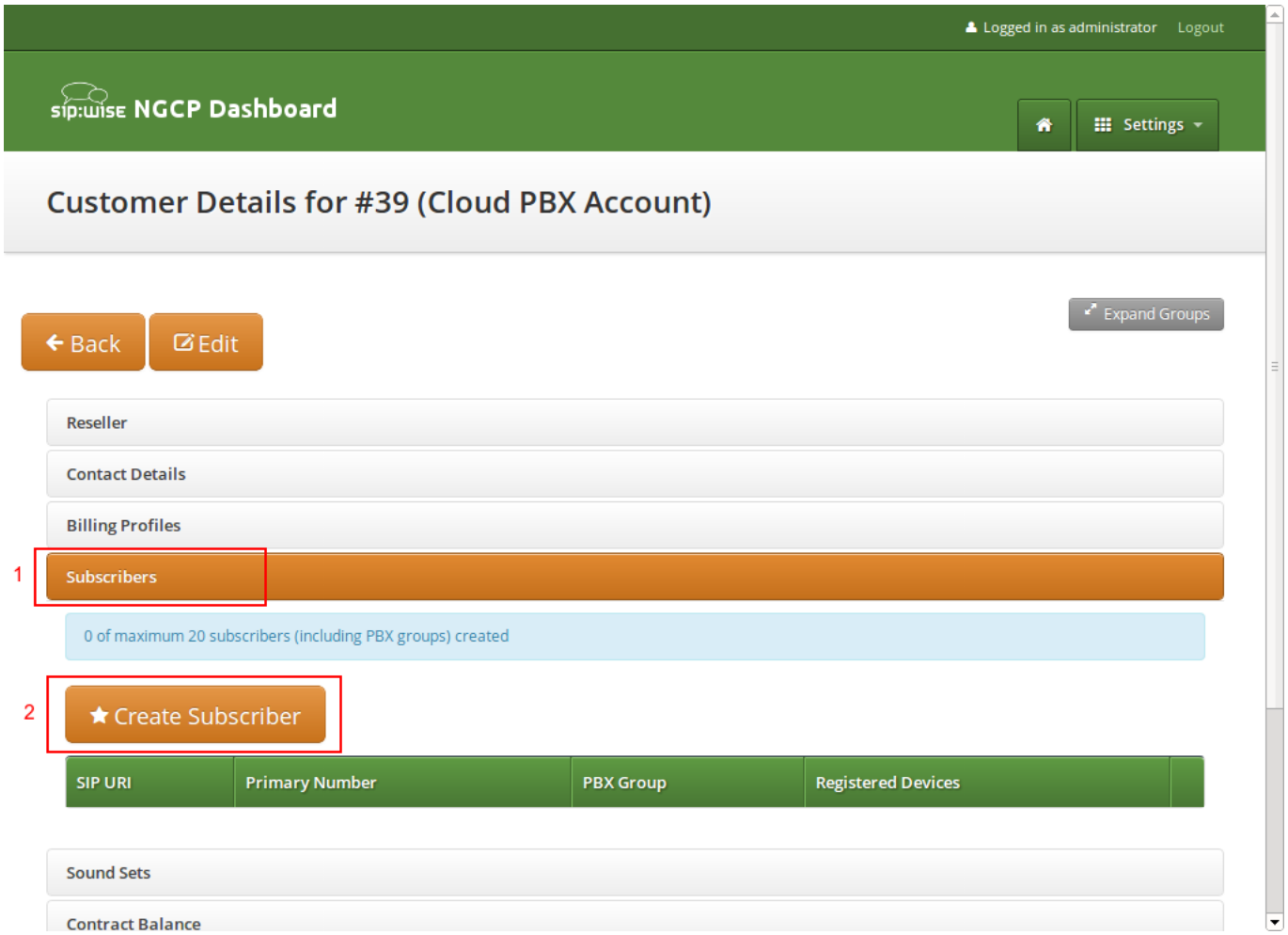


Figure 28: Go to Create Subscriber

For your pilot subscriber, you need a SIP domain, a pilot number (the main number of the customer PBX), the web credentials for the customer to log into the web interfaces, and the SIP credentials to authenticate via a SIP device.

Important

In a PBX environment, customers can create their own subscribers. As a consequence, each PBX customer should have its own SIP domain, in order to not collide with subscribers created by other customers. This is important because two customers are highly likely to create a subscriber (or group, which is also just a subscriber) called `office`. If they are in the same SIP domain, they'd both have the SIP URI `office@pbx.example.org`, which is not allowed, and the an end customer will probably not understand why `office@pbx.example.org` is already taken, because he (for obvious reasons, as it belongs to a different customer) will not see this subscriber in his subscribers list.

Tip

To handle one domain per customer, you should create a wild-card entry into your DNS server like `*.pbx.example.org`, which points to the IP address of `pbx.example.org`, so you can define SIP domains like `customer1.pbx.example.org` or `customer2.pbx.example.org` without having to create a new DNS entry for each of them. For proper secure access to the web interface and to the SIP and XMPP services, you should also obtain a SSL wild-card certificate for `*.pbx.example.org` to avoid certification warnings on customers' web browsers and SIP/XMPP clients.

So to create a new domain for the customer, click *Create Domain*.

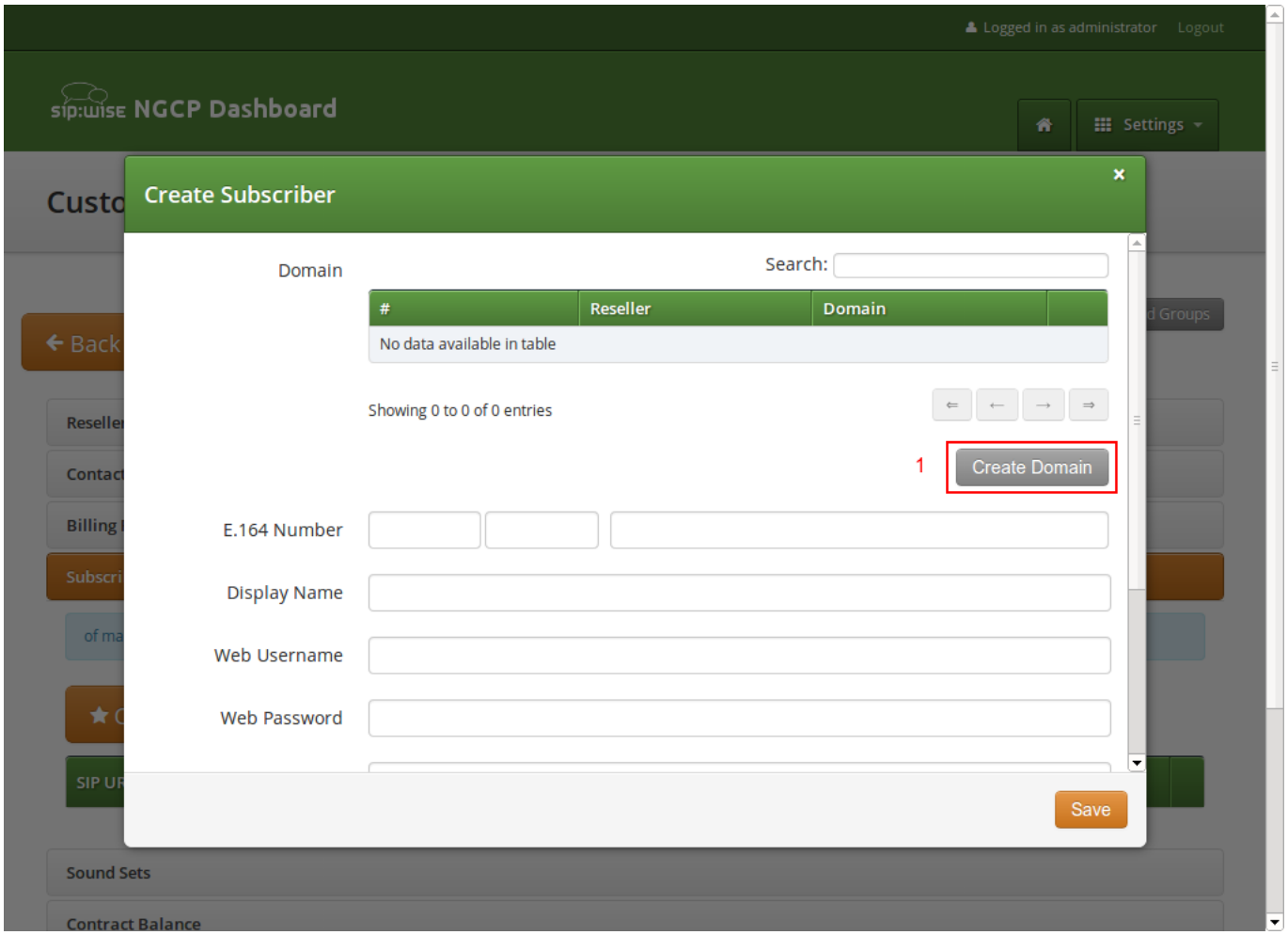


Figure 29: Go to Create Customer Domain

Specify the domain you want to create, and select the PBX *Rewrite Rule Set* which you created in Section A.2, then click *Save*.

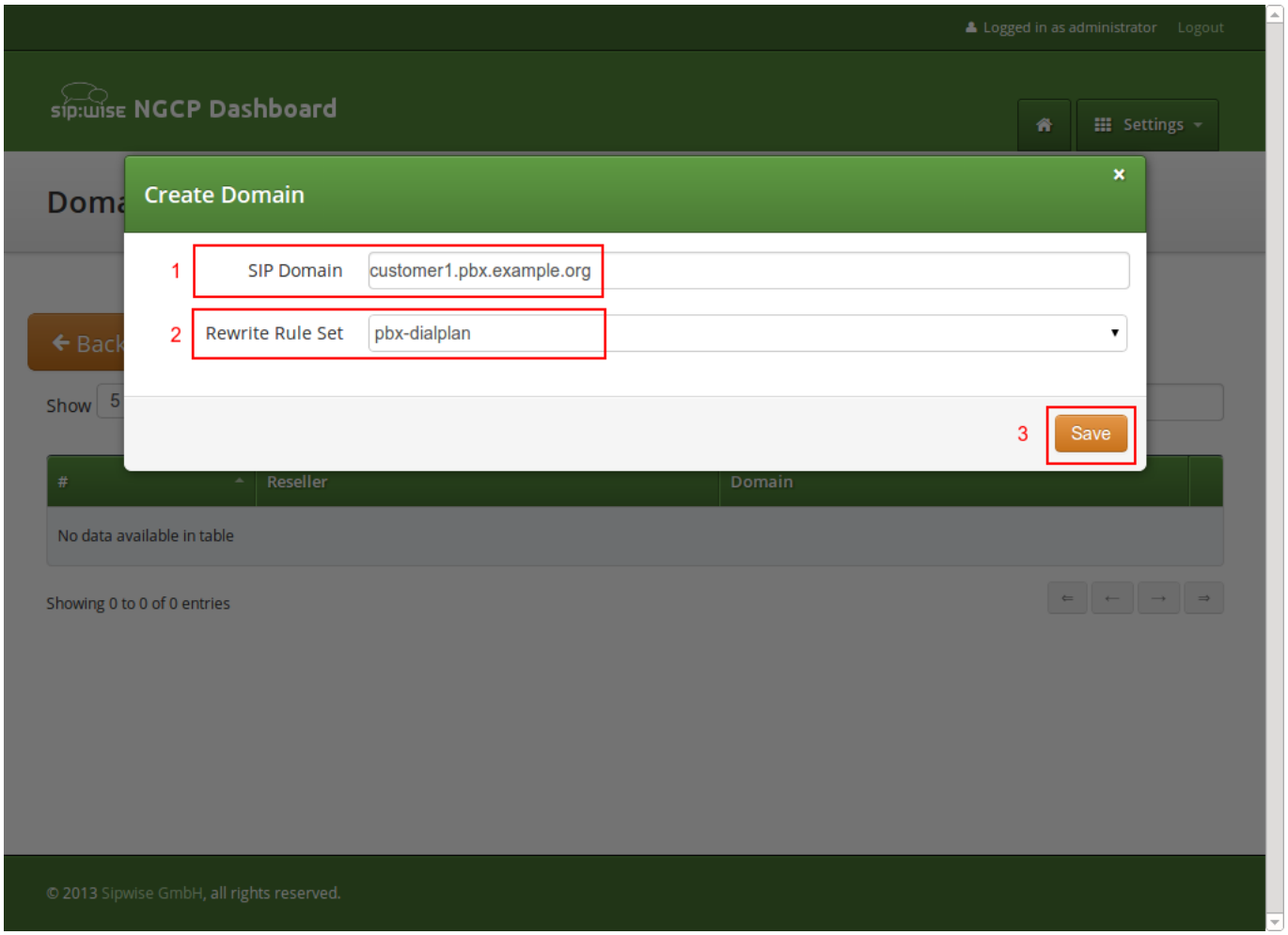


Figure 30: Create Customer Domain

Finish the subscriber creation by providing an E.164 number, which is going to be the base number for all other subscribers within this customer, the web username and password for the pilot subscriber to log into the web interface, and the sip username and password for a SIP device to connect to the PBX.

The parameters are as follows:

- **Domain**: The domain in which to create the pilot subscriber. *Each customer should get his own domain as described above to not collide with SIP usernames between customers*.
- **E.164 Number**: The primary number of the PBX. Calls to this number are routed to the pilot subscriber, and each subsequent subscriber created for this customer will use this number as its base number, suffixed by an individual extension. You can later assign alias numbers also for DID support.
- **Display Name**: This field is used on phones to identify subscribers by their real names instead of their number or extension. On outbound calls, the display name is signalled in the Display-Field of the From header, and it's used as a name in the XMPP contact lists.
- **Web Username**: The username for the subscriber to log into the customer self-care web interface. This is optional, if you don't

want a subscriber to have access to the web interface.

- **Web Password**: The password for the subscriber to log into the customer self-care web interface.
- **SIP Username**: The username for the subscriber to authenticate on the SIP and XMPP service. It is automatically used for devices, which are auto-provisioned via the *Device Management*, or can be used manually by subscribers to sign into the SIP and XMPP service with any arbitrary clients.
- **SIP Password**: The password for the subscriber to authenticate on the SIP and XMPP service.

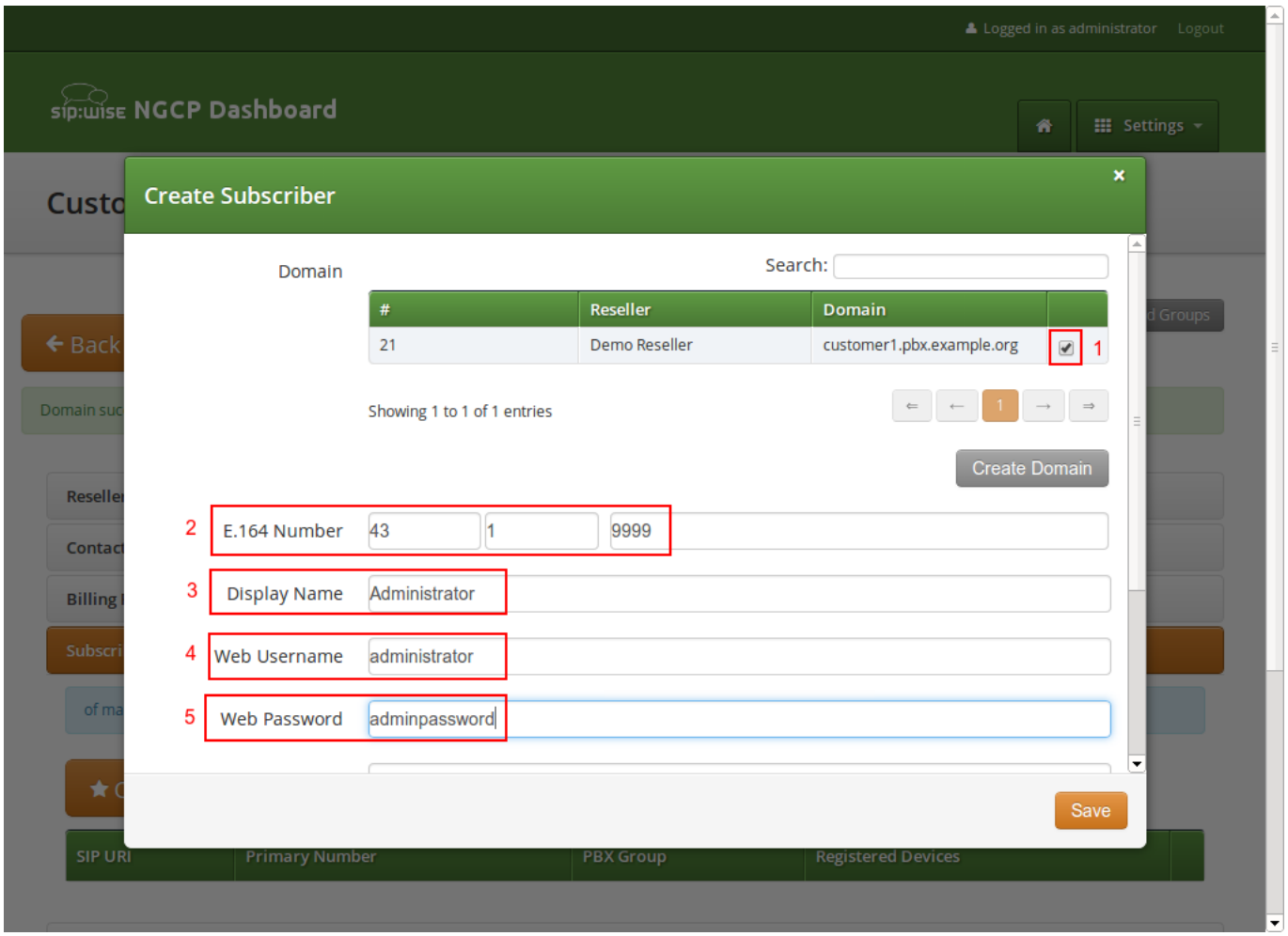


Figure 31: Create Pilot Subscriber Part 1

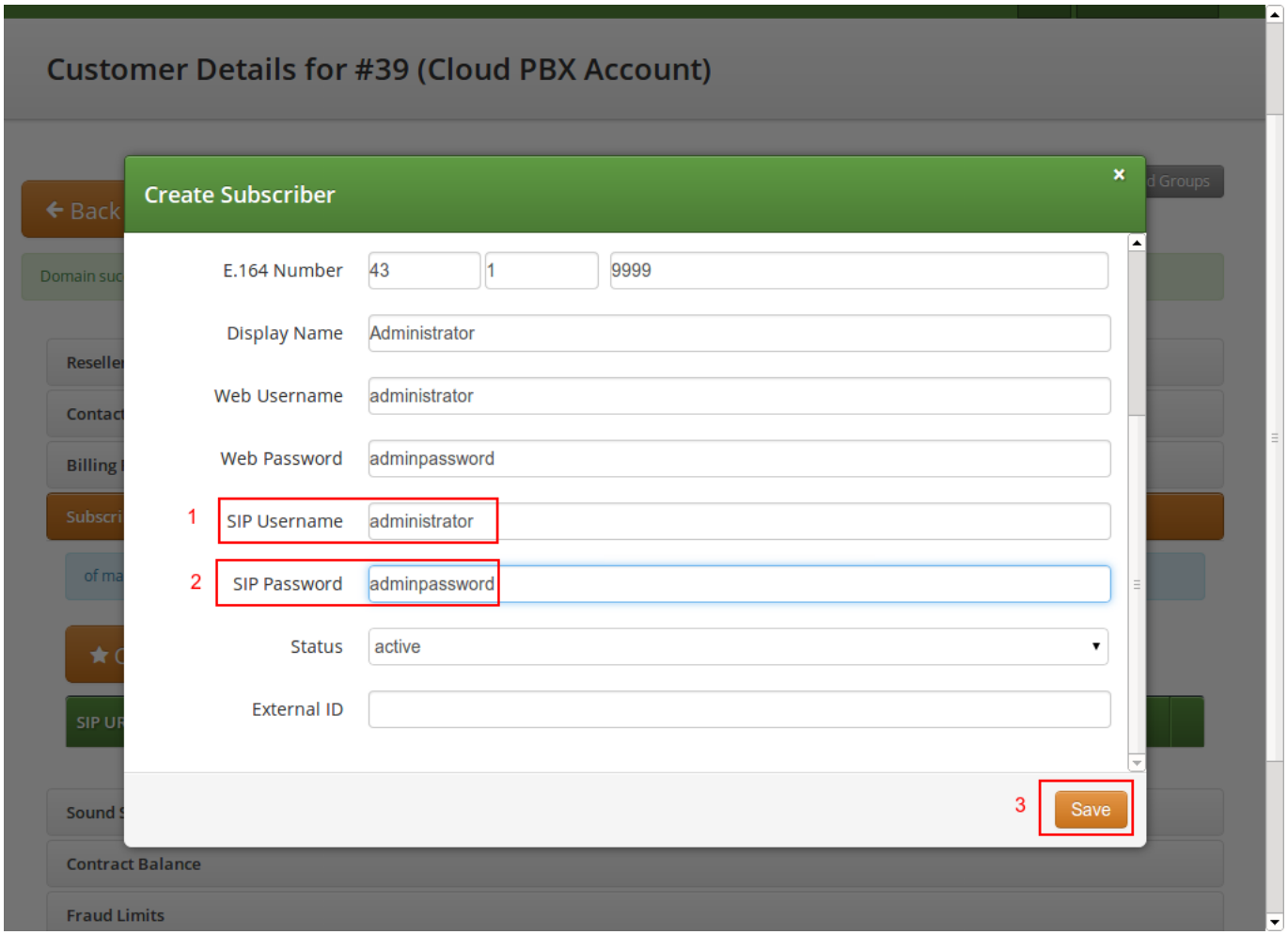


Figure 32: Create Pilot Subscriber Part 2

Once the subscriber is created, he can log into the customer self-care interface at `https://<your-ip>:1443/login/subscriber` and manage his PBX, like creating other users and groups, assigning Devices to subscribers and configure the Auto Attendant and more.

