

Gigaset

Provisioning Guide

Content

Introduction	3
Roles in the provisioning process	4
Server in the provisioning process	5
Gigaset server	5
Provisioning server	5
Provisioning methods	6
Provisioning data	8
Provisioning methods	9
Manual Gigaset VoIP phone set-up – standard procedure	9
Methods for providing the provisioning server URL	10
Setting up redirection information using the web user interface	10
Setting up redirection information using the XML-RPC interface	13
Providing the provisioning server URL via the SIP multicast mechanism	22
DHCP option (dhcp_url)	23
MAC based provisioning	23
Message flow	25
HTTP request	26
Auto-provisioning via activation code	27
Message flow	29
HTTP request	30
Security aspects	31
Updating the configuration data	32
The XML provisioning file	33
Setting up an own provisioning server	38
Installing the auto-provisioning application	38
Preparing the file system	38
Required libraries	39
Installing the gigaset_profile_gen application	39
Setting the access rights for the auto-provisioning files and script	40
Auto-provisioning example script	41
Testing the installation	47
The gigaset_profile_gen application	48
File system structure	49
Index	51

Introduction

Gigaset VoIP phones are delivered to the end-user requiring minimal user interaction for set-up and keeping up-to-date. The end-user experiences the same “plug & play” behaviour as for analogue phones. Unlike classic phones using a PSTN connection, VoIP phones require a variety of configuration parameters which have to be loaded automatically when the device is connected to the Internet.

Provisioning

Provisioning is the process for uploading the necessary configuration and account data to the phone. This is done by means of profiles. A profile is a configuration file that contains Gigaset VoIP phone-specific settings, VoIP provider data as well as user-specific content. It has to be available on an HTTP provisioning server which is accessible for the phone in the public (Internet) or local network.

A profile is loaded to the phone via its Ethernet interface.

Auto-provisioning

Auto-provisioning is defined as the mode of operation by which the Gigaset VoIP phone connects automatically to a server and downloads both provider-specific parameters (such as the URL of the SIP server) and user-specific parameters (such as the user name and password) and stores them in its non-volatile memory.

Auto-provisioning is not necessarily limited to the parameters required for doing VoIP telephony. Auto-provisioning can also be used to configure other parameters, e.g. the eMail settings, if the Gigaset VoIP phones support these features. However, for technical reasons auto-provisioning is not possible for all of the configuration parameters of the phone. As a general rule, all parameters having to do with IP services can be modified by auto-provisioning.