A.4 Managing a Customer PBX

With the pilot subscriber created before, the customer can log into the customer self-care interface and manage the PBX.

As an administrator, you can also do this for him, and we will walk through the typical steps as an administrator to configure the different features.

Go the the *Customer Details* of the PBX customer you want to configure, e.g. by navigating to *Settings→Customers* and clicking the *Details* button of the customer you want to configure.

A.4.1 Creating more Subscribers

Since we already created a pilot subscriber, more settings now appear on the *Customer Details* view. The sections we're interested in for now are the *Subscribers* and *PBX Groups* sections.

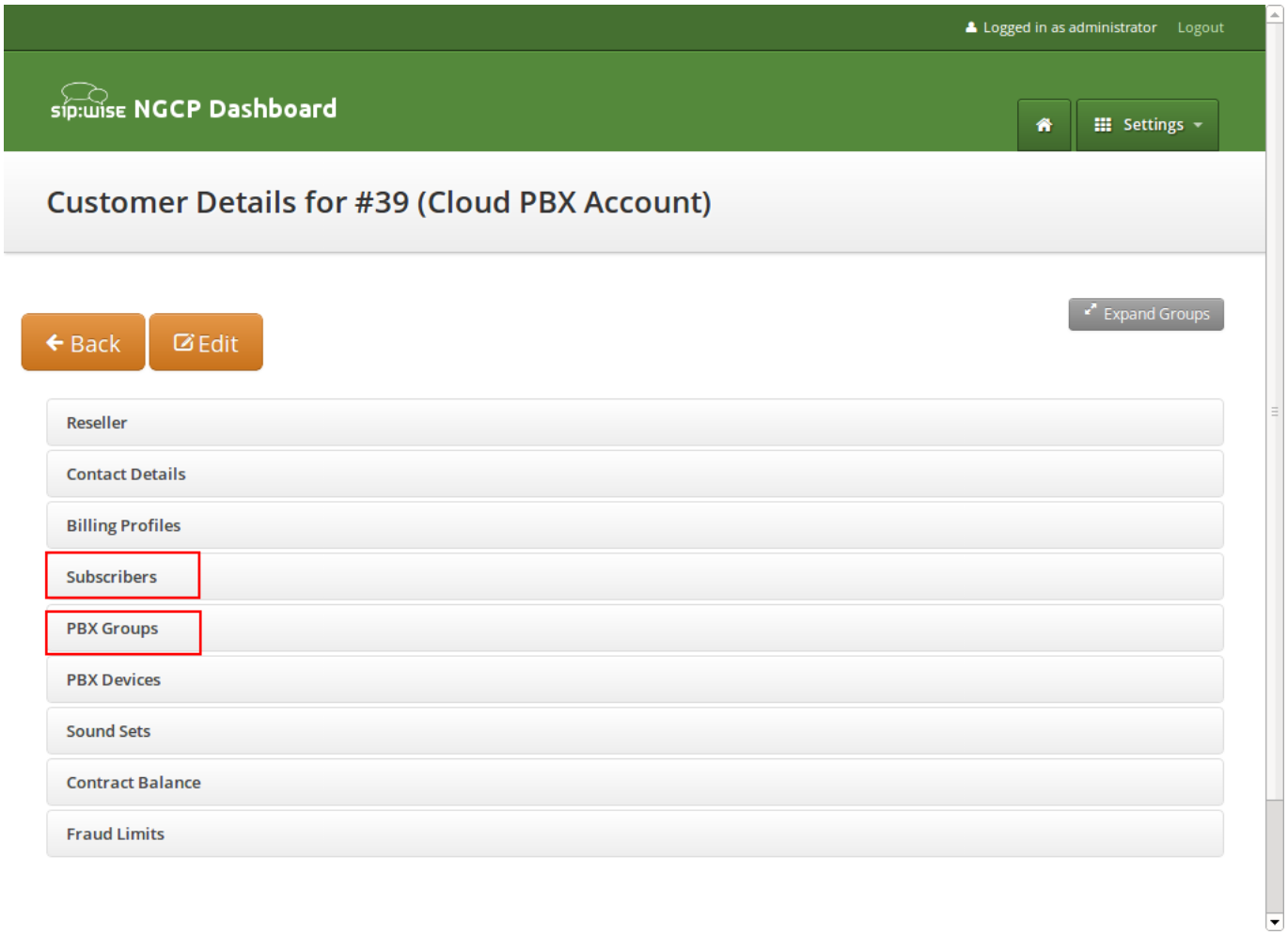


Figure 33: Subscribers and PBX Groups

To create another subscriber for the customer PBX, open the *Subscribers* row and click *Create Subscriber*.

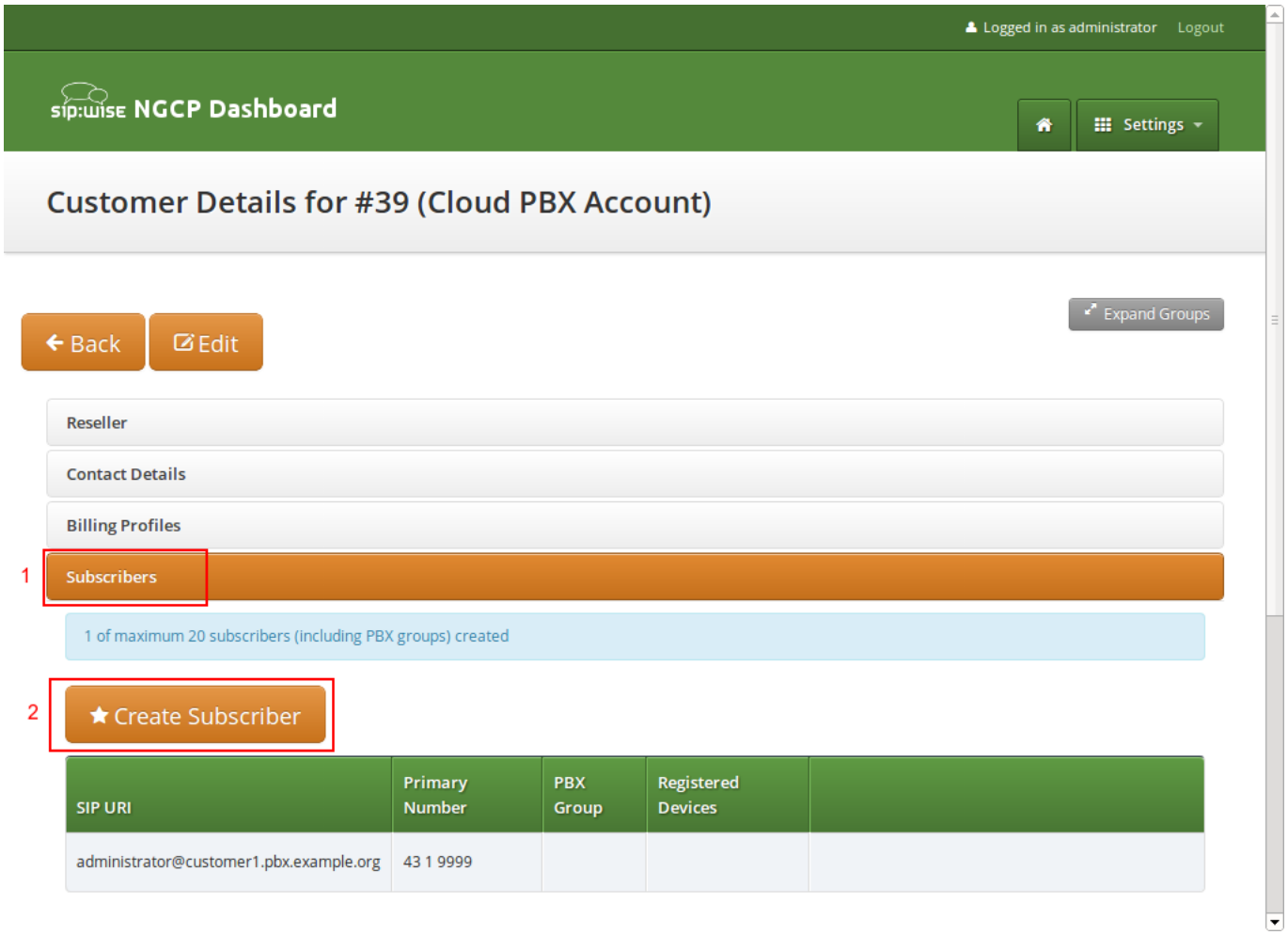


Figure 34: Create a Subscriber Extension

When creating another subscriber in the PBX after having the pilot subscriber, some fields are different now, because the *Domain* and *E.164 Number* are already pre-defined at the pilot subscriber level.

What you need to define for a new subscriber is the *Group* the subscriber is supposed to be in. We don't have a group yet, so create one by clicking *Create Group*.

A *PBX Group* has four settings:

- **Name**: The name of the group. This is used to identify a group when assigning it to subscribers on one hand, and also subscribers are pushed as server side contact lists to XMPP clients, where they are logically placed into their corresponding groups.
- **Extension**: The extension of the group, which is appended to the primary number of the pilot subscriber, so you can actually call the group from the outside. If our pilot subscriber number is `43 1 9999` and the extension is `100`, you can reach the group from the outside by dialing `43 1 9999 100`. Since PBX Groups are actually just normal subscribers in the system, you can assign *Alias Numbers* to it for DID later, e.g. `43 1 9998`.
- **Hunting Policy**: If you call a group, then all members in this group are ringing based on the policy you choose. `Serial`

`Ringing` causes each of the subscribers to be tried one after another, until one of them picks up or all subscribers are tried. `Parallel Ringing` causes all subscribers in the group to be tried in parallel. Note that a subscriber can have a call-forward configured to some external number (e.g. his mobile phone), which will work as well.

- **Serial Hunting Timeout**: This value defines for how long to ring each member of a group in case of serial hunting until the next subscriber is being tried.

We will only fill in the *Name* and *Extension* for now, as the hunting policy can be changed later if needed. Click *Save* to create the group.

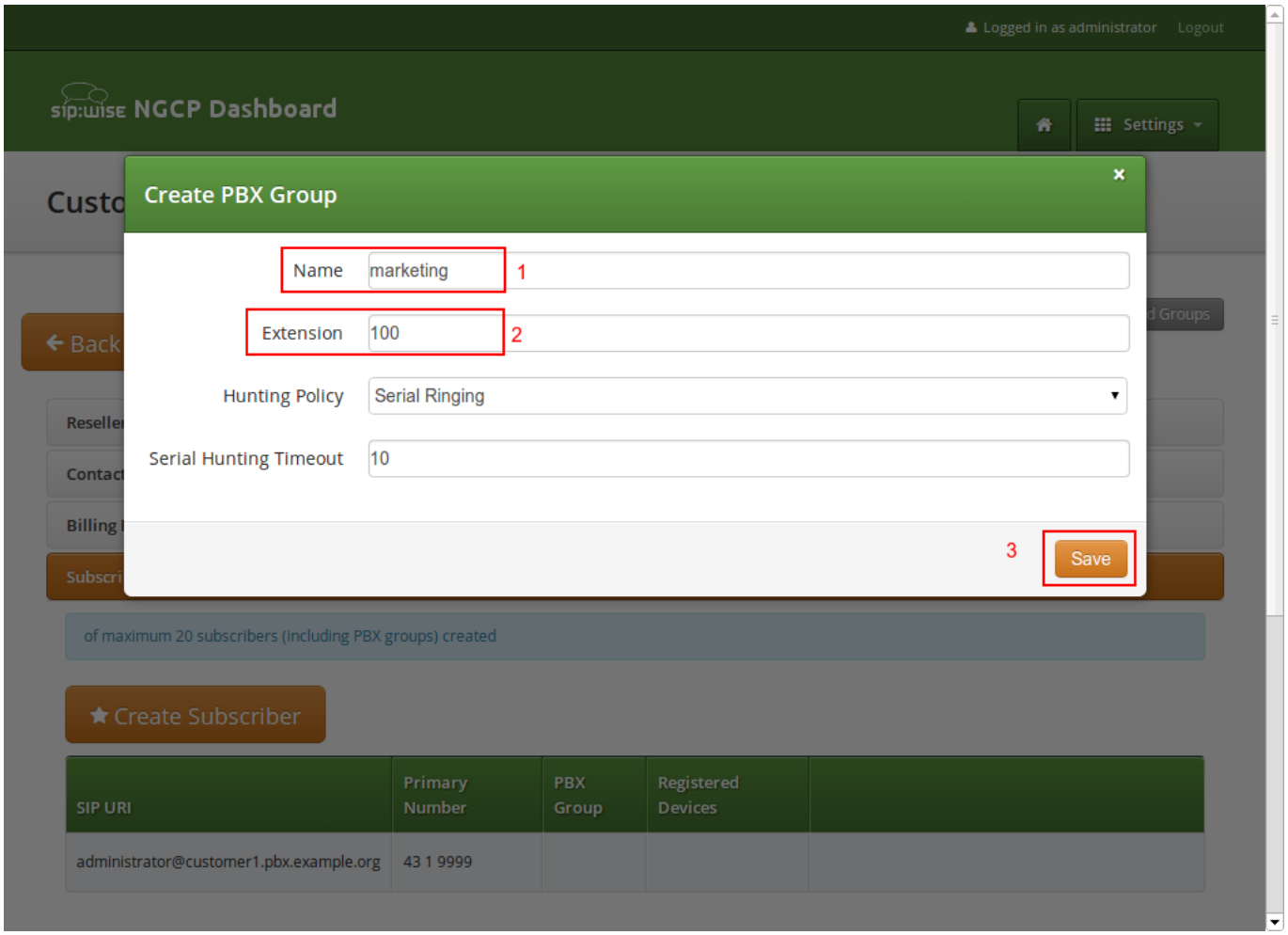


Figure 35: Create a PBX Group

Once the group is created and selected, fill out the rest of the form as needed. Instead of the *E.164 Number*, you can now only choose the *Extension*, which is appended to the primary number of the pilot subscriber and is then used as primary number for this particular subscribers. Again, if your pilot number is `43 1 9999` and you choose extension `101` here, the number of this subscriber is going to be `43 1 9999 101`. Also, you can again later assign more alias numbers (e.g. `43 1 9997`) to this subscriber for DID.

The rest of the fields is as usual, with *Display Name* defining the real name of the user, *Web Username* and *Web Password* allowing the subscriber to log into the customer self-care interface, and the *SIP Username* and *SIP Password* to allow signing into

the SIP and XMPP services.

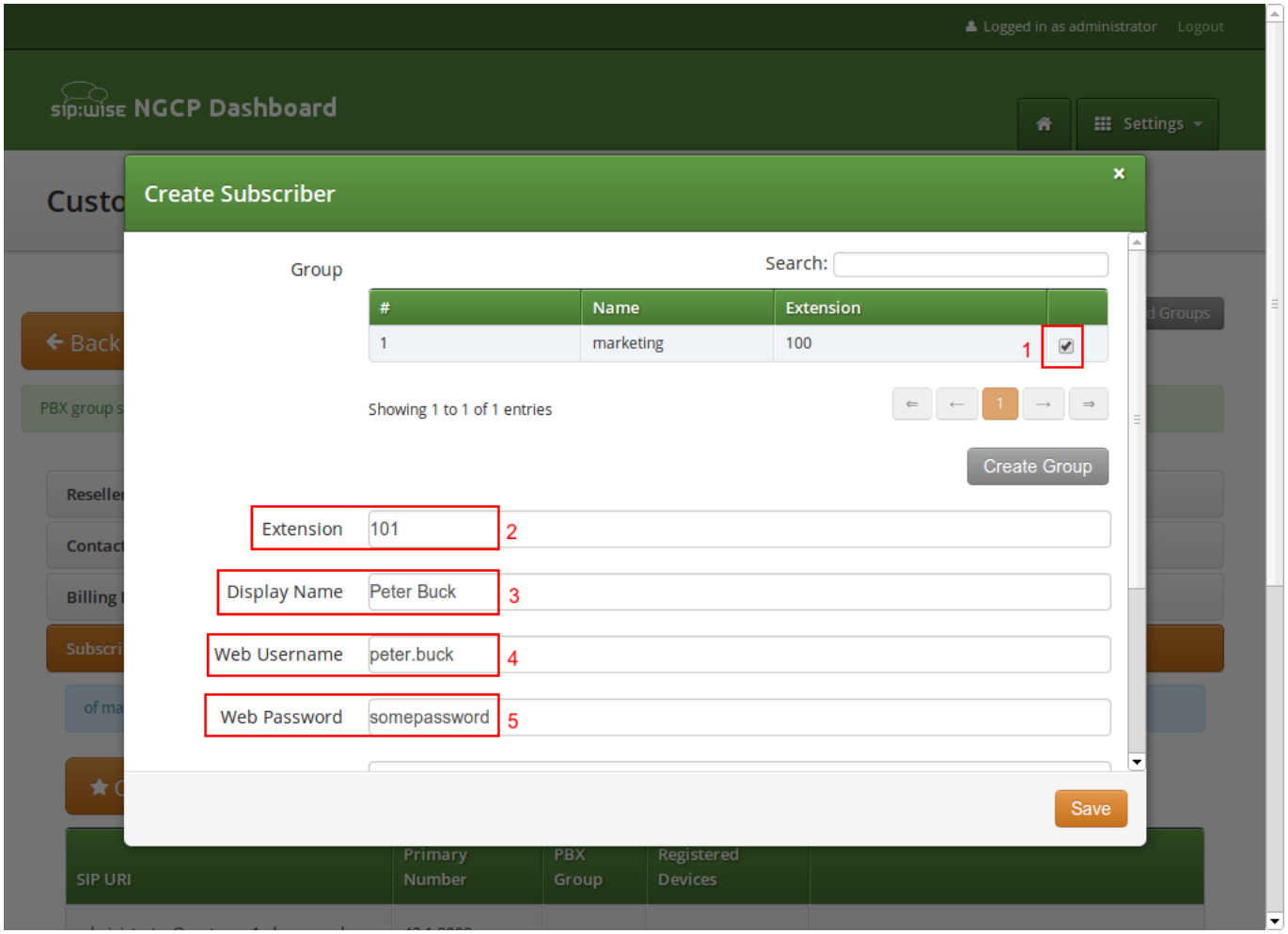


Figure 36: Finish PBX Subscriber Creation Part 1

Click *Save* to create the subscriber.

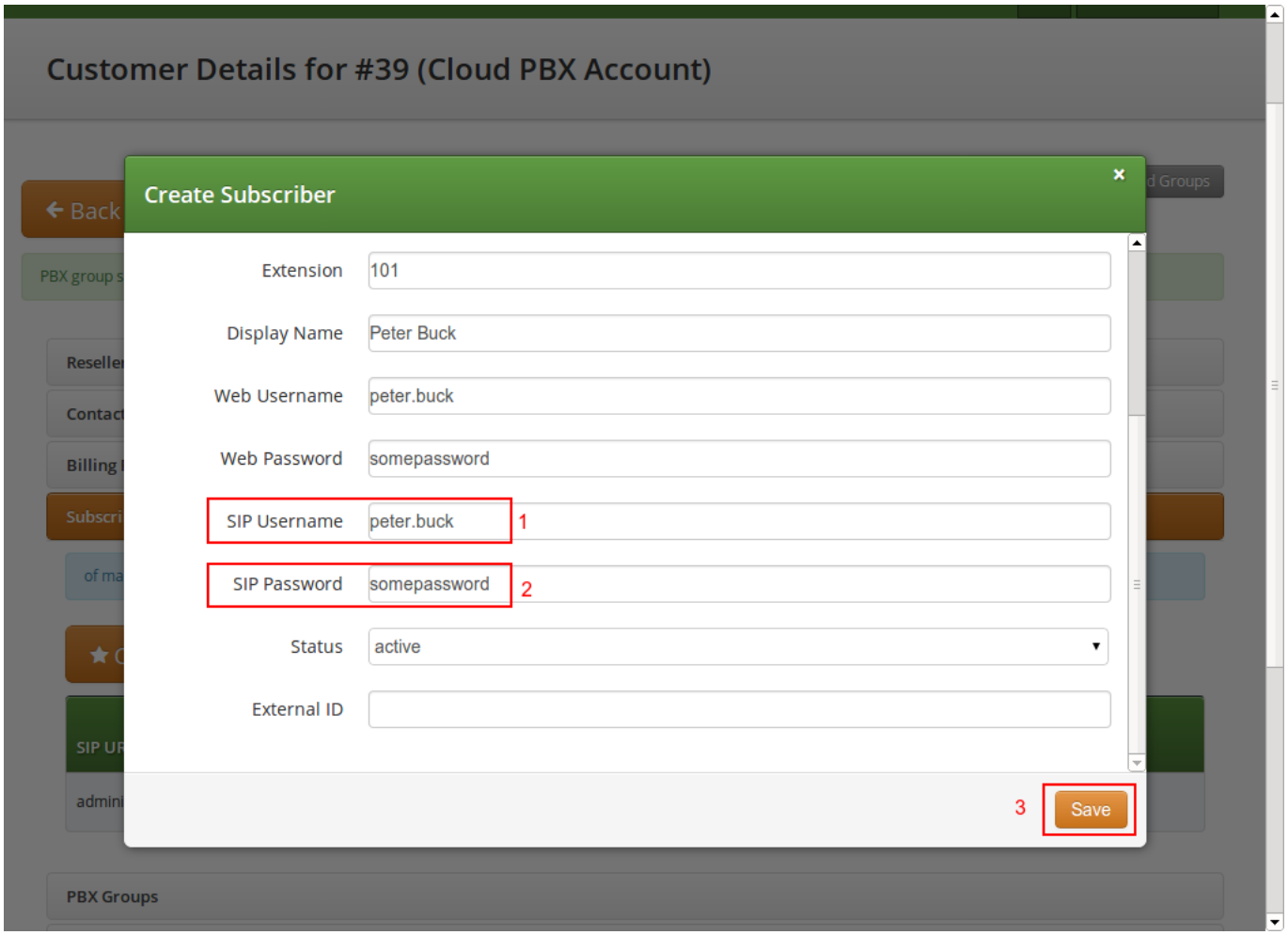


Figure 37: Finish PBX Subscriber Creation Part 2

Repeat the steps to create all the subscribers and groups as needed. An example of a small company configuration in terms of subscribers and groups might look like this:

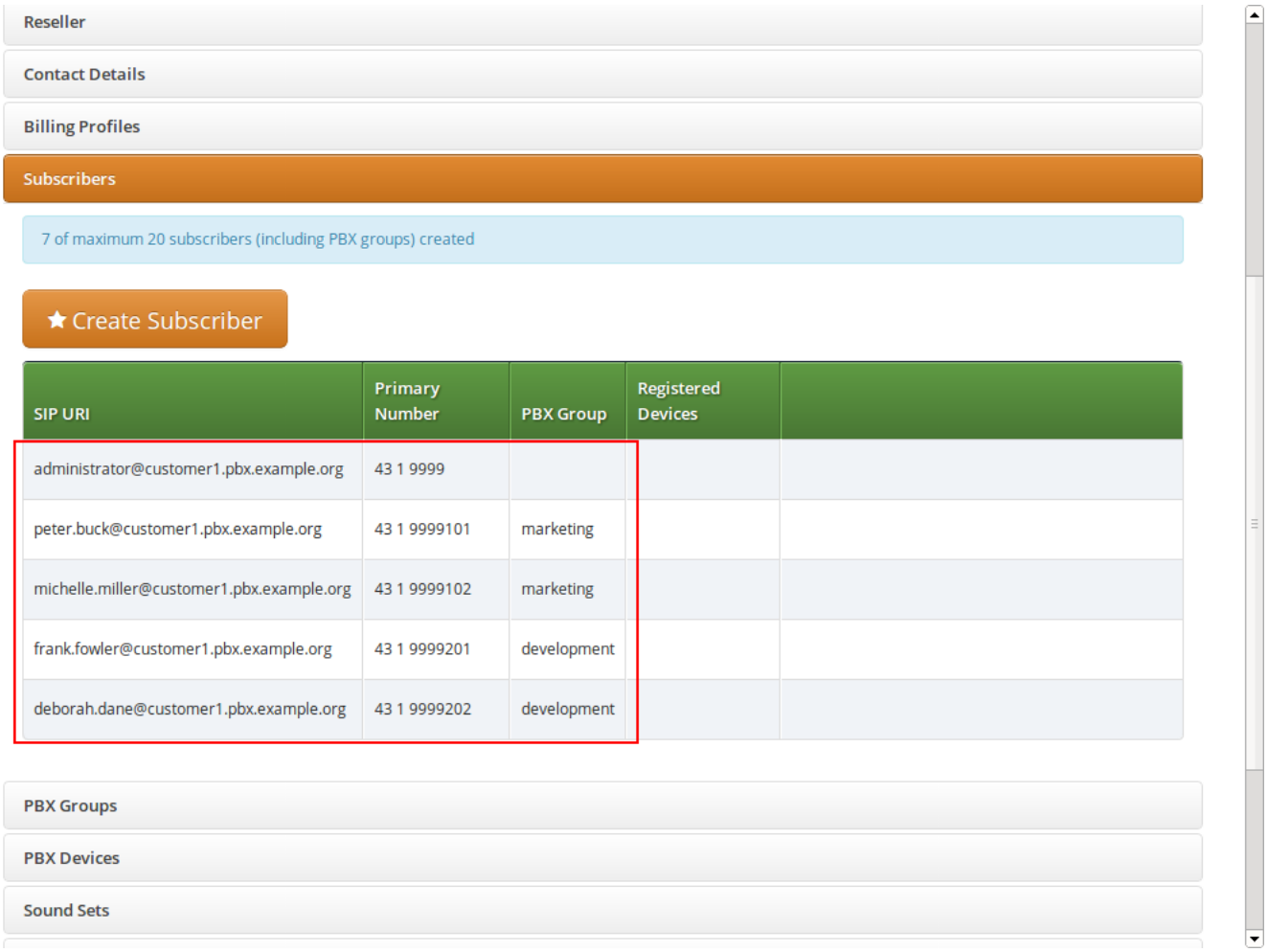


Figure 38: Example of Subscribers List

Tip

The subscribers can be reached via 3 different ways. First, you can call them by their SIP URIs (e.g. by dialing `frank.fowler@customer1.pbx.example.org`) from both inside and outside the PBX. Second, you can dial by the full number (e.g. `43 1 9999 201`; depending on your rewrite rules, you might need to add a leading `\+` or `00` or leave out the country code when dialing from the outside, and adding a `0` as break-out digit when dialing from the inside) from both inside and outside the PBX. Third, you can dial just the extension (e.g. `201`) from inside the PBX. If the subscriber also has an alias number assigned, you can dial that number also, according to your dial-plan in the rewrite rules.

A.4.2 Assigning Subscribers to Devices

Basically you can register any SIP phone to the system using the SIP credentials of your subscribers. However, the platform supports *Device Provisioning* of certain vendors and models, as described in Section A.1.

To configure a physical device, open the *PBX Devices* row in the *Customer Details* view and click *Create Device*.

You have to set three general parameters for your new device, which are:

- **Device Profile**: The actual device profile you want to use. This has been pre-configured in the *Device Management* by the administrator or reseller, and the customer can choose from the list of profiles (which is a combination of an actual device plus its corresponding configuration).
- **MAC Address/Identifier**: The MAC address of the phone to be added. The information can usually either be found on the back of the phone, or in the phone menu itself.
- **Station Name**: Since you can (depending on the actual device) configure more lines on a phone, you can give it a station name, like `Reception` or the name of the owner of the device.

In addition to that information, you can configure the lines (subscribers) you want to use on which key, and the mode of operation (e.g. if it's a normal private phone line, or if you want to monitor another subscriber using BLF, or if you want it to act as shared line using SLA).

For example, a *Cisco SPA504G* has 4 keys you can use for private and shared lines as well as BLF on the phone itself, and in our example we have an *Attendant Console* attached to it as well, so you have 32 more keys for BLF.

The settings per key are as follows:

- **Subscriber**: The subscriber to use (for private/shared lines) or to monitor (for BLF).
- **Line/Key**: The key where to configure this subscriber to.
- **Line/Key Type**: The mode of operation for this key, with the following options (depending on which options are enabled in the *Device Model* configuration for this device:
  - **Private Line**: Use the subscriber as a regular SIP phone line. This means that the phone will register the subscriber, and you can place and receive phone calls with/for this subscriber.
  - **Shared Line**: The subscriber is also registered on the system and you can place and receive calls. If another phone has the same subscriber also configured as shared line, both phones will ring on incoming calls, and you can pick the call up on either of them. You cannot place a call with this subscriber though if the line is already in use by another subscriber. However, you can "steal" a running call by pressing the key where the shared line is configured to barge into a running call. The other party (the other phone where the shared line is configured too) will then be removed from the call (but can steal the call back the same way).
  - **BLF Key**: The *Busy Lamp Field* monitors the call state of another subscriber and provides three different functionalities, depending on the actual state:
    - **Speed Dial**: If the monitored subscriber is on-hook, the user can press the button and directly call the monitored subscriber.
    - **Call Pickup**: If the monitored subscriber is ringing, the user can press the button to pick up the call on his own phone.
    - **State Indication**: It the monitored subscriber is on the phone, the key is indicating that the monitored subscriber is currently busy.

In our example, we will first configure a private line on the first key, and BLF for another subscriber on the second key.

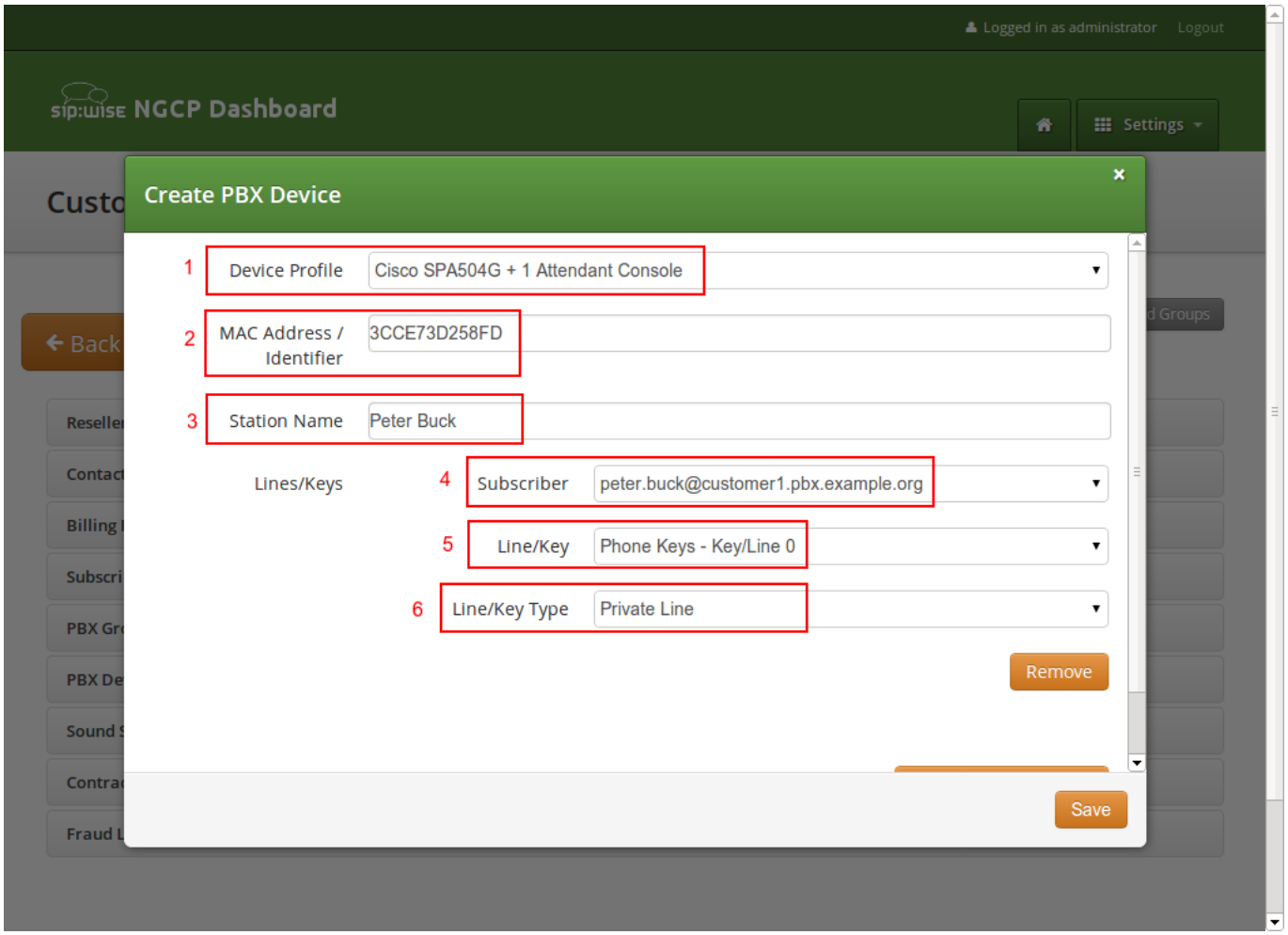


Figure 39: Configuring a PBX Device Part 1

This configures the general options plus the first key. To configure the second key, click *Add another Line/Key* and fill out the second line config accordingly. Click *Save* to save your PBX device configuration.

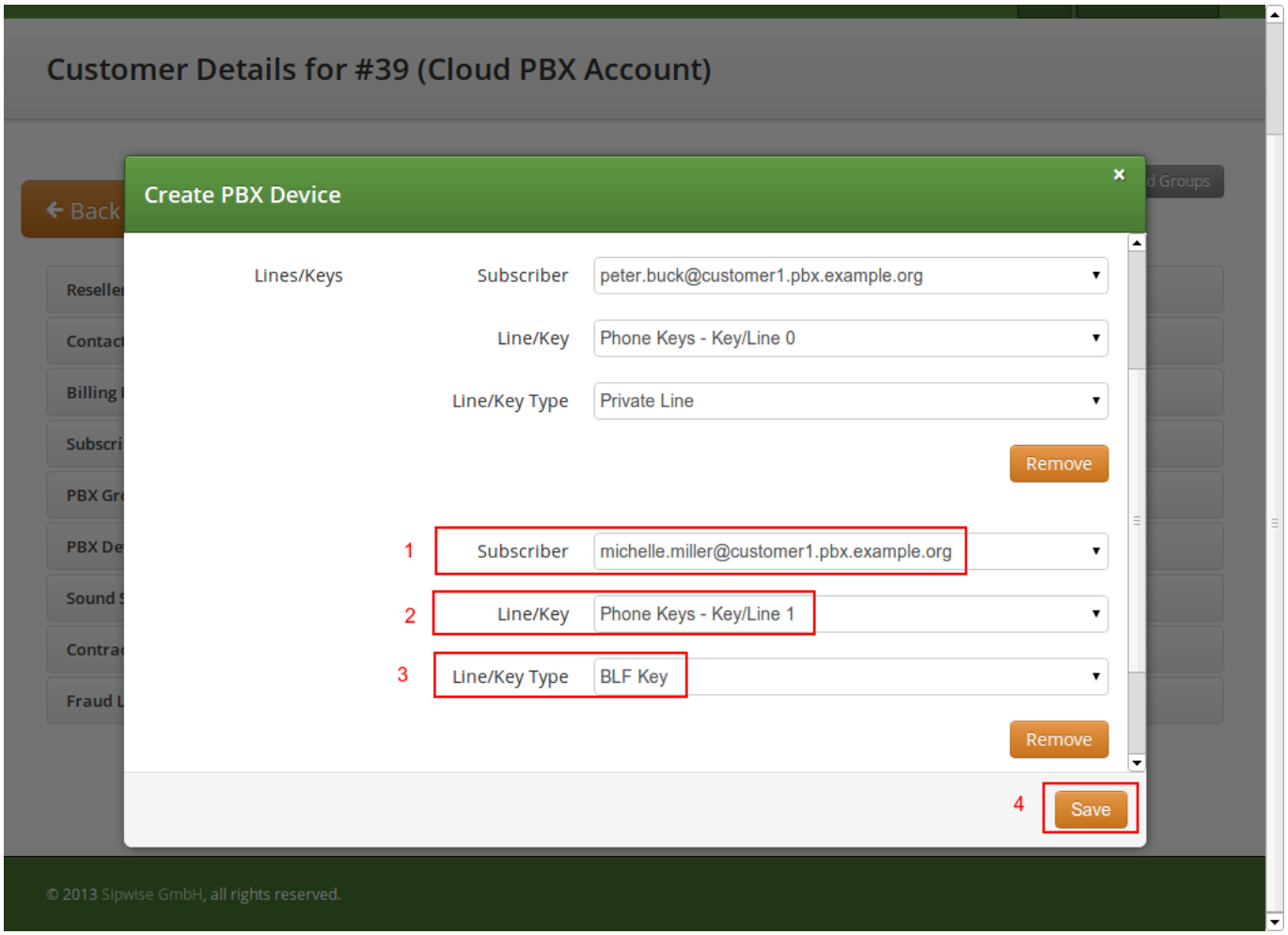


Figure 40: Configuring a PBX Device Part 2

Once the PBX device is saved, you will see it in the list of *PBX Devices*.

Synchronizing a PBX Device for initial Usage

Since a stock device obtained from an arbitrary distributor doesn't know anything about your system, it can't fetch its configuration from there. For that to work, you need to push the URL of where the phone can get the configuration to the phone once.

In order to do so, click the *Sync Device* button on the device you want to configure for the very first time.

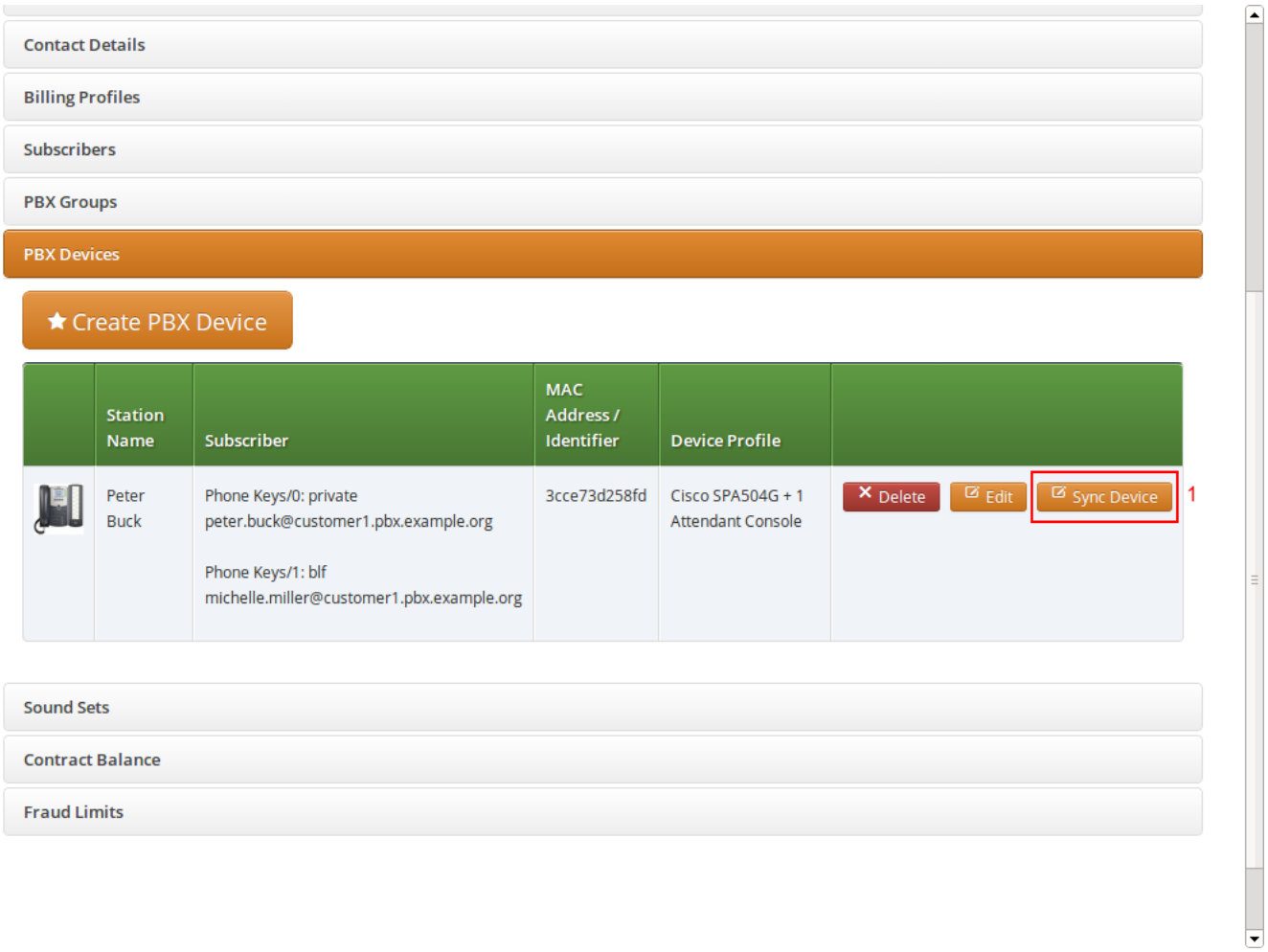


Figure 41: Go to Sync Device

Important

As you will see in the next step, you need the actual IP address of the phone to push the provisioning URL onto it. That implies that you need access to the phone to get the IP, and that your browser is in the same network as the phone in order to be able to connect to it, in case the phone is behind NAT.