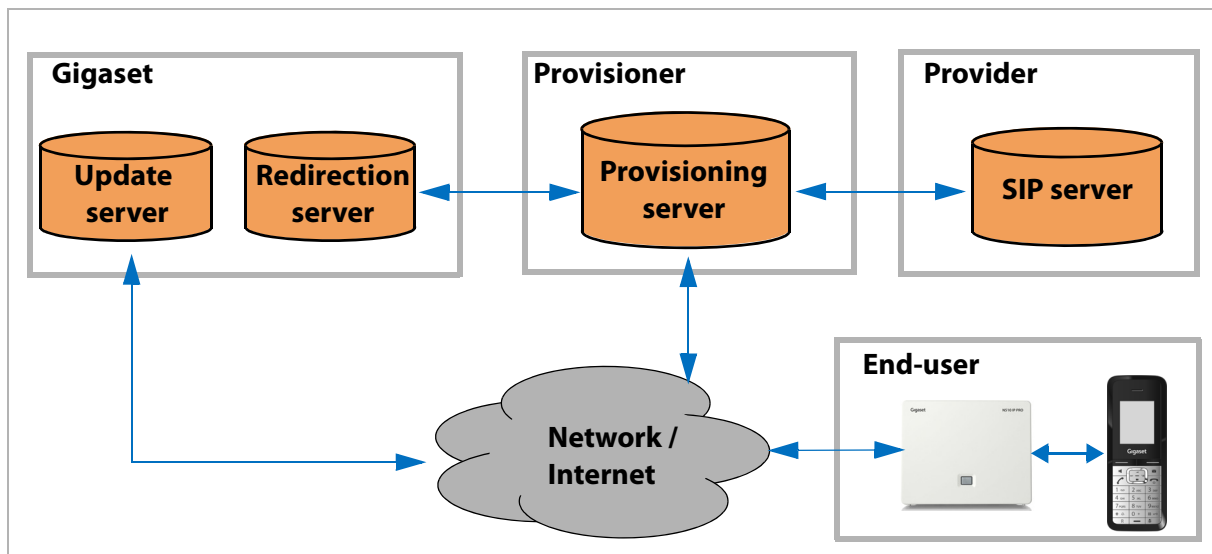
XML-provisioning

Gigaset offers XML-provisioning to the customer, i.e. Gigaset VoIP phones can be supplied with the necessary configuration data via XML content. Existing XML-tooling can be used. At present, the XML file containing the configuration data is used by a profile generation tool to generate a binary file that is supported by the phones. In future, Gigaset VoIP phones will accept XML files directly.

The following Gigaset VoIP phones are supported for auto-provisioning:

	XML-provisioning with binary	Plain XML-provisioning
Gigaset DE900 IP PRO	Yes	Planned Q2 2012
Gigaset DE700 IP PRO	Yes	Planned Q2 2012
Gigaset DE410 IP PRO	Yes	Planned Q2 2012
Gigaset DE310 IP PRO	Yes	Planned Q2 2012
Gigaset DX800A all in one	Yes	–
Gigaset C610 IP/N300 IP	Yes	Planned Q2 2012
Gigaset N510 IP PRO	Yes	Planned Q2 2012
Gigaset N720 DECT IP Multicell System	Yes	Planned 2012

Roles in the provisioning process



Roles in the provisioning process

Gigaset Communications GmbH – the manufacturer

- ◆ Gigaset is the manufacturer of the VoIP phones which are the subject of this document. For Gigaset VoIP phones the MAC address including a check sum is used for identification. Gigaset ensures that this MAC ID is printed on all phone boxes. This is necessary for assigning a specific phone to a specific provider in order to provide the phone with SIP account data.
- ◆ Each device is preconfigured with the same parameters. By default, all Gigaset VoIP phones contact the Gigaset update server when they are connected to the Internet for the first time to get further information, e.g. the URL of the responsible provisioning server.
- ◆ Gigaset provides a web or XML-RPC interface which can be used by provisioners to deploy redirection data on the Gigaset redirection server.

The provider

- ◆ The provider hosts the SIP servers required to offer a complete VoIP telephony service to the end-user. Occasionally, the provider simultaneously assumes the role of provisioner and can host his own provisioning server.

The provisioner

- ◆ The provisioner has direct contact to the end-user and actually manages the VoIP configuration parameters for each individual end-user VoIP phone. The provisioner has to provide the content and perhaps even to operate the server that will be accessed by the phone in order to download the end-user's configuration parameters.
- ◆ When using the Gigaset redirection service the provisioner can operate an own provisioning server. If not, the provisioner is responsible for creating redirection data on the Gigaset server. The MAC ID printed on the device's box can be read by a barcode scanner by the provisioner who has to deploy this information on the Gigaset redirection database using the web interface.
- ◆ The provisioner is also responsible for storing the custom-built data on the provisioning server. If a VoIP phone requests this server, an end-user specific profile is generated and sent to the device.
- ◆ Last but not least, the provisioner has to keep the custom-built data up-to-date.

The end-user

The end-user has to connect the VoIP phone to the Internet only. All related information will be downloaded automatically and there is no need for the end-user to configure parameters manually.

Server in the provisioning process

Gigaset server

◆ Update server

The update server is responsible for providing the Gigaset VoIP phones with

- provider profiles (user-independent data),
- firmware updates,
- language files for the Web user interface (optional),
- help texts (language-specific) for the Web user interface (optional).

Gigaset VoIP phones establish a connection to the Gigaset server when connected to the Internet for the first time and then periodically in order to check if there is an updated configuration file for the Gigaset VoIP phone-specific settings.

By default the Gigaset server *profile.gigaset.net/device* is used as update server.

◆ Redirection server

When the VoIP phone contacts the Gigaset server, in order to get all the necessary configuration data, the redirection server supplies the URL of the provisioning server which is responsible for providing the VoIP phone with the provider data (SIP account).

To enable auto-provisioning (i.e. the end-user does not need to select the provider manually) the provisioner must add the redirection information for the VoIP phones to the redirection database.

Provisioning server

The provisioning server stores custom-built data for providing the VoIP phones with the VoIP specific data (e.g. SIP account).