Enter the IP Address of the phone (on Cisco SPAs, press `Settings 9`, where `Settings` is the paper sheet symbol, and note down the `Current IP` setting), then click *Push Provisioning URL*.

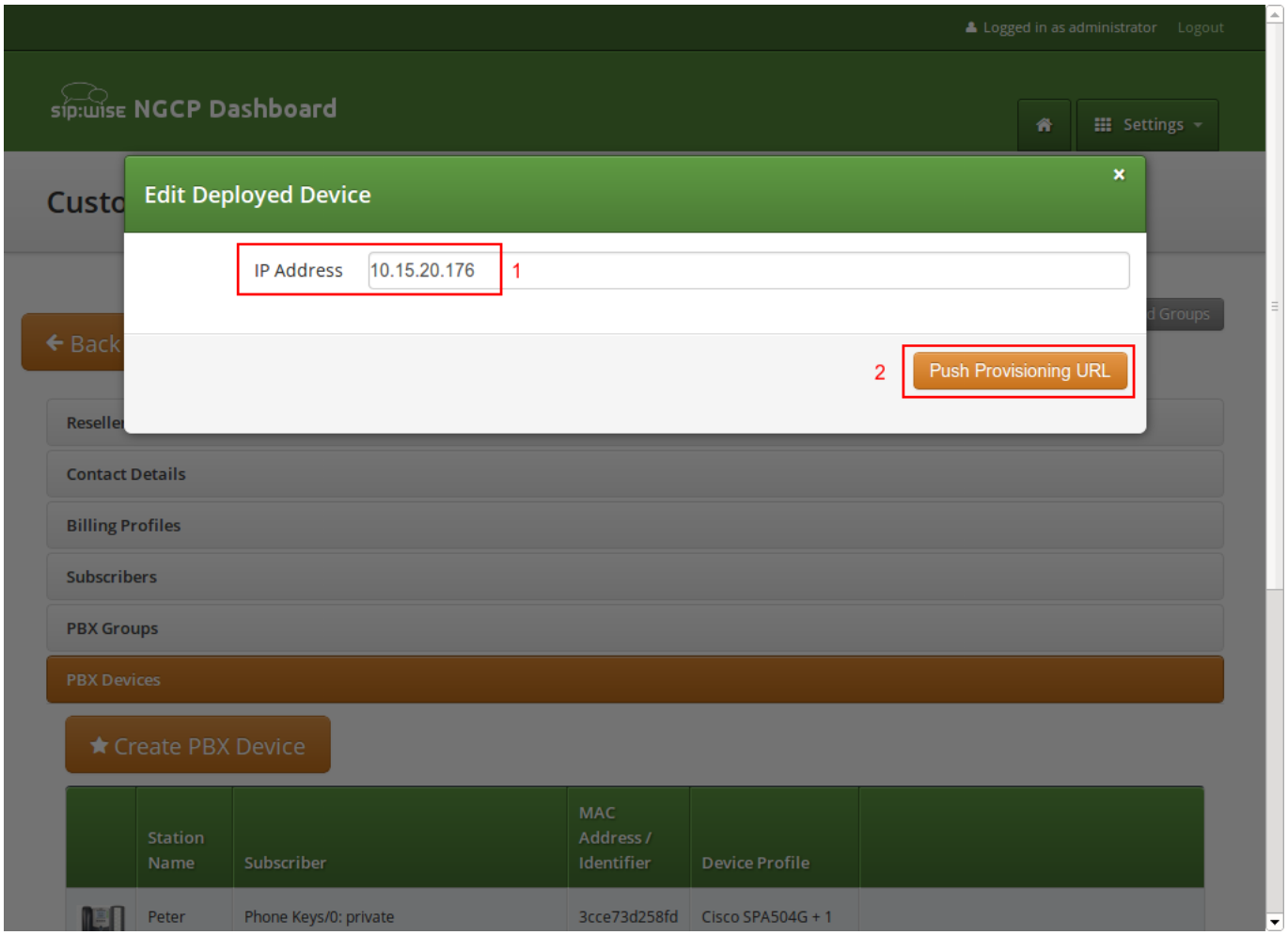


Figure 42: Sync Device

You will be redirected directly to the phone, and the Provisioning URL is automatically set. If everything goes right, you will see a confirmation page from the phone that it's going to reboot.

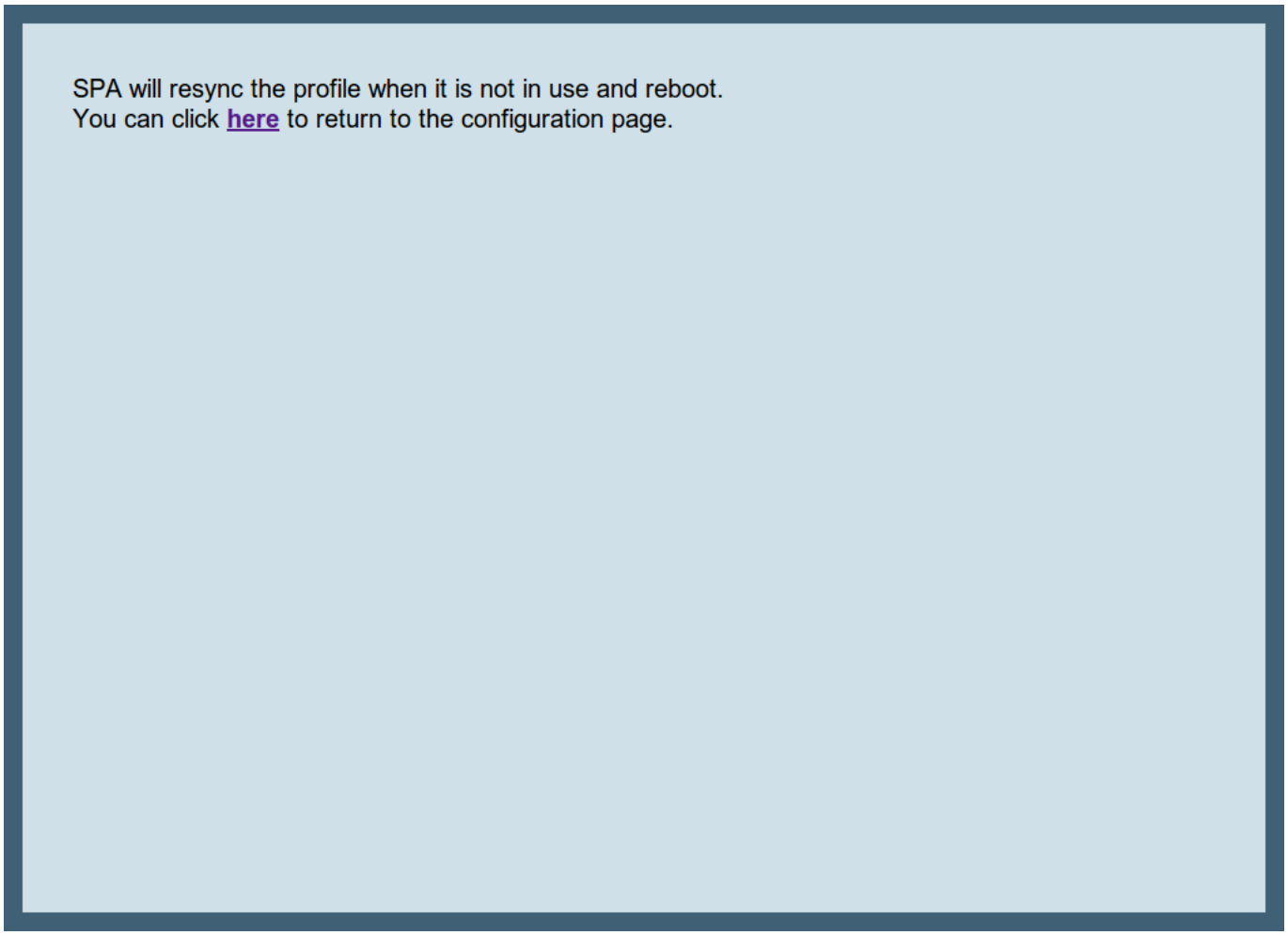


Figure 43: Device Sync Confirmation from Phone

You can close the browser window/tab and proceed to sync the next subscriber.

Tip

You only have to do this step once per phone to tell it the actual provisioning URL, where it can fetch the configuration from. From there, it will regularly sync with the server automatically to check for configuration changes, and applies them automatically.

A.4.3 Configuring Sound Sets for the Customer PBX

In the *Customer Details* view, there is a row *Sound Sets*, where the customer can define his own sound sets for *Auto Attendant*, *Music on Hold* and the *Office Hours Announcement*.

To create a new sound set, open the *Sound Sets* row and click *Create Sound Set*.

If you do this as administrator or reseller, the Reseller and/or Customer is pre-selected, so keep it as is. If you do this as customer, you don't see any *Reseller* or *Customer* fields.

So the important settings are:

- **Name**: The name of the sound set as it will appear in the *Subscriber Preferences*, where you can assign the sound set to a subscriber.
- **Description**: A more detailed description of the sound set.
- **Default for Subscribers**: If this setting is enabled, then the sound set is automatically assigned to all already existing subscribers which do NOT have a sound set assigned yet, and also for all newly created subscribers.

Fill in the settings and click *Save*.

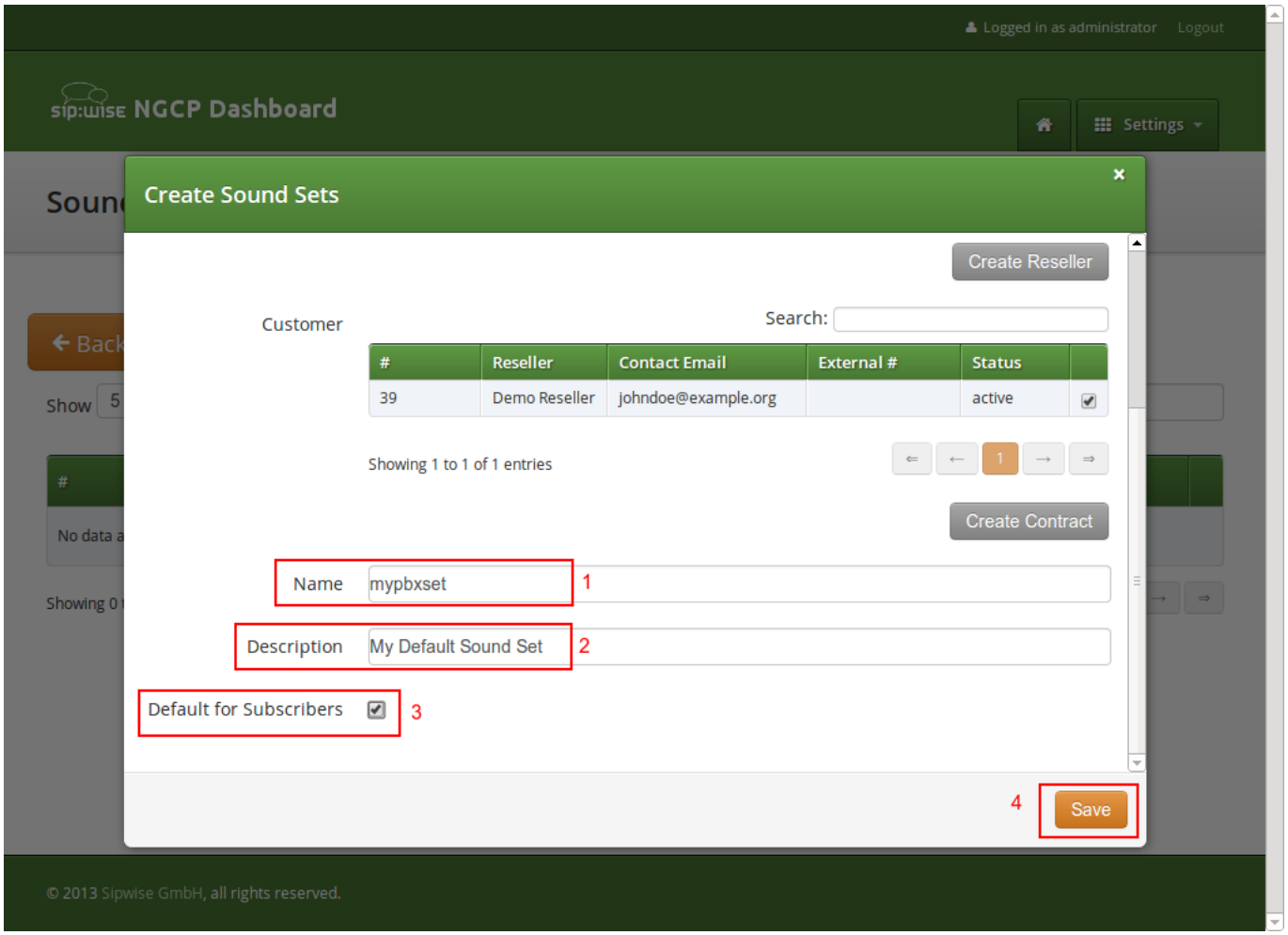


Figure 44: Create Customer Sound Set

To upload files to your Sound Set, click the *Files* button for the Sound Set.

Uploading a Music-on-Hold File

Open the *music_on_hold* row and click *Upload* on the *music_on_hold* entry. Choose a WAV file from your file system, and click the *Loopplay* setting if you want to play the file in a loop instead of just once. Click *Save* to upload the file.

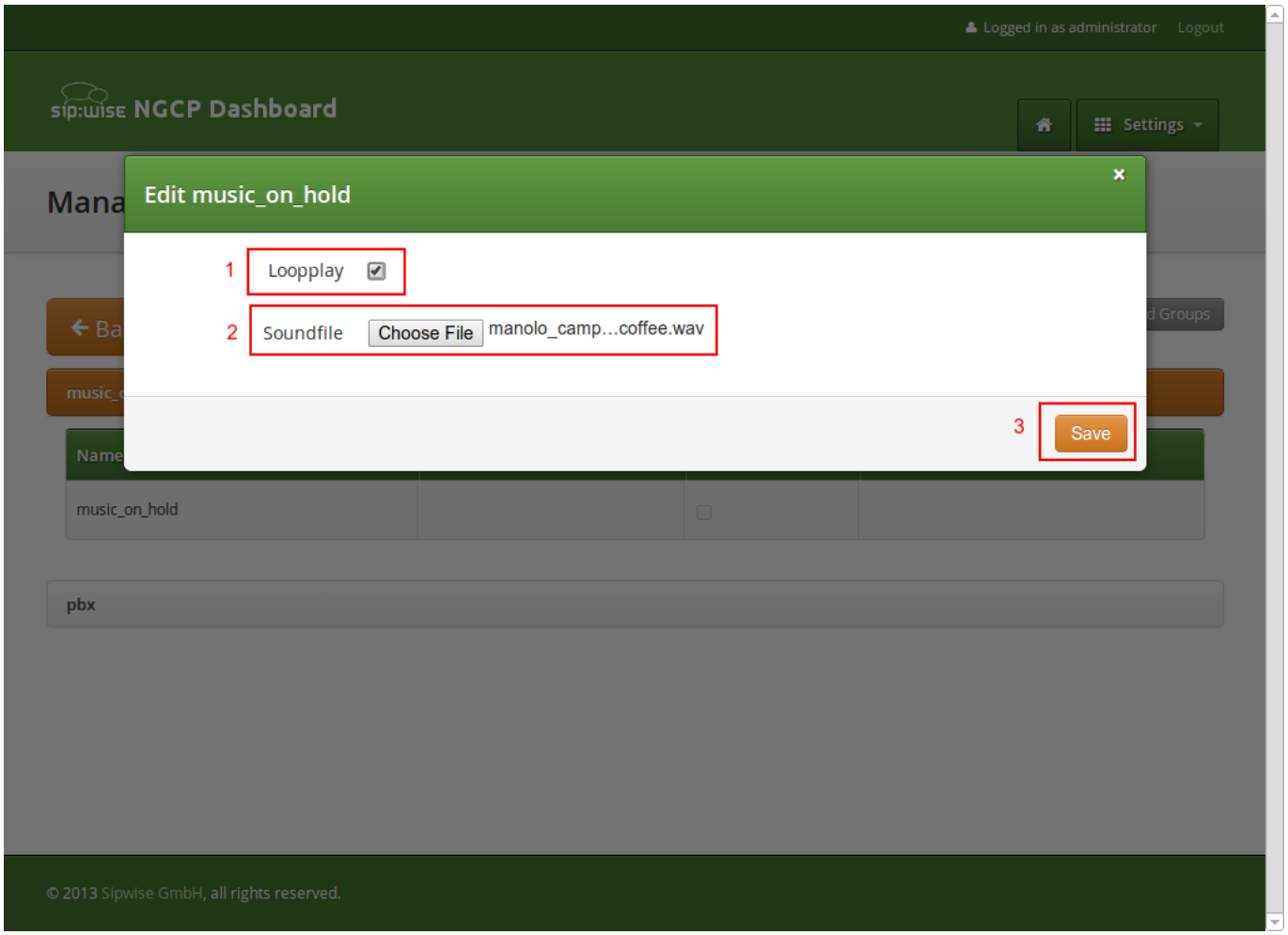


Figure 45: Upload MoH Sound File

Uploading Auto-Attendant Sound Files

When configuring a Call Forward to the *Auto Attendant*, it will play the following files:

- `aa_welcome`: This is the welcome message (the greeting) which is played when someone calls the Auto Attendant.
- each available pair of `aa_X_for`/`aa_X_option`: Each menu item in the Auto Attendant consists of two parts. The `for` part, which plays something like *Press One for*, and the `option` part, which play something like *Marketing*. The Auto Attendant only plays those menu options where both the `for` part and the `option` part is present, so if you only have 3 destinations you'd like to offer, and you want them to be on keys 1, 2 and 3, you have to upload files for `aa_1_for`, `aa_1_option`, `aa_2_for`, `aa_2_option` and `aa_3_for` and `aa_3_option`.

Important

The sound files only define the general structure of what is being played to the caller. The actual destinations behind your options are configured separately in Section A.4.4.

An example configuration could look like this:

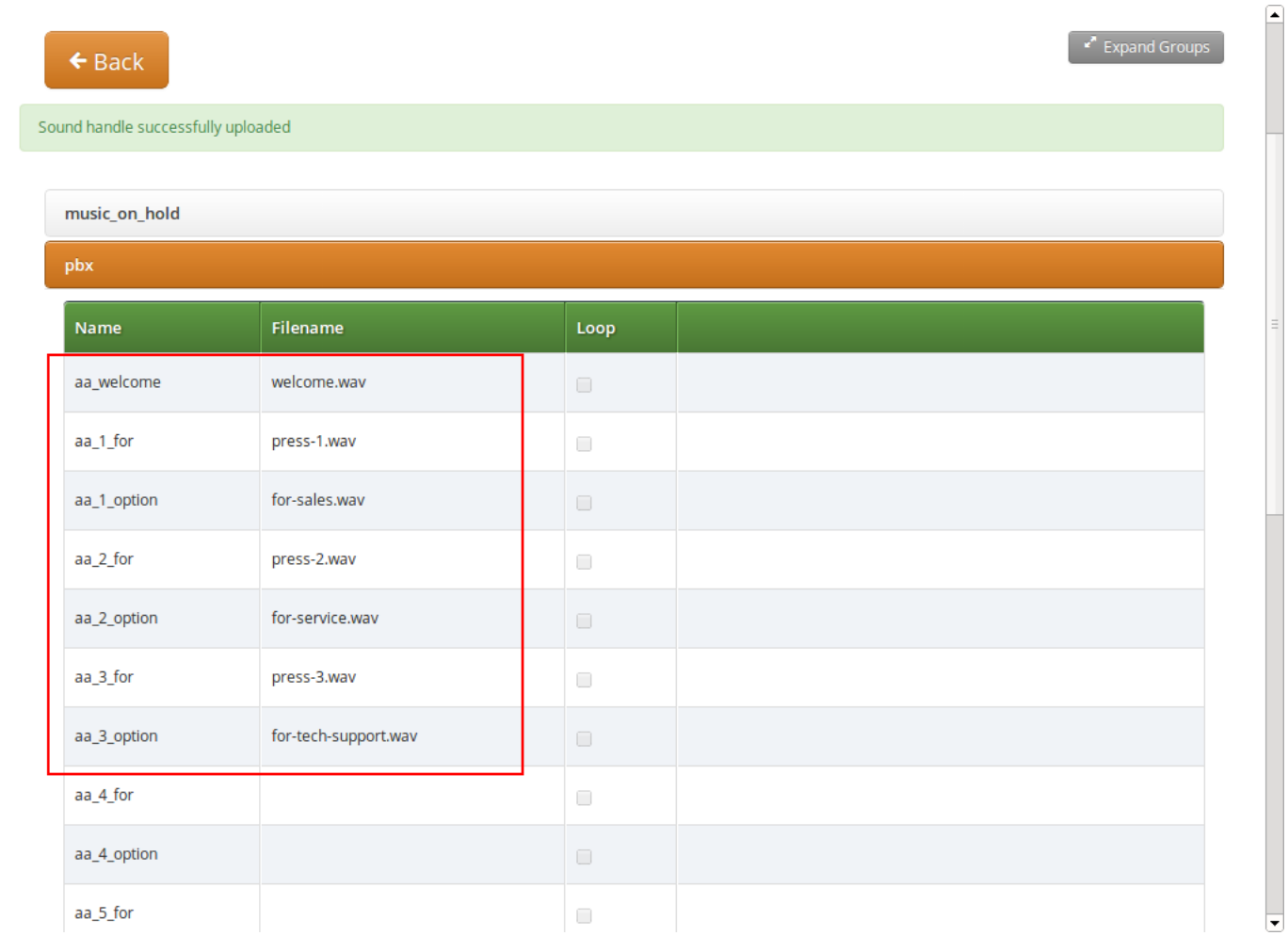


Figure 46: Upload Auto Attendant Sound File

A.4.4 Configuring the Auto Attendant

The Auto Attendant feature can be activated for any subscriber in the Customer PBX individually. There are three steps involved. First, you have to prepare a *Sound Set* to have Auto Attendant sound files. Second, you have to configure the destinations for the various options you provide (e.g. pressing `1` should go to the `marketing` subscriber, `2` to `development` and `3` to some external number). Third, you have to set a Call Forward to the Auto Attendant.

To do so, go to *Customer Details* and in the *Subscribers* section, click the *Preferences* button of the subscriber, where the Auto Attendant should be set.

Preparing the Sound Set

Create a Sound Set and upload the Sound Files for it as described in Section A.4.3. Back in the *Subscriber Preferences* view, set the *Customer Sound Set* preference to the Sound Set to be used. To do so, click *Edit* on the *Customer Sound Set* preference and assign the set to be used.

Configuring the Auto Attendant Slots

In the *Auto Attendant Slots* section, click the *Edit Slots* button to configure the destination options.

Click *Add another Slot* to add a destination option, select the Key the destination should be assigned to, and enter a Destination. The destination can be a subscriber username (e.g. `marketing`), a full SIP URI (e.g. `sip:michelle.miller@customer1.pbx.example.org` or any external SIP URI) or a number or extension (e.g. `491234567` or `101`).

Repeat the step for every option you want to add, then press *Save*.

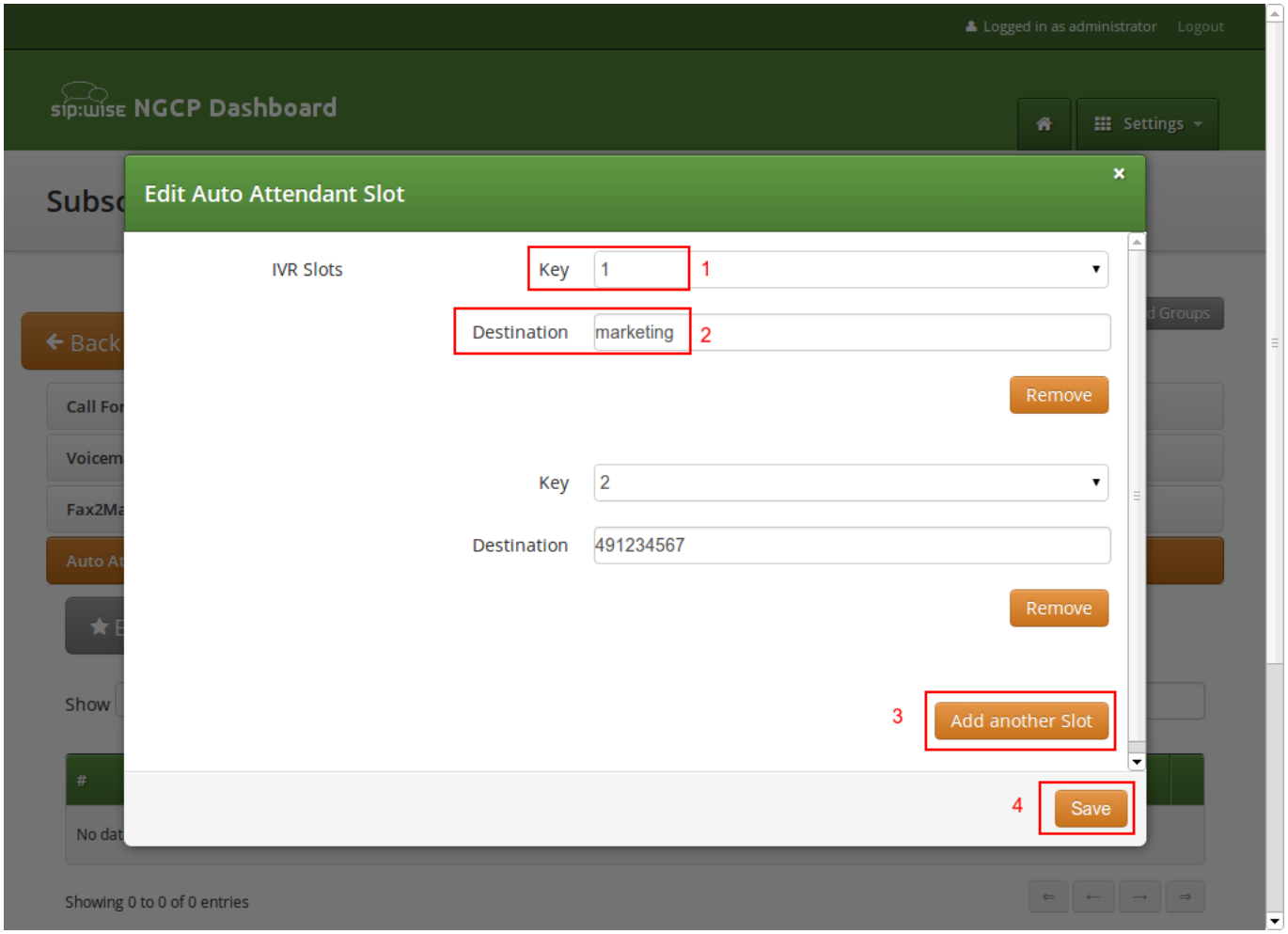


Figure 47: Define the Auto Attendant Slots

Activating the Auto Attendant

Once the Sound Set and the Slots are configured, activate the Auto Attendant by setting a Call Forward to Auto Attendant.

To do so, open the *Call Forwards* section in the *Subscriber Preferences* view and press *Edit* on the Call Forward type (e.g. *Call Forward Unconditional* if you want to redirect callers unconditionally to the Auto Attendant).

Select *Auto Attendant* and click *Save* to activate the Auto Attendant.

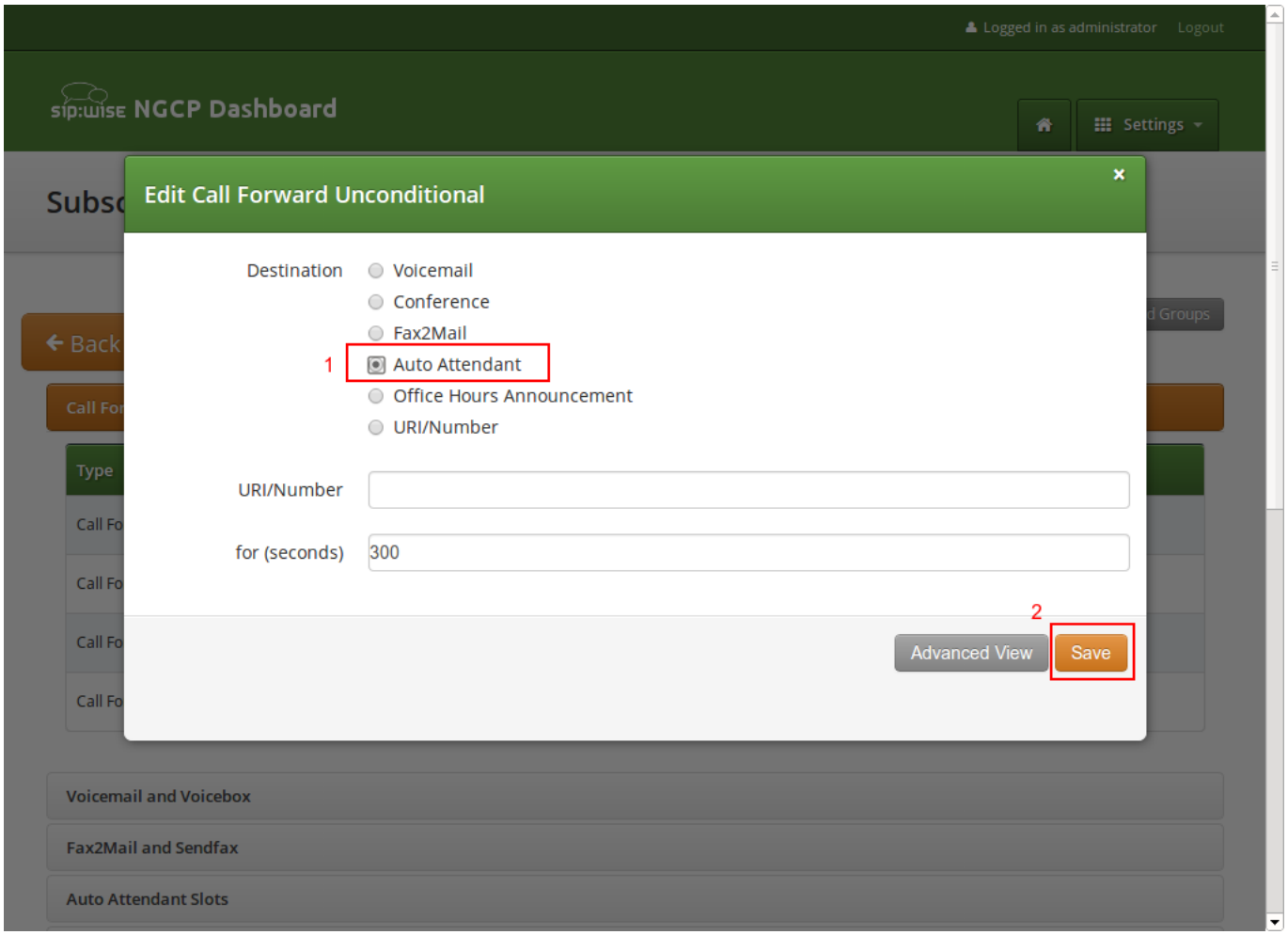


Figure 48: Set a Call Forward to Auto Attendant

Tip

As with any other Call Forward, you can define more complex forwarding rules in the *Advanced View* to only forward the call to the Auto Attendant during specific time periods, or as a fallback if no one picks up the office number.

A.5 Device Auto-Provisioning Security

A.5.1 Server Certificate Authentication

The Cisco SPA phones can connect to the provisioning interface of the PBX via HTTP and HTTPS. When perform secure provisioning over HTTPS, the phones validate the server certificate to check if its a legitimate Cisco provisioning server. To pass this check, the provisioning interface must provide a certificate signed by Cisco for that exact purpose.

The following steps describe how to obtain such a certificate.

First, a new SSL key needs to be generated:

```
$ openssl genrsa -out provisioning.key 2048
```

```
Generating RSA private key, 2048 bit long modulus
...+++
.....................................................................+++
e is 65537 (0x10001)
```

Next, a certificate signing request needs to be generated as follows. Provide your company details.

Important

The **Common Name (e.g. server FQDN or YOUR name)** field is crucial here. Provide an FQDN which the phones will later use via DNS to connect to the provisioning interface, for example *pbx.example.org*. Cisco does **NOT** support wild-card certificates.

Important

Leave the password empty when asked for it (press Enter without entering anything).

```
$ openssl req -new -key provisioning.key -out provisioning.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:AT
State or Province Name (full name) [Some-State]:Vienna
Locality Name (eg, city) []:Vienna
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Sipwise GmbH
Organizational Unit Name (eg, section) []:Operations
Common Name (e.g. server FQDN or YOUR name) []:pbx.example.org
Email Address []:office@sipwise.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Finally, compress the `provisioning.csr` file via ZIP and send it to our Cisco sales representative. If in doubt, you can try to send it directly to `ciscosb-certadmin@cisco.com` asking them to sign it.

Important

Only send the CSR file. **Do NOT send the key file, as this is your private key!**

Important

Ask for both the signed certificate AND a so-called *combinedca.crt* which is needed to perform client authentication via SSL. Otherwise you can not restrict access to Cisco SPAs only.

You will receive a signed CRT file, which Sipwise can use to configure the PBX provisioning interface.

A.5.2 Client Certificate Authentication

If a client connects via HTTPS, the server also checks for the client certificate in order to validate that the device requesting the configuration is indeed a legitimate Cisco phone, and not a fraudulent user with a browser trying to fetch user credentials.

A.6 Device Bootstrap and Resync Workflows

The IP phones supported by the PBX need to initially be configured to fetch their configuration from the system. Since the phones have no initial information about the system and its provisioning URL, they need to be boot-strapped. Furthermore, changes for a specific device might have to be pushed to the device immediately instead of waiting for it to re-fetch the configuration automatically.

The following chapters describe the work-flows how this is accomplished without having the customer directly accessing the phone.

A.6.1 Cisco SPA Device Bootstrap

Initial Bootstrapping

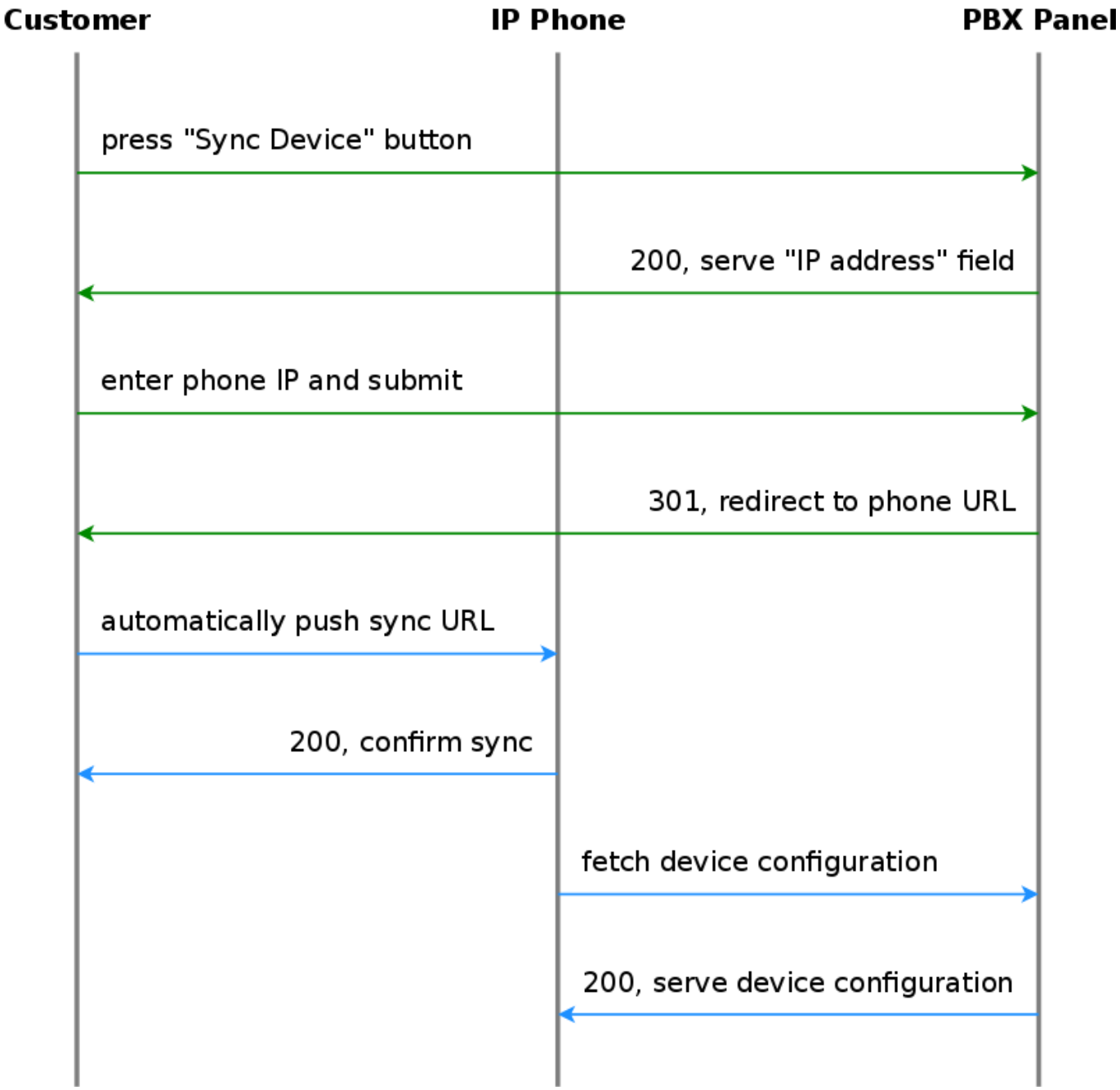


Figure 49: Initially bootstrap a PBX device

Subsequent Device Resyncs

If one of the subscribers configured on a PBX device is registered via SIP, the system can trigger a re-sync of the phone directly via SIP without having the customer enter the IP of the phone again. This is accomplished by sending a special `NOTIFY` message

to the subscriber:

```
NOTIFY sip:subscriber@domain SIP/2.0
To: <sip:subscriber@domain>
From: <sip:subscriber@domain>;tag=some-random-tag
Call-ID: some-random-call-id
CSeq: 1 NOTIFY
Subscription-State: active
Event: check-sync
Content-Length: 0
```

In order to prevent unauthorized re-syncs, the IP phone challenges the request with its own SIP credentials, so the NOTIFY is sent twice, once without authentication, and the second time with the subscriber's own SIP credentials.

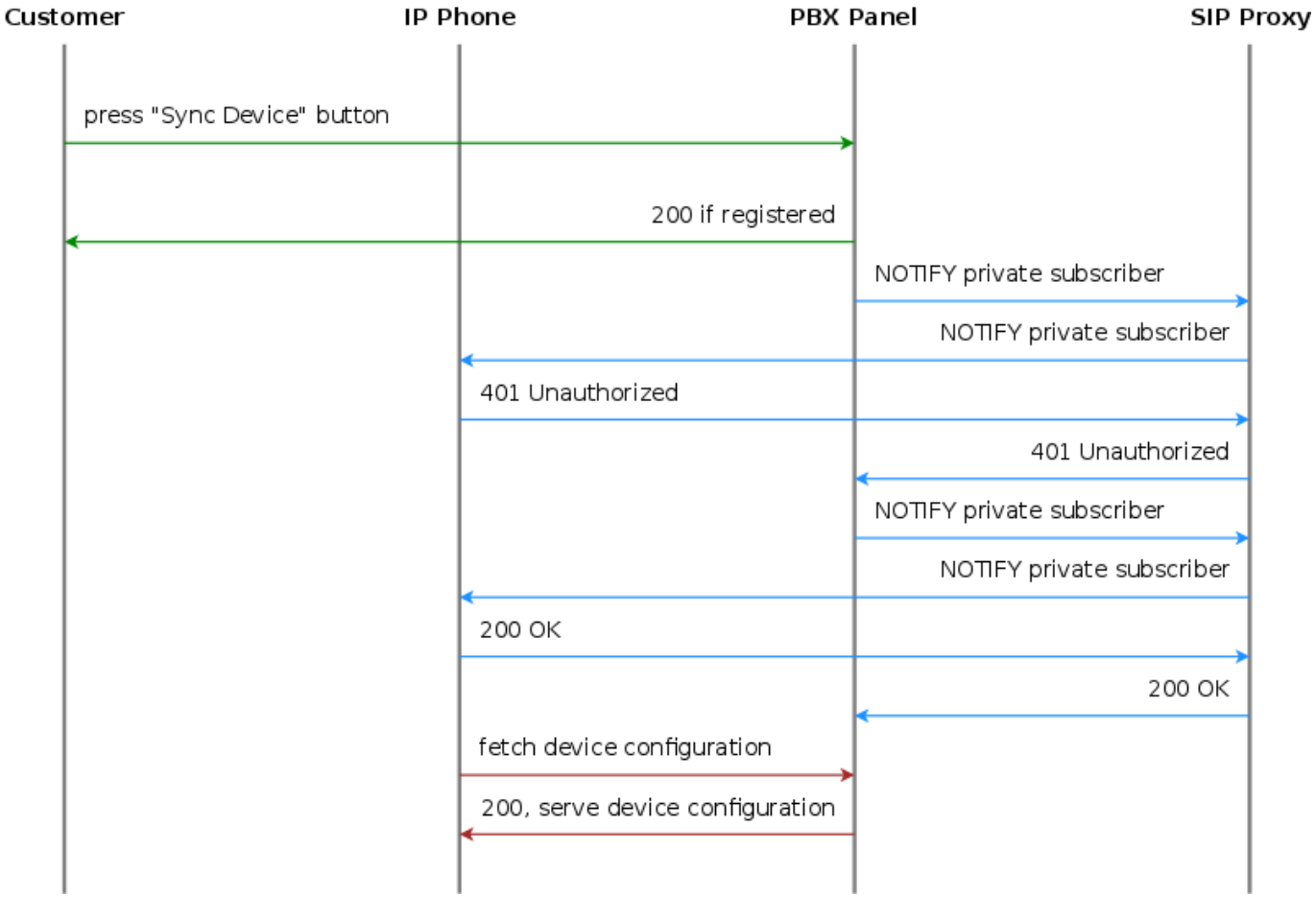


Figure 50: Resync a registered PBX device

A.6.2 Panasonic Device Bootstrap

Initial Bootstrapping

Panasonic provides a zero-touch provisioning mechanism in their firmwares, which causes the factory-reset phones to connect to a Panasonic web service at *https://provisioning.e-connecting.net* to check if a custom provisioning URL is configured for the MAC address of the phone. If an association between the MAC and a provisioning URL is found, the web service redirects the phone to the provisioning URL, where the phone connects to in order to obtain the configuration file.

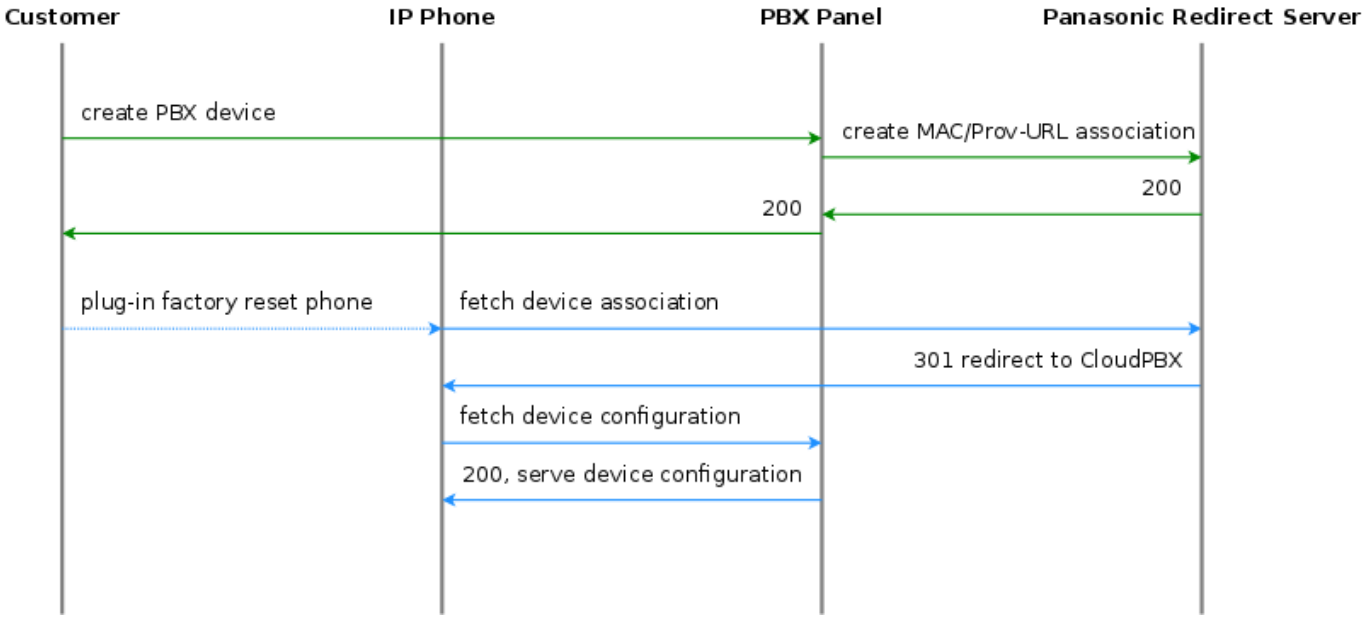


Figure 51: Initially bootstrap a Panasonic phone

The CloudPBX module ensures that when an end customer creates a Panasonic device, the MAC address is automatically provisioned on the Panasonic web service via an API call, so the customer's phone can use the correct provisioning URL to connect to the auto-provisioning server of the CloudPBX.

As a result, no customer interaction is required to bootstrap Panasonic phones, other than just creating the phone with the proper MAC on the CloudPBX web interface.

Factory Reset

For already provisioned phones, the end customer might need to perform a factory reset:

- Press *Settings* or *Setup*
- Enter *#136*
- Select *Factory Setting* and press *Enter*
- Select *Yes* and press *Enter*

- Select *Yes* and press *Enter*

The default username for factory-reset phones is *admin* with password *adminpass*.

Subsequent Device Resyncs

The same procedure as with Cisco SPA phones applies, once a subscriber configured on the phone is registered.

A.6.3 Yealink Device Bootstrap

Initial Bootstrapping

Yealink provides a zero-touch provisioning mechanism in their firmwares, which causes the factory-reset phones to connect to a Yealink web service at *https://rps.yealink.com* to check if a custom provisioning URL is configured for the MAC address of the phone. If an association between the MAC and a provisioning URL is found, the web service redirects the phone to the provisioning URL, where the phone connects to in order to obtain the configuration file.

If both Cisco SPA and Yealink phones are used, an issue with the Cisco-signed server certificate configured on the provisioning port (1444 by default) of the CloudPBX provisioning server arises. Yealink phones by default only connect to trusted server certificates, and the Cisco CA certificate used to sign the server certificate is not trusted by Yealink. Therefore, a two-step approach is used to disable the trusted check via a plain insecure http port (1445 by default) first, where only device-generic config options are served. No user credentials are provided in this case, because no SSL client authentication can be performed. The generic configuration disables the trusted check, and at the same time changes the provisioning URL to the secure port, where the Yealink phone is now able to connect to.

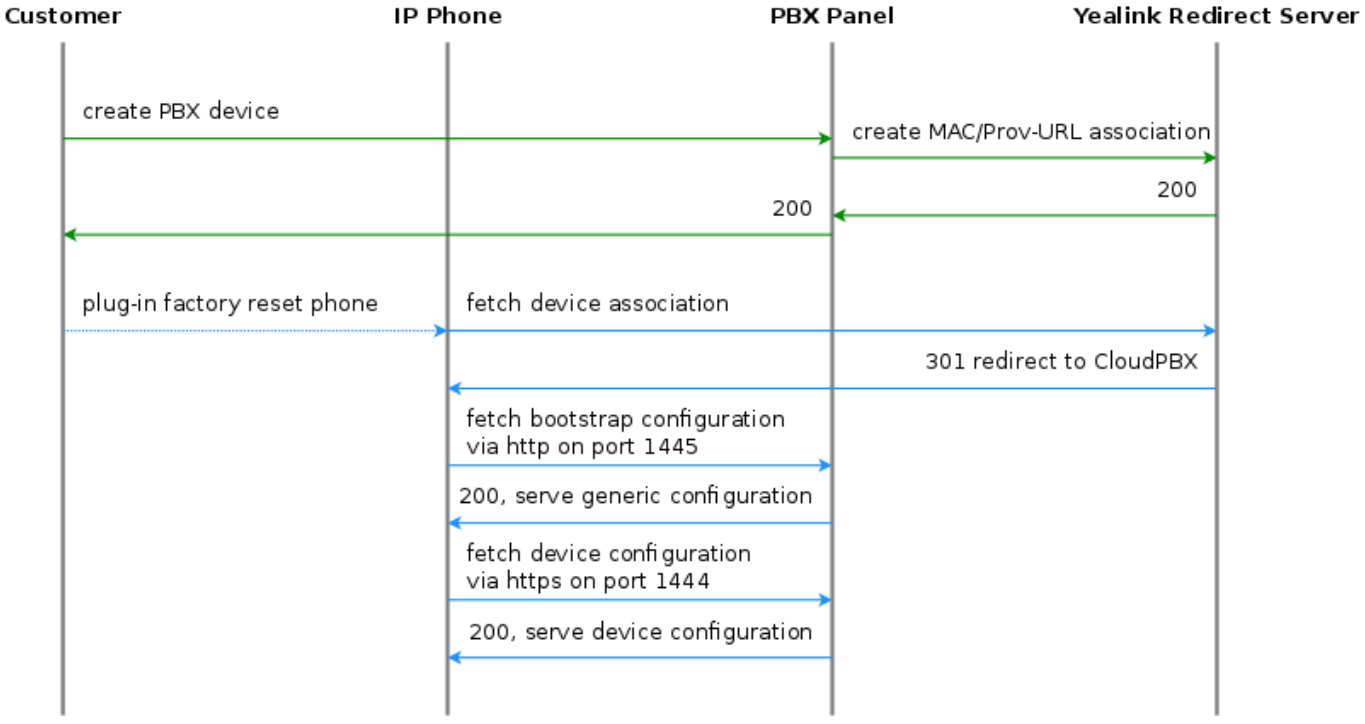


Figure 52: Initially bootstrap a Yealink phone

The CloudPBX module ensures that when an end customer creates a Yealink device, the MAC address is automatically provisioned on the Yealink web service via an API call, so the customer's phone can use the correct insecure bootstrap provisioning URL to connect to the auto-provisioning server of the CloudPBX for the generic configuration, which in turn provides the information on where to connect to for the secure, full configuration.

As a result, no customer interaction is required to bootstrap Yealink phones, other than just creating the phone with the proper MAC on the CloudPBX web interface.

Factory Enable Yealink Auto-Provisioning

Older Yealink firmwares don't automatically connect to the Yealink auto-provisioning server on initial boot, so it needs to be enabled manually by the end customer.

- Log in to *http://phone-ip/servlet?p=hidden&q=load* using *admin* and *admin* as user/password when prompted
- Change *Redirect Active* to *Enabled*
- Press *Confirm* and power-cycle phone

Subsequent Device Resyncs

The same procedure as with Cisco SPA phones applies, once a subscriber configured on the phone is registered.

B Sipwise Clients and Apps

The sip:carrier comes with two optional and commercial Unified Communication Clients for full end-to-end integration of voice, video, chat and presence features. On one hand, there is the sip:pone Desktop client for Microsoft Windows, Apple OSX and Linux. On the other hand, Sipwise provides the sip:phone Mobile App for Apple iOS and Android.

Both clients are fully brand-able to the customer's corporate identity. The clients are not part of the standard delivery and need to be licensed separately. The mobile client does not yet support the full range of features.

B.1 sip:phone Mobile App

The sip:phone Mobile App is a mobile client for iOS and Android and supports voice calls via SIP, as well as presence and instant messaging via XMPP. The following chapters describe the steps needed to integrate it into the sip:carrier.

B.1.1 Zero Config Launcher

Part of the mobile apps is a mechanism to sign up to the service via a 3rd party web site, which is initiated on the login screen and rendered within the app. During the sign-up process, the 3rd party service is supposed to create a new account and/or subscriber on the sip:carrier (e.g. automatically via the API) and provide the end user with the access credentials.

In order to minimize the end customer steps to log in using these credentials (especially ruling out the need to manually enter them), the mobile apps come with a zero config mechanism, which allows to deliver the access credentials via a side channel (e.g. Email, SMS) and packed into a URL, which the user just has to click, and which automatically launches the app with the correct credentials. The following picture shows the overall work flow.

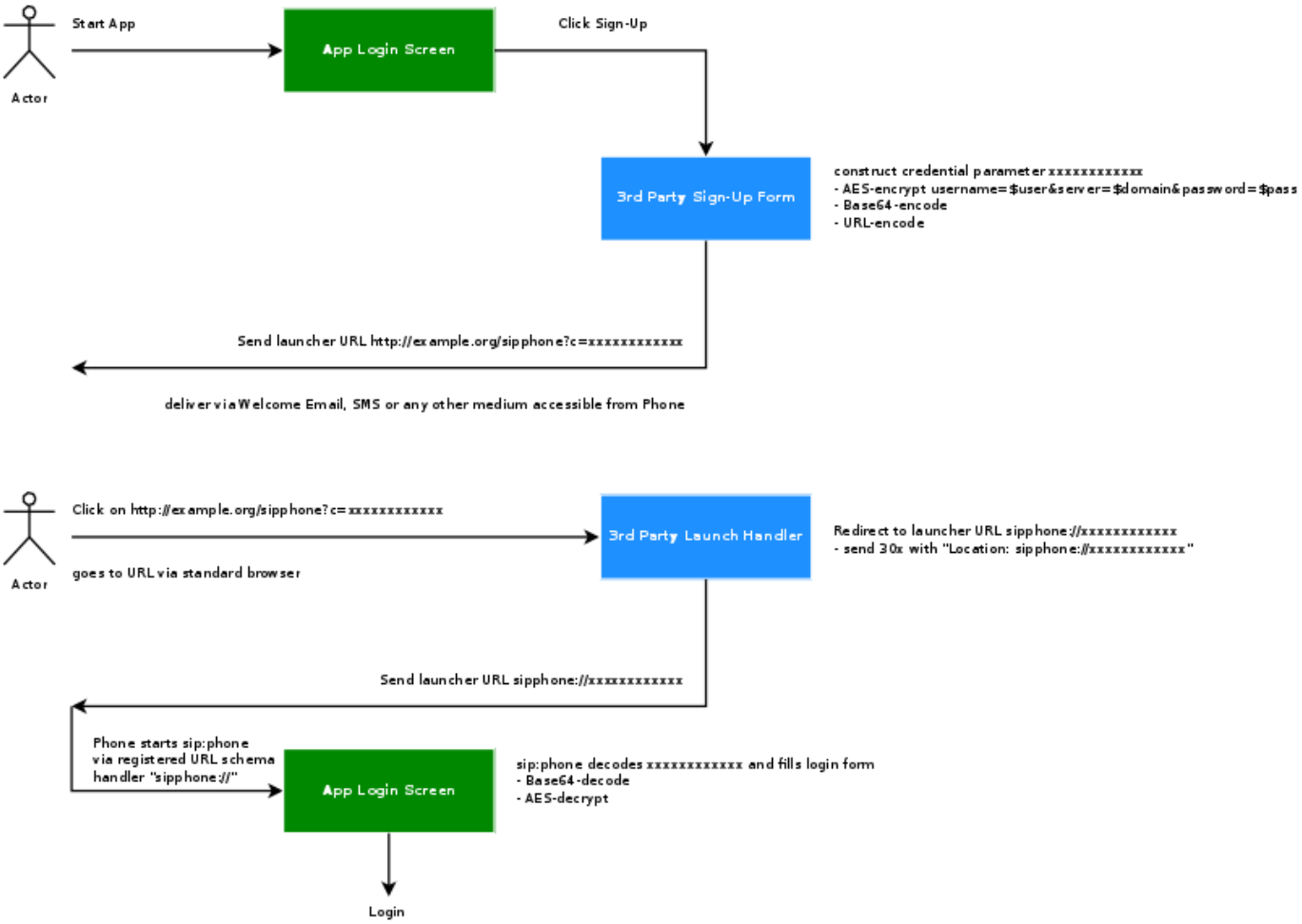


Figure 53: Provisioning Push Workflow

There are two components provided by a 3rd party system, which are not part of the sip:carrier. One is the *3rd Party Sign-Up Form*, and the other is the *3rd Party Launch Handler*. The purpose of these components is to make the end customer to open a link with the access credentials via the sip:phone app.

3rd Party Sign-Up Form

The 3rd Party Sign-Up Form is a web site the app shows to the end user when he taps the sign-up link on the *Login Screen* of the app. There, the end customer usually provides his contact details like name, address, phone number and/or email address etc. After validation, this web site creates the account and/or subscriber on the sip:carrier via the API.

After successfully creating the account and/or subscriber, this site needs to construct a specially crafted URL, which is sent back to the end customer via a side channel. Ideally, this channel would be SMS if you want to verify the end user's mobile number, or an email if you want to verify her email address.

The sip:phone app registers a URL schema handler for URLs starting with `sipphone://`. If you start such a link, the app performs a Base64 decoding of the string right after the `sipphone://` schema string, then decrypts the resulting binary string via AES using keys defined during the branding step. The resulting string is supposed to be

username=$user&server=$domain&password=$password.

Therefore, the *3rd Party Sign-Up Form* needs to construct this string using the credentials defined while creating the subscriber via the sip:carrier API, then encrypt it via AES, and finally perform a Base64 encoding of the result.

Note

Up until and including version mr3.8.13 of the sip:carrier, the SIP login credentials are used here. Future versions will connect to the REST interface of the sip:carrier using the web credentials first and fetch the SIP credentials along with other settings from there.

An example code snipped in Perl to properly encode such a string is outlined here. The AES key and initialization vector ($key and $iv) are the standard values of the sip:phone app and should work, if you haven't specified other values during the branding process.

```
#!/usr/bin/perl -w
use strict;
use Crypt::Rijndael;
use MIME::Base64;
use URI::Escape;

my $key = 'iBmTdavJ8joPW3HO';
my $iv = 'tww21lQe6cmywrp3';

my $plain = do { local $/; <> };
# pkcs#5 padding to 16 bytes blocksize
my $pad = 16 - (length $plain) % 16;
$plain .= pack('C', $pad) x $pad;

my $cipher = Crypt::Rijndael->new(
        $key,
        Crypt::Rijndael::MODE_CBC()
);
$cipher->set_iv($iv);
my $crypted = $cipher->encrypt($plain);
# store b64-encoded string and print to STDOUT
my $b64 = encode_base64($crypted, '');
print $b64, "\n";
# print to STDOUT using URL escaping also
print uri_escape($b64), "\n";
```

This snippet takes a string from STDIN, encrypts it via AES, encodes it via Base64 and prints the result on STDOUT. It also prints a second line with the same string, but this time URL escaped. To test it, you would run it as follows on a shell, granted it's stored at `/path/to/encrypt.pl`.

```
echo -n 'username=testuser&server=example.org&password=testpass' \
  | /path/to/encrypt.pl
```

This command would result in the output strings `CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI/Wv/VaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg==` and `CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D`. The sip:phone can use the former string to automatically fill in the login form of the Login Screen if started via a Link like `sipphone://CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI/Wv/VaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg==`.

Here is the same code in PHP.

```
#!/usr/bin/php
<?php
$key = "iBmTdavJ8joPW3HO";
$iv = "tww21lQe6cmywrp3";

$clear = fgets(STDIN);
$cipher = fnEncrypt($clear, $key, $iv);

echo $cipher, "\n";
echo urlencode($cipher), "\n";

function fnEncrypt($clear, $key, $iv) {
        $pad = 16 - strlen($clear) % 16;
        $clear .= str_repeat(pack('C', $pad), $pad);
        return rtrim(base64_encode(mcrypt_encrypt(
                MCRYPT_RIJNDAEL_128, $key, $clear,
                MCRYPT_MODE_CBC, $iv)), "\0");
}
?>
```

Similar to the perl version, you can call it like this:

```
echo -n 'username=testuser&server=example.org&password=testpass' \
  | /path/to/encrypt.php
```

However, a URL with the `sipphone://` schema is not displayed as a link in SMS or Email clients and thus can not be clicked by the end customer, so you need to make a detour via a normal `http://` URL. To do so, you need a *3rd Party Launch Handler* to trick the phone to open such a link.

This means that the *3rd Party Sign-Up Form* needs to return a link with an URL pointing to the *3rd Party Launch Handler* and pass the URL escaped string gathered above to the client via SMS or Email. Since it is a standard `http://` link, it is click-able on the phone and can be launched from virtually any client (SMS, Email etc.) which properly renders an HTML link.

A possible SMS sent to the end customer (via the phone number entered in the sign-up from) could therefore look as follows (trying to stay below 140 chars).