For many reasons it is possible to use a customised provisioning sever, e.g.:

- ◆ The phone has no possibility to reach the Gigaset server via the web (e.g. closed network without HTTP proxy).
- ◆ The phone is used behind a VoIP PBX and the provisioning has to be independent from the LAN/WAN infrastructure.
- ◆ The provider wants to handle profiles and firmware himself.
- ◆ The provider wants to use an auto-provisioning procedure to support the VoIP phones.

The Gigaset auto-provisioning methods are scalable over a wide area. This means – for example – that it is possible to set up a system completely independent from the Gigaset server or to use the Gigaset server for redirecting to a provider-specific provisioning server.

Setting up a customised provisioning server

The following is required to set up an own provisioning server:

◆ HTTP server (e.g. Apache)

The provisioner has to provide a Linux system with an operative HTTP server, where the specific Gigaset software package can be installed.

◆ Provisioning package provided by Gigaset

Gigaset provides the customer with a provisioning software package. The package includes all the necessary files, scripts, tools and the manual for setting up a provisioning server. The provisioner only has to create a connection between the provisioning script and its database containing the user-specific account data. For detailed information please refer to the chapter **Setting up an own provisioning server** (→ **page 38**).

◆ Database (e.g. MySQL)

The use of a database is optional but it is the usual way for providing custom-built data. Furthermore, the use of a database administration tool (e.g. phpMyAdmin) is helpful to manage the database content.

Provisioning methods

For implementing auto-provisioning of the VoIP phones it must first be ensured that the device receives the address (URL) of the server responsible for provisioning (→ [page 5](#)). As the provisioning server location cannot be anticipated – in the case of a private PBX it may be located within the phone's local network, in the case of a hosted PBX it may be located somewhere in the Internet – the phones use a Gigaset server (*profile.gigaset.net/device*) by default which must be changed according to the provisioner's requirements.

The following methods are provided for the provisioning server URL update, depending on the prevailing network infrastructure:

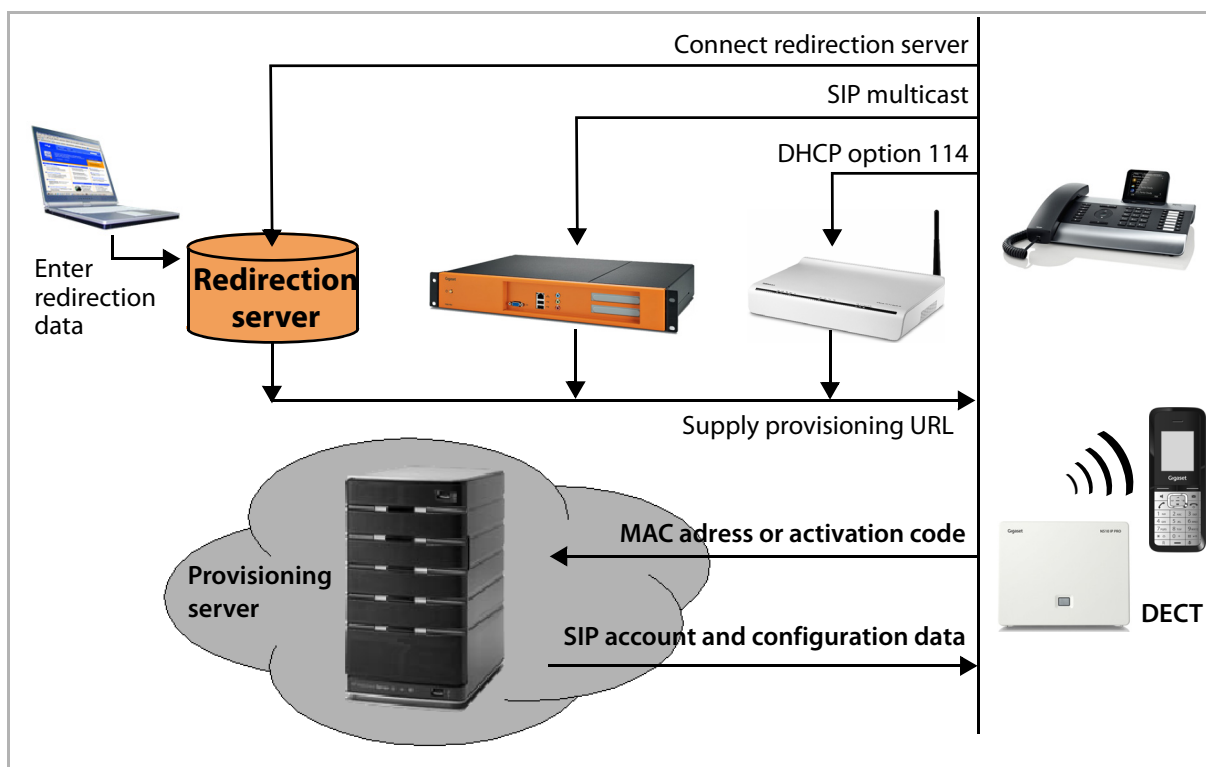
- ◆ Gigaset redirection service (→ [page 10](#))
The provisioner can use either the Gigaset provider/provisioner portal to enter the provisioning server URL for the phones to be managed or use the XML-RPC interface.
- ◆ SIP multicast mechanism (→ [page 22](#))
The phone requests the provisioning server address from a local network instance via SIP multicast. This method is predominantly used by local PBX systems.
- ◆ DHCP option 114 (→ [page 23](#))
The phone requests the provisioning server address via a DHCP request with option 114 (dhcp_url). This method is predominantly used by stand-alone provisioning servers within the same LAN.
- ◆ Manually using the device's Web UI