```
http://example.org/p?c=CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI
%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D to launch sipphone
```

An HTML Email could look like this:

```
Welcome to Example.org,
<a href="http://www.example.org/sipphone?c=CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI
%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D">
click here
</a> to log in.
```

That way, you can on one hand verify the contact details of the user, and on the other hand send her the login credentials in a secure manner.

3rd Party Launch Handler

The URL `http://www.example.org/sipphone` mentioned above can be any simple script, and its sole purpose is to send back a `301 Moved Permanently` or `302 Moved Temporarily` with a `Location:sipphone://xxxxxxxxxxxx` header to tell the phone to open this link via the sip:phone app. The *xxxxxxxxxxxx* is actually the plain (non-URL-escaped) string generated by the script above.

An example CGI script performing this task follows.

```
#!/usr/bin/perl -w
use strict;
use CGI;

my $q = CGI->new;
my $c = $q->param('c');
print CGI::redirect("sipphone://$c");
```

The script simply takes the URL parameter `c` from the URL `http://www.example.org/sipphone?c=CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D` crafted above and puts its content into a `Location` header using the `sipphone://` schema, and finally sends a `301 Moved Permanently` back to the phone.

The phone follows the redirect by opening the URL using the sip:phone app, which in turn decrypts the content and fills in the login form.

Note

Future versions of the sip:carrier will ship with this launch handler integrated in the system. Up until and including version mr3.8.13, this script needs to be installed on any webserver manually.

B.1.2 Mobile Push Notification

The sip:phone provides *mobile push* functionality to remotely start the app via the Google GCM or Apple APNS notification systems on inbound calls, in case the app is not registered.

Caution

Although stopping the App on the phone and letting it wake up via the push notification system safes some battery power on the phone, the whole push notification concept is a best effort framework for both iOS and Android provided by Apple and Google, respectively, and is therefore not 100% reliable.

Architecture

If the *mobile push* functionality is enabled, the call-flow looks as follows if there are no devices registered for a subscriber.

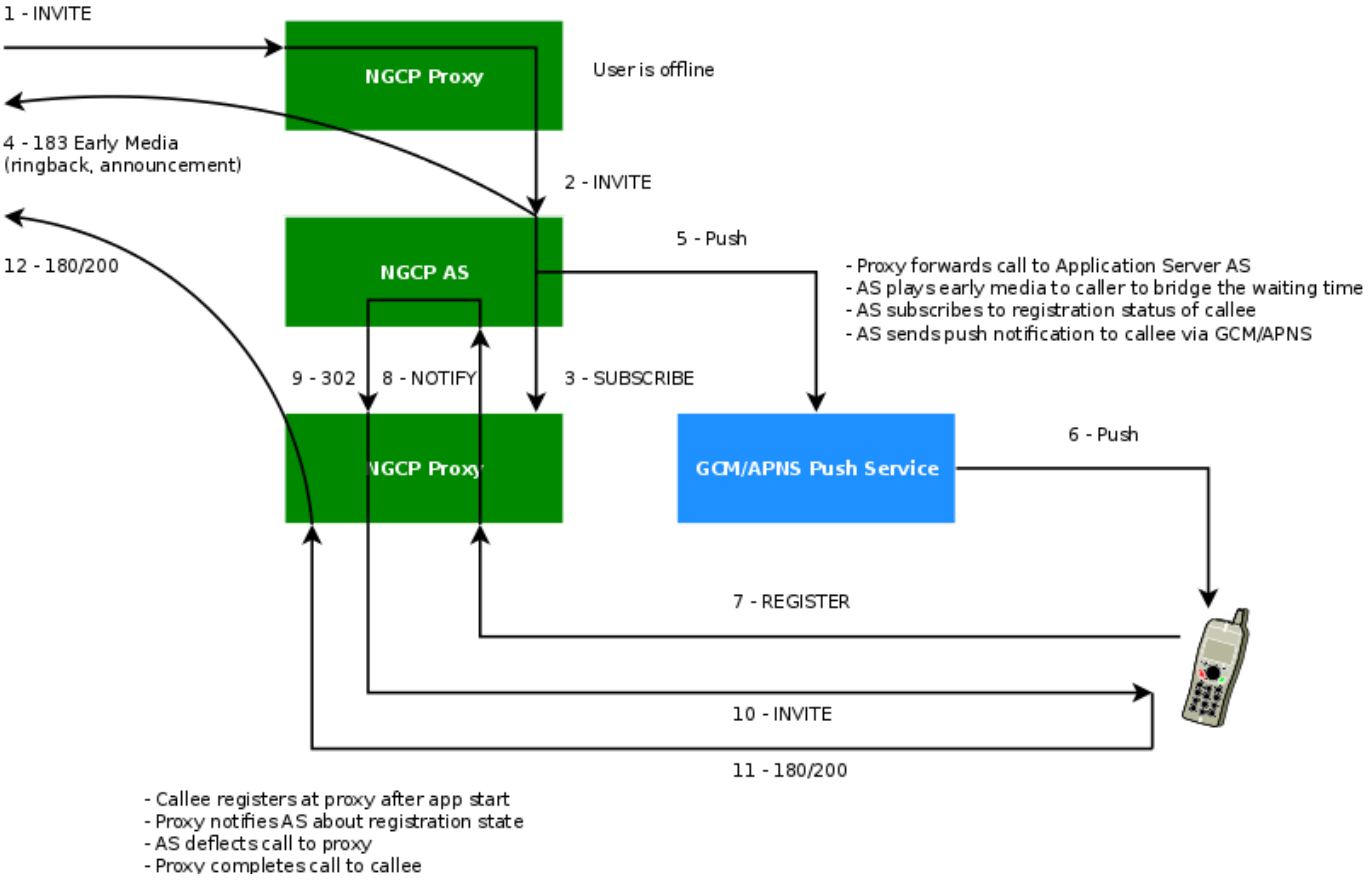


Figure 54: Mobile Push Workflow

1. Caller sends INVITE to proxy
2. Callee is offline, proxy forwards call to AS
3. AS subscribes to registration state of callee at proxy
4. AS plays early media to caller for feedback, as process might take a while
5. AS sends push request to GCM/APNS service
6. GCM/APNS service delivers request to callee

7. Callee accepts request and confirms app start (unattended on Android), registers at proxy
8. Proxy sends registration notification to AS
9. AS deflects call back to proxy
10. Proxy sends INVITE to callee
11. Callee accepts call
12. Response is sent back to caller, call setup completed

In case of a timeout (no registration notification within a certain time) at the application server, the call is rejected with an error.

Configuring the Push Daemon

The push daemon needs your specific keys and/or certificates obtained from Apple and Google, respectively.

Please read the official GCM Getting Started Guide for Android on how to obtain a push notification key from Google for GCM.

For instructions how to generate Apple push notification certificates and keys, please read the official Provisioning Procedures from Apple.

The final configuration in your */etc/ngcp-config/config.yml* should look as follows.

```
pushd:
  apns:
    certificate: '/etc/ssl/private/your.phone.push.dev.pem'
    enable: 'yes'
    endpoint: gateway.push.apple.com
    feedback_endpoint: feedback.push.apple.com
    feedback_interval: 3600
    key: ''
    socket_timeout: 0
  enable: 'yes'
  gcm:
    enable: 'yes'
    key: 'xxxxxxxxxxxxxxxxxxxxxxxx-yyyyyyyyyyyyyy'
  port: 45060
  processes: 4
  ssl: 'yes'
```

Once configured, execute `ngcpcfg apply` to confirm your changes.

C NGCP configs overview

C.1 config.yml overview

Config.yml is the main configuration YAML file used by Sipwise NGCP. After every changes it need to run the command ngcpcfg apply to apply changes (followed by ngcpcfg push in the PRO version to apply changes to sp2). The following is a brief description of the main variables contained into /etc/ngcp-config/config.yml file.

C.1.1 asterisk

The following is the asterisk section:

```
asterisk:
  log:
    facility: local6
  rtp:
    maxport: 20000
    minport: 10000
  sip:
    bindport: 5070
    dtmfmode: rfc2833
  voicemail:
    enable: 'no'
    fromstring: 'Voicemail server'
    greeting:
      busy_custom_greeting: '/home/user/file_no_extension'
      busy_overwrite_default: 'no'
      busy_overwrite_subscriber: 'no'
      unavail_custom_greeting: '/home/user/file_no_extension'
      unavail_overwrite_default: 'no'
      unavail_overwrite_subscriber: 'no'
    mailbody: 'You have received a new message from ${VM_CALLERID} in voicebox ${VM_MAILBOX ↩
        } on ${VM_DATE}.'
    mailsubject: '[Voicebox] New message ${VM_MSGNUM} in voicebox ${VM_MAILBOX}'
    max_msg_length: 180
    maxgreet: 60
    maxmsg: 30
    maxsilence: 0
    min_msg_length: 3
    normalize_match: '^00|\+([1-9][0-9]+)$'
    normalize_replace: '$1'
    serveremail: voicebox@sip.sipwise.com
```

- log.facility: rsyslog facility for asterisk log, defined in /etc/asterisk/logger.conf.
- rtp.maxport: RTP maximum port used by asterisk.

- rtp.minport: RTP minimun port used by asterisk.
- sip.bindport: SIP asterisk internal bindport.
- voicemail.greetings.*: set the audio file path for voicemail custom unavailable/busy greetings
- voicemail.mailbody: Mail body for incoming voicemail.
- voicemail.mailsubject: Mail subject for incoming voicemail.
- voicemail.max_msg_length: Sets the maximum length of a voicemail message, in seconds.
- voicemail.maxgreet: Sets the maximum length of voicemail greetings, in seconds.
- voicemail.maxmsg: Sets the maximum number of messages that may be kept in any voicemail folder.
- voicemail.min_msg_length: Sets the minimun length of a voicemail message, in seconds.
- voicemail.maxsilence: Maxsilence defines how long Asterisk will wait for a contiguous period of silence before terminating an incoming call to voice mail. The default value is 0, which means the silence detector is disabled and the wait time is infinite.
- voicemail.serveremail: Provides the email address from which voicemail notifications should be sent.
- voicemail.normalize_match: Regular expression to match the From number for calls to voicebox.
- voicemail.normalize_replace: Replacement string to return, in order to match an existing voicebox.

C.1.2 backuptools

The following is the backup tools section:

```
backuptools:
  cdrexport_backup:
    enable: 'no'
  etc_backup:
    enable: 'no'
  mail:
    address: noc@company.org
    error_subject: '[ngcp-backup] Problems detected during daily backup'
    log_subject: '[ngcp-backup] Daily backup report'
    send_errors: 'no'
    send_log: 'no'
  mysql_backup:
    enable: 'no'
    exclude_dbs: 'syslog sipstats information_schema'
  rotate_days: 7
  storage_dir: '/var/backup/ngcp_backup'
  temp_backup_dir: '/tmp/ngcp_backup'
```

- backuptools.cdrexport_backup.enable: Enable backup of cdrexport (.csv) directory.

- backuptools.etc_backup.enable: Enable backup of /etc/* directory.
- backuptools.mail.address: Destination email address for backup emails.
- backuptools.mail.error_subject: Subject for error emails.
- backuptools.mail.log_subjetc: Subject for daily backup report.
- backuptools.mail.send_error: Send daily backup error report.
- backuptools.mail.send_log: Send daily backup log report.
- backuptools.mysql_backup.enable: Enable daily mysql backup.
- backuptools.mysql_backup.exclude_dbs: exclude mysql databases from backup.
- backuptools.rotate_days: Number of backups to keep stored.
- backuptools.storage_dir: Storage directory of backups.
- backuptools.temp_backup_dir: Temporary storage directory of backups.

C.1.3 bootenv

The following is the bootenv section:

```
bootenv:
  dhcp:
    boot: '/srv/tftp/pxelinux.0'
    enable: 'yes'
    end: 192.168.1.199
    expire: 12h
    start: 192.168.1.101
  nfs:
    enable: 'yes'
    link: '/srv/deployment'
    root: '/mnt/glusterfs/mgmt-share/deployment'
  ro_port: 9998
  rw_port: 9999
  tftp:
    enable: 'yes'
    root: '/srv/tftp'
```

- bootenv.dhcp.enable: enable dnsmasq DHCP server
- bootenv.dhcp.boot: PXE image boot location
- bootenv.dhcp.start: first IP of DHCP scope
- bootenv.dhcp.end: last IP of DHCP scope
- bootenv.dhcp.expire: DHCP leasing expiration

- bootenv.nfs.enable: enable NFS server for PXE boot
- bootenv.nfs.link: synlink on *bootenv.nfs.root* for PXE access
- bootenv.nfs.root: root folder for NFS server
- bootenv.ro_port: HTTP port for read-only access to Approx cache
- bootenv.rw_port: HTTP port for read-write access to Approx cache
- bootenv.tftp.enable: enable tftp server for PXE boot
- bootenv.tftp.root: root folder for tftp server

C.1.4 cdrexport

The following is the cdr export section:

```
cdrexport:
  daily_folder: 'yes'
  export_failed: 'no'
  export_incoming: 'no'
  exportpath: '/home/jail/home/cdrexport'
  full_names: 'yes'
  monthly_folder: 'yes'
```

- cdrexport.daily_folder:: Set *yes* if you want to create a daily folder for CDRs under the configured path.
- cdrexport.export_failed: Export CDR for failed calls.
- cdrexport.export_incoming: Export CDR for incoming calls.
- cdrexport.exportpath: The path to store CDRs in .csv format.
- cdrexport.full_names: Use full namen for CDRs instead of short ones.
- cdrexport.monthly_folder: Set *yes* if you want to create a monthly folder (ex. 201301 for January 2013) for CDRs under configured path.

C.1.5 checktools

The following is the check tools section:

```
checktools:
  collcheck:
    cpuidle: 0.1
    dfused: 0.9
    eximmaxqueue: 15
    loadlong: 2
    loadmedium: 2
```

```
    loadshort: 3
    maxage: 600
    memused: 0.7
    siptimeout: 15
    swapfree: 0.5
  asr_nsr_statistics: 1
  exim_check_enable: 0
  force: 0
  kamailio_check_dialog_active_enable: 1
  kamailio_check_dialog_early_enable: 1
  kamailio_check_dialog_incoming_enable: 1
  kamailio_check_dialog_local_enable: 1
  kamailio_check_dialog_outgoing_enable: 1
  kamailio_check_dialog_relay_enable: 1
  kamailio_check_usrloc_regdevices_enable: 1
  kamailio_check_usrloc_regusers_enable: 1
  mpt_check_enable: 1
  mysql_check_enable: 1
  mysql_check_replication: 1
  oss_check_provisioned_subscribers_enable: 1
  sip_check_enable: 1
  sipstats_check_num_packets: 1
  sipstats_check_num_packets_perday: 1
  sipstats_check_partition_size: 1
  snmpd:
    communities:
      public:
        - localhost
```

- checktools.collcheck.cpuidle: Sets the minimum value for CPU usage (0.1 means 10%).
- checktools.collcheck.dfused: Sets the maximun value for DISK usage (0.9 means 90%).
- checktools.collcheck.loadlong/loadlong/loadshort: Max values for load (long, short, medium term).
- checktools.collcheck.maxage: Max age in seconds.
- checktools.collcheck.memused: Sets the maximun value for MEM usage (0.7 means 70%).
- checktools.collcheck.siptimeout: Max timeout for sip options.
- checktools.collcheck.swapfree: Sets the minimun value for SWAP free (0.5 means 50%).
- checktools.exim_check_enable: Exim queue check plugin for collectd.
- checktools.asr_nsr_statistics: enable/Disable ASR/NSR statistics.
- checktools.force: Perform checks even if not active in /etc/motd.
- checktools.kamailio_check_dialog_*/kamailio_check_usrloc_*: Enable/Disable SNMP collective check pluglin for Kamailio.

- checktools.mpt_check_enable: MPT raid SNMP check plugin.
- checktools.mysql_check_enable: MySQL SNMP check plugin.
- checktools.mysql_check_replication: MySQL replication check.
- checktools.oss_check_provisioned_subscribers_enable: OSS provisioned subscribers count plugin.
- checktools.sip_check_enable/sipstats_check_*: Enable/Disable SIP check plugins.
- checktools.snmpd.communities: Sets the snmp community and sources (separated by comma , - ex. source: 127.0.0.1, 10.10.10.2, 10.10.10.3).

C.1.6 cleanuptools

The following is the cleanup tools section:

```
cleanuptools:
  acc_cleanup_days: 90
  archive_targetdir: '/var/backups/cdr'
  binlog_days: 15
  cdr_archive_months: 12
  cdr_backup_months: 6
  cdr_backup_retro: 3
  compress: gzip
  sql_batch: 10000
  trash_cleanup_days: 30
```

- cleanuptools.acc_cleanup_days: Clean up ACC entry older then 90 days.
- cleanuptools.binlog_days: Expire MySQL binlogs after 15 days.
- cleanuptools.cdr_archive_months: How many months worth of records to keep in the table and not move into the monthly archive tables.
- cleanuptools.cdr_backup_months: How many months worth of records to keep in the table and not move into the monthly backup tables.
- cleanuptools.cdr_backup_retro: How many months to process for backups, going backwards in time. Using the example above, with this value set to "3", the months October, September and August would be backed up, while any older records would be left untouched.
- cleanuptools.sql_batch: How many records to process within a single statement.
- cleanuptools.trash_cleanup_days: Clean up acc_trash and acc_backup entry after 30 days.

C.1.7 database

The following is the database section:

```
database:
  bufferpoolsize: 24768M
```

- database.bufferpoolsize: Innodb_buffer_pool_size value in /etc/mysql/my.cnf

C.1.8 faxserver

The following is the fax server section:

```
faxserver:
  default_owner: 4312345
  failfax_recv_email: root@localhost
  failfax_send_email: failfax@ngcp.sipwise.local
  fax_gateways:
    - sip:127.0.0.1:5070
  hylafax:
    jobretry: 1
    start: 'yes'
  iaxmodem:
    start: 'yes'
  type: software
  mail_from: 'Sipwise NGCP FaxServer <voipfax@ngcp.sipwise.local>'
  webfax_user: ngcpwebfax
```

- faxserver.failfax_recv_email: A recipient of a failed "fax receive".
- faxserver.failfax_send_email: A recipient of a failed "fax send".
- faxserver.fax_gateways: Set here the correct Patton gateway ip address and port (Available only with the hardware fax solution). Otherwise leave as it is.
- faxserver.hylafax.jobretry: How many times the hylafax faxserver should retry to send fax.
- faxserver.hylafax.start: Enable hylafax at startup.
- faxserver.iaxmodem.start: Enable iaxmodem at startup.
- faxserver.type: Type of faxserver solution. Accepted values are *software* or *hardware* (with Patton Gateway).
- faxserver.mail_from: Sets the e-mail From Header for incoming fax.
- faxserver.webfax_user: User used when sending fax from CSC web interface.

C.1.9 general

The following is the general section:

```
general:
  adminmail: adjust@example.org
  companyname: sipwise
  lang: en
```

- general.adminmail: Email address used by monit to send notifications to.
- general.lang: Sets sounds language (e.g: *de* for German)

C.1.10 haproxy

The following is the haproxy section:

```
haproxy:
  admin: 'no'
  admin_port: 8080
  admin_pwd: iKNPFuPFHMCHh9dsXgVg
  enable: 'no'
```

- haproxy.enable: enable haproxy

C.1.11 heartbeat

The following is the heartbeat section:

```
heartbeat:
  hb_watchdog:
    action_max: 5
    enable: 'yes'
    interval: 10
    transition_max: 10
  pingnodes:
    - 10.60.1.1
    - 192.168.3.4
```

- heartbeat.hb_watchdog.enable: Enable heartbeat watchdog in order to prevent and fix split brain scenario.
- heartbeat.hb_watchdog.action_max: Max errors before taking any action.
- heartbeat.hb_watchdog.interval: Interval in secs for the check.
- heartbeat.hb_watchdog.transition_max: Max checks in transition state.

- heartbeat.pingnodes: List of pingnodes for heartbeat. Minimun 2 entries, otherwise by default NGCP will set the default gateway and DNS servers as pingnodes.

C.1.12 intercept

The following is the legal intercept section:

```
intercept:
  captagent:
    port: 18090
    schema: http
  enabled: 'no'
```

- intercept.captagent.enable: Enable captagent for Lawful Interception (addiotional NGCP module).

C.1.13 kamailio

The following is the kamailio section:

```
kamailio:
  lb:
    debug: 'no'
    extra_sockets: ~
    max_forwards: 70
    nattest_exception_ips:
      - 1.2.3.4
      - 5.6.7.8
    pkg_mem: 16
    port: 5060
    security:
      dos_ban_enable: 'yes'
      dos_ban_time: 300
      dos_reqs_density_per_unit: 50
      dos_sampling_time_unit: 5
      dos_whitelisted_ips: ~
      dos_whitelisted_subnets: ~
      failed_auth_attempts: 3
      failed_auth_ban_enable: 'yes'
      failed_auth_ban_time: 3600
    shm_mem: 2012
    start: 'yes'
    strict_routing_safe: 'no'
    tcp_children: 8
    tcp_max_connections: 2048
    tls:
      enable: 'no'
```

```
    port: 5061
    sslcertfile: '/etc/kamailio/kamailio-selfsigned.pem'
    sslcertkeyfile: '/etc/kamailio/kamailio-selfsigned.key'
  udp_children: 8
  use_dns_cache: 'on'
proxy:
  allow_info_method: 'no'
  allow_peer_relay: 'no'
  allow_refer_method: 'no'
  authenticate_bye: 'no'
  cf_depth_limit: 10
  children: 8
  debug: 'no'
  default_expires: 3600
  enum_suffix: e164.arpa.
  filter_100rel_from_supported: 'yes'
  fritzbox:
    enable: 'no'
    prefixes:
      - 112
      - 110
      - 118[0-9]{2}
  foreign_domain_via_peer: 'no'
  ignore_auth_realm: 'no'
  keep_original_to: 'no'
  max_expires: 43200
  max_gw_lcr: 128
  max_registrations_per_subscriber: 5
  min_expires: 60
  nathelper_dbro: 'no'
  natping_interval: 30
  natping_processes: 7
  nonce_expire: 300
  pbx:
    hunt_display_indicator: '[h]'
  perform_peer_lcr: 0
  pkg_mem: 16
  port: 5062
  presence:
    enable: 'yes'
  proxy_lookup: 'no'
  set_ruri_to_peer_auth_realm: 'no'
  shm_mem: 2012
  start: 'yes'
  tcp_children: 4
  use_enum: 'no'
  usrloc_dbmode: 1
```

- kamailio.lb.debug: Enable intensive debug level.
- kamailio.lb.extra_sockets: Add here extra sockets for Load Balancer.
- kamailio.lb.max_forwards: Set the value for the Max Forwards SIP header for outgoing messages.
- kamailio.lb.nattest_exception_ips: List of IPs that don't need the NAT test.
- kamailio.lb.shm_mem: Shared memory used by Kamailio Load Balancer. The default value is auto generated by the system, depending on your system architecture.
- kamailio.lb.pkg_mem: PKG memory used by Kamailio Load Balancer. The default value is auto generated by the system, depending on your system architecture.
- kamailio.lb.security.dos_ban_enable: Enable/Disable DoS Ban.
- kamailio.lb.security.dos_ban_time: Sets the ban time.
- kamailio.lb.security.dos_reqs_density_per_unit:: Sets the requests density per unit (if we receive more then * lb.dos_reqs_density_per_u within dos_sampling_time_unit the user will be banned).
- kamailio.lb.security.dos_sampling_time_unit: Sets the DoS unit time.
- kamailio.lb.security.dos_whitelisted_ips: Write here the whitelisted IPs.
- kamailio.lb.security.failed_auth_attempts: Sets how many authentication attempts allowed before ban.
- kamailio.lb.security.failed_auth_ban_enable: Enable/Disable authentication ban.
- kamailio.lb.security.failed_auth_ban_time: Sets how long a user/IP has be banned.
- kamailio.lb.strict_routing_safe: Enable strict routing handle feature.
- kamailio.lb.tls.enable: Enable TLS socket.
- kamailio.lb.tls.port: Set TLS listening port.
- kamailio.lb.tls.sslcertificate: Path for the SSL certificate.
- kamailio.lb.tls.sslcertkeyfile: Path for the SSL key file.
- kamailio.proxy.allow_info_method: Allow INFO method.
- kamailio.proxy.allow_peer_relay: Allow peer relay. Call coming from a peer that doesn't matcha a local subscriber will try to go out again, matching the peering rules.
- kamailio.proxy.allow_refer_method: Allow REFER method. Enable it with caution.
- kamailio.proxy.authenticate_bye: Enable BYE authentication.
- kamailio.proxy.cf_depth_limit: CF loop detector. How many CF loops are allowed before drop the call.
- kamailio.proxy.debug: Enable intensive debug level.
- kamailio.proxy.default_expires: Default expires value in seconds for REGISTER messages.
- kamailio.proxy.foreign_domain_via_peer: Enable calls to foreign domains via peers.

- kamailio.proxy.shm_mem: Shared memory used by Kamailio Proxy. The default value is auto generated by the system, depending on your system architecture.
- kamailio.proxy.pkg_mem: PKG memory used by Kamailio Proxy. The default value is auto generated by the system, depending on your system architecture.
- kamailio.proxy.enum_suffix: Sets ENUM suffix - don't forget . (dot).
- kamailio.proxy.filter_100rel_from_supported: Enable filtering of *100rel* from Supported header, to disable PRACK.
- kamailio.proxy.fritzbox.enable: Enable detection for Fritzbox special numbers. Ex. Fritzbox add the AC prefix to emergency numbers.
- kamailio.proxy.fritzbox.prefixes: Specifies special prefixes to detect in order to remove the AC prefix added by Fritzbox.
- kamailio.proxy.ignore_auth_realm: Ignore SIP authentication realm.
- kamailio.proxy.keep_original_to: Not used now.
- kamailio.proxy.max_expires: Sets the maximum expires in seconds for registration.
- kamailio.proxy.max_gw_lcr: Defines the maximum number of gateways in lcr_gw table
- kamailio.proxy.max_registrations_per_subscriber: Sets the maximum registration per subscribers.
- kamailio.proxy.min_expires: Sets the minimum expires in seconds for registration.
- kamailio.proxy.natping_interval: Sets the NAT ping interval in seconds.
- kamailio.proxy.nathelper_dbro: Defaul is "no". This will be "yes" on CARRIER in order to activate the use of a read-only connection using LOCAL_URL
- kamailio.proxy.nonce_expire: Nonce expire time in seconds.
- kamailio.proxy.perform_peer_lcr: Enable/Disable Least Cost Routing based on peering fees.
- kamailio.proxy.port: SIP listening port.
- kamailio.proxy.presence.enable: Enable/disable presence feature
- kamailio.proxy.set_ruri_to_peer_auth_realm: Set R-URI using peer auth realm
- kamailio.proxy.use_enum: Enable/Disable ENUM feature.

C.1.14 mediator

The following is the mediator section:

```
mediator:
  interval: 10
```

- mediator.interval: Running interval of mediator.

C.1.15 nginx

The following is the nginx section:

```
nginx:
  status_port: 8081
  xcap_port: 1080
```

- nginx.status_port: Status port used by nginx server
- nginx.xcap_port: XCAP port used by nginx server

C.1.16 ntp

The following is the ntp server section:

```
ntp:
  servers:
    - 0.debian.pool.ntp.org
    - 1.debian.pool.ntp.org
    - 2.debian.pool.ntp.org
    - 3.debian.pool.ntp.org
```

- ntp.servers: Define your NTP server list.

C.1.17 ossbss

The following is the ossbss section:

```
ossbss:
  apache:
    autoprov:
      certfile: '/etc/ngcp-config/ssl/myserver.crt'
      certkeyfile: '/etc/ngcp-config/ssl/myserver.key'
      host: localhost
      port: 1444
      sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
      sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
      sslclientauth: 'yes'
    port: 2443
    proxyluport: 1080
    restapi:
      sslcertfile: '/etc/ngcp-panel/api_ssl/api_ca.crt'
      sslcertkeyfile: '/etc/ngcp-panel/api_ssl/api_ca.key'
    serveradmin: support@sipwise.com
    servername: "\"myserver\""
```

```
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
frontend: fcgi
htpasswd:
  -
    pass: '{SHA}w4zj3mxbmynIQ1jsUEjSkN2z2pk='
    user: ngcpsoap
logging:
  apache:
    acc:
      facility: daemon
      identity: oss
      level: info
    err:
      facility: local7
      level: info
  ossbss:
    facility: local0
    identity: provisioning
    level: DEBUG
  web:
    facility: local0
    level: DEBUG
provisioning:
  allow_ip_as_domain: 1
  allow_numeric_usernames: 0
  auto_allow_cli: 1
  carrier:
    account_distribution_function: roundrobin
    prov_distribution_function: roundrobin
  credit_warnings:
    -
      domain: example.com
      recipients:
        - nobody@example.com
      threshold: 1000
  faxpw_min_char: 0
  log_passwords: 0
  no_logline_truncate: 0
  pw_min_char: 6
  routing:
    ac_regex: '[1-9]\d{0,4}'
    cc_regex: '[1-9]\d{0,3}'
    sn_regex: '[1-9]\d+'
  tmpdir: '/tmp'
```

- ossbss.htpasswd: Sets the username and SHA hashed password for SOAP access. You can generate the password using the following command: htpasswd -nbs myuser mypassword.
- ossbss.provisioning.allow_ip_as_domain: Allow or not allow IP address as SIP domain (0 is not allowed).
- ossbss.provisioning.allow_numeric_usernames: Allow or not allow numeric SIP username (0 is not allowed).
- ossbss.provisioning.faxpw_min_char: Minimum number of characters for fax passwords.
- ossbss.provisioning.pw_min_char: Minimum number of characters for sip passwords.
- ossbss.provisioning.log_password: Enable logging of passwords.
- ossbss.provisioning.routing: Regexp for allowed AC (Area Code), CC (Country Code) and SN (Subscriber Number).

C.1.18 pbx (only with additional cloud PBX module installed)

The following is the PBX section:

```
pbx:
  bindport: 5085
  enable: 'no'
  highport: 55000
  lowport: 50001
  media_processor_threads: 10
  session_processor_threads: 10
  xmlrpcport: 8095
```

- pbx.enable: Enable Cloud PBX module.

C.1.19 prosody

The following is the prosody section:

```
prosody:
  ctrl_port: 5582
  log_level: info
```

- prosody.ctrl_port: XMPP server control port.
- prosody.log_level: Prosody loglevel.

C.1.20 pushd

The following is the pushd section:

```
pushd:
  apns:
    certificate: ''
    enable: 'no'
    endpoint: gateway.sandbox.push.apple.com
    feedback_endpoint: feedback.sandbox.push.apple.com
    feedback_interval: 3600
    key: ''
    socket_timeout: 0
  enable: 'no'
  gcm:
    enable: 'no'
    key: ''
  port: 45060
  processes: 4
  ssl: 'yes'
  unique_device_ids: 'no'
```

- pushd.enable: Enable/Disable Push Notification feature.
- pushd.apns.certificate: Specify the Apple certificate for push notification.
- pushd.apns.enable: Enable/Disable Apple push notification.
- pushd.apns.key: Specify the Apple key for push notification.
- pushd.gcm.enable: Enable/Disable Google push notification.
- pushd.gcm.key: Specify the Google key for push notification.

C.1.21 qos

The following is the QOS section:

```
qos:
  tos_rtp: 184
  tos_sip: 184
```

- qos.tos_rtp: TOS value for RTP traffic.
- qos.tos_sip: TOS value for SIP traffic.

C.1.22 rate-o-mat

The following is the rate-o-mat section:

```
rateomat:
  enable: 'yes'
  loopinterval: 10
  splitpeakparts: 0
```

- rateomat.enable: Enable/Disable Rate-o-mat
- rateomat.loopinterval: How long we shall sleep before looking for unrated CDRs again.
- rateomat.splitpeakparts: Whether we should split CDRs on peaktime borders.

C.1.23 redis

The following is the redis section:

```
redis:
  database_amount: 16
  port: 6379
  syslog_ident: redis
```

- redis.database_amout: Set the number of databases in redis. The default database is DB 0.
- redis.port: Accept connections on the specified port, default is 6379
- redis.syslog_ident: Specify the syslog identity.

C.1.24 reminder

The following is the reminder section:

```
reminder:
  retries: 2
  retry_time: 60
  sip_fromdomain: voicebox.sipwise.local
  sip_fromuser: reminder
  wait_time: 30
  weekdays: '2, 3, 4, 5, 6, 7'
```

- reminder.retries: How many times the reminder feature have to try to call you.
- reminder.retry_time: Seconds between retries.
- reminder.wait_time: Seconds to wait for an answer.

C.1.25 rsyslog

The following is the rsyslog section:

```
rsyslog:
  elasticsearch:
    action:
      resumeretrycount: '-1'
    bulkmode: 'on'
    dynSearchIndex: 'on'
    enable: 'yes'
    queue:
      dequeuebatchsize: 300
      size: 5000
      type: linkedlist
  external_address:
  external_log: 0
  external_loglevel: warning
  external_port: 514
  external_proto: udp
  ngcp_logs_preserve_days: 93
```

- rsyslog.elasticsearch.enable: Enable/Disable Elasticsearch web interface
- rsyslog.external_address: Set the remote rsyslog server.
- rsyslog.ngcp_logs_preserve_days: Specify how many days to preserve old rotated log files in /var/log/ngcp/old path.

C.1.26 rtpproxy

The following is the rtp proxy section:

```
rtpproxy:
  allow_userspace_only: 'yes'
  maxport: 40000
  minport: 30000
  rtp_timeout: 21600
  rtp_timeout_onhold: 3600
```

- rtpproxy.allow_userspace_only: Enable/Disable the user space failover for rtpengine (*yes* means enable). By default rtpengine works in kernel space.
- rtpproxy.maxport: Maximum port used by rtpengine for RTP traffic.
- rtpproxy.minport: Minimum port used by rtpengine for RTP traffic.
- rtpproxy.rtp_timeout: Maximum limit in seconds for a call (6h).
- rtpproxy.rtp_timeout_onhold: Maximum limit in seconds for an onhold (1h).

C.1.27 security

The following is the security section:

```
security:
  firewall:
    blacklist_networks_4: ~
    blacklist_networks_6: ~
    enable: 'yes'
    sipwise_support_access: 'no'
    whitelist_networks_4: ~
    whitelist_networks_6: ~
```

- security.firewall.enable: Enable/Disable security configuration for IPv6 and IPv6 (sysctl_ipv6.conf, sysctl_ipv4.conf).

C.1.28 sems

The following is the SEMS section:

```
sems:
  bindport: 5080
  conference:
    enable: 'yes'
    max_participants: 10
  debug: 'no'
  highport: 50000
  lowport: 40001
  media_processor_threads: 10
  prepaid:
    enable: 'yes'
  sbc:
    calltimer_enable: 'yes'
    calltimer_max: 3600
    outbound_timeout: 6000
    sdp_filter:
      codecs: PCMA,PCMU,telephone-event
      enable: 'yes'
      mode: whitelist
    session_timer:
      enable: 'yes'
      max_timer: 7200
      min_timer: 90
      session_expires: 300
  session_processor_threads: 10
  vsc:
    block_override_code: 80
    cfb_code: 90
```

```
    cfna_code: 93
    cft_code: 92
    cfu_code: 72
    clir_code: 31
    directed_pickup_code: 99
    enable: 'yes'
    park_code: 97
    reminder_code: 55
    speedial_code: 50
    unpark_code: 98
    voicemail_number: 2000
  xmlrpcport: 8090
```

- sems.conference.enable: Enable/Disable conference feature.
- sems.conference.max_participants: Sets the number of concurrent participant.
- sems.highport: Maximum ports used by sems for RTP traffic.
- sems.debug: Enable/Disable debug mode.
- sems.lowport: Minimum ports used by sems for RTP traffic.
- sems.prepaid.enable: Enable/Disable prepaid feature.
- sems.sbc.calltimer_max: Sets the maximum call duration for inter-domain calls.
- sems.sbc.outbound_timeout:: Sets the maximum call duration for outboud calls.
- sems.sbc.session_timer.enable: Enable/Disable session timers (deprecated, use the web interface configuration).
- sems.vsc.*: Define here the VSC codes.

C.1.29 sshd

The following is the sshd section:

```
sshd:
  listen_addresses:
    - 0.0.0.0
```

- sshd: specify interface where SSHD should run on. By default sshd listens on all IPs found in network.yml with type *ssh_ext*. Unfortunately sshd can be limited to IPs only and not to interfaces. Current option allows to specify allowed IPs (or all IPs with 0.0.0.0).

C.1.30 voisniff

The following is the voice sniffer section:

```
voisniff:
  admin_panel: 'no'
  daemon:
    bpf: 'port 5060 or 5062 or ip6 proto 44 or ip[6:2] & 0x1fff != 0'
    external_interfaces: 'eth0 eth1'
    filter:
      exclude:
        -
          active: 0
          case_insensitive: 1
          pattern: '\ncseq: *\d+ +(register|notify|options)'
      include: []
    internal_interfaces: lo
    mysql_dump_threads: 4
    start: 'no'
    threads_per_interface: 10
  partitions:
    increment: 700000
    keep: 10
```

- voisniff.admin_panel: Enable/Disable SIP STATS on Admin interface. Default is *no*.
- voisniff.deamon.external_interfaces: Define binding interfaces.
- voisniff.deamon.start: Change to *yes* if you want voisniff start at boot. Default is *no*.

C.1.31 www_admin

The following is the WEB Admin interface (www_admin) section:

```
www_admin:
  ac_dial_prefix: 0
  apache:
    autoprov_port: 1444
  billing_features: 1
  callingcard_features: 0
  callthru_features: 0
  cc_dial_prefix: 00
  conference_features: 1
  contactmail: adjust@example.org
  dashboard:
    enabled: 1
  default_admin_settings:
    call_data: 0
```

```
  is_active: 1
  is_master: 0
  read_only: 0
  show_passwords: 1
domain:
  preference_features: 1
  rewrite_features: 1
  vsc_features: 0
fastcgi_workers: 2
fax_features: 1
fees_csv:
  element_order:
    - source
    - destination
    - direction
    - zone
    - zone_detail
    - onpeak_init_rate
    - onpeak_init_interval
    - onpeak_follow_rate
    - onpeak_follow_interval
    - offpeak_init_rate
    - offpeak_init_interval
    - offpeak_follow_rate
    - offpeak_follow_interval
    - use_free_time
http_admin:
  autoprov_port: 1444
  port: 1443
  serveradmin: support@sipwise.com
  servername: "\"myserver\""
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
http_csc:
  autoprov_bootstrap_port: 1445
  autoprov_port: 1444
  port: 443
  serveradmin: support@sipwise.com
  servername: "\"myserver\""
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
logging:
  apache:
    acc:
      facility: daemon
      identity: oss
```

```
      level: info
    err:
      facility: local7
      level: info
peer:
  preference_features: 1
peering_features: 1
security:
  password_allow_recovery: 0
  password_max_length: 40
  password_min_length: 6
  password_musthave_digit: 0
  password_musthave_lowercase: 1
  password_musthave_specialchar: 0
  password_musthave_uppercase: 0
  password_sip_autogenerate: 0
  password_sip_expose_subadmin: 1
  password_web_autogenerate: 0
  password_web_expose_subadmin: 1
speed_dial_vsc_presets:
  vsc:
    - '*0'
    - '*1'
    - '*2'
    - '*3'
    - '*4'
    - '*5'
    - '*6'
    - '*7'
    - '*8'
    - '*9'
subscriber:
  auto_allow_cli: 0
  extension_features: 0
voicemail_features: 1
```

- www_admin.http_admin.*: Define the Administration interface and certificates.
- www_admin.http_csc.*: Define the Customers interface and certificates.
- www_admin.contactmail: Email to show in the GUI's Error page.

C.1.32 www_csc

The following is the WEB Subscriber selfcare (www_csc) section:

```
www_csc:
```

```
ac_dial_prefix: 0
apache:
  port: 443
  serveradmin: support@sipwise.com
  servername: myserver
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
cc_dial_prefix: 00
display_account_info: 0
enable: 'no'
fastcgi_workers: 2
logging:
  apache:
    acc:
      facility: daemon
      identity: csc
      level: info
    err:
      facility: local7
      level: info
main_menu:
  account: 1
  addressbook: 1
  callblock: 1
  callforward: 1
  calllist: 1
  desktop: 1
  device: 0
  fax: 1
  reminder: 1
  voicebox: 1
payment_features: 0
sip_server: sip.yourdomain.tld
site_config:
  company:
    city: ''
    email: ''
    fax: ''
    hotline: ''
    logo: https://please.adjust.invalid/path/to/logo.gif
    name: 'Your Company'
    phone: ''
    street: ''
    webserver: ''
  default_language: en
  default_uri: '/desktop'
  languages:
```

```
    - en
    - es
    - fr
    - de
    - it
  title: 'Sipwise NGCP CSC'
site_domain: sip.yourdomain.tld
tftp_server: tftp.yourdomain.tld
```

- www_csc.apache: Apache configuration for /etc/apache/site-enable/ngcp-csc-admin file.
- www_csc.company: Set here your own company information.
- www_csc.default_language: Default language for CSC interface.
- www_csc.title: Title for CSC interface.
- www_csc.sip_server: Set here your sip server URL.