The following methods are provided for auto-provisioning:

- ◆ Mac-based auto-provisioning (→ [page 23](#))
The VoIP phone requests the provisioning data from the provisioning server based on its MAC address. No user input is necessary. This method is used for VoIP phones connected to (hosted) PBX systems.
- ◆ Auto-provisioning based on an activation code (→ [page 27](#))
The VoIP phone requests the provisioning data from the provisioning server based on an activation code manually entered by the user. This method is used for devices distributed via retail sales.

Profile download can only be started from the VoIP phone, i.e. that the phone must be triggered to perform an update when new configuration data is provided. This can be carried out as follows:

- ◆ VoIP phone restart
- ◆ Manually by the user via the device's Web UI
- ◆ Regular version checks initiated daily by the phone
- ◆ SIP check-sync mechanism (→ [page 32](#))



The following methods are supported by the specific devices:

	MAC-based auto-provisioning	Auto-provisioning with activation code	SIP multicast and check-sync	DHCP option 114
Gigaset DE900 IP PRO	Yes	–	Yes	Yes
Gigaset DE700 IP PRO	Yes	–	Yes	Yes
Gigaset DE410 IP PRO	Yes	–	Yes	Yes
Gigaset DE310 IP PRO	Yes	–	Yes	Yes
Gigaset DX800A all in one	Yes	Yes	Planned 2012	Yes
Gigaset C610 IP/N300 IP	Yes	Yes	Yes	Yes
Gigaset N510 IP PRO	Yes	Yes	Yes	Yes
Gigaset N720 DECT IP Multicell System	Yes	Yes	Yes	Yes

Provisioning data

The following are provisioning data:

Parameters

Gigaset VoIP phones have many configuration parameters but only a small subset is required for provisioning.

◆ General device data

This data is supplied statically via a template and comprises

- general settings for the SIP account, e.g. proxy, registration and STUN server address, port numbers, etc.
- NTP settings, e.g. a time server address
- settings for Info services

◆ User-specific data

It can be extracted from the provisioning database (→ [page 38](#)) and comprises, for example

- SIP username and password
- LAN settings
- voice mail settings, e.g. mail account data
- settings for network directories, e.g. online phonebooks

WebUI texts (optional)

For Gigaset DECT IP phones (e.g. Gigaset N510 IP PRO) only the English language is implemented by default in the firmware. All other languages have to be downloaded from the provisioning server when selected by the user.

Firmware update files (optional)

The provisioner can decide to also host the firmware files on the provisioning server. Gigaset delivers a package with all the current firmware files for the VoIP phones.

Provisioning methods

When the phone is connected to the network for the first time, it needs to connect to a server in the local or public network in order to download the necessary data to be able to make VoIP calls. By default, this is a Gigaset server but it could also be a customer provisioning server, e.g. on a PBX.

Below, the standard manual procedure is described in short.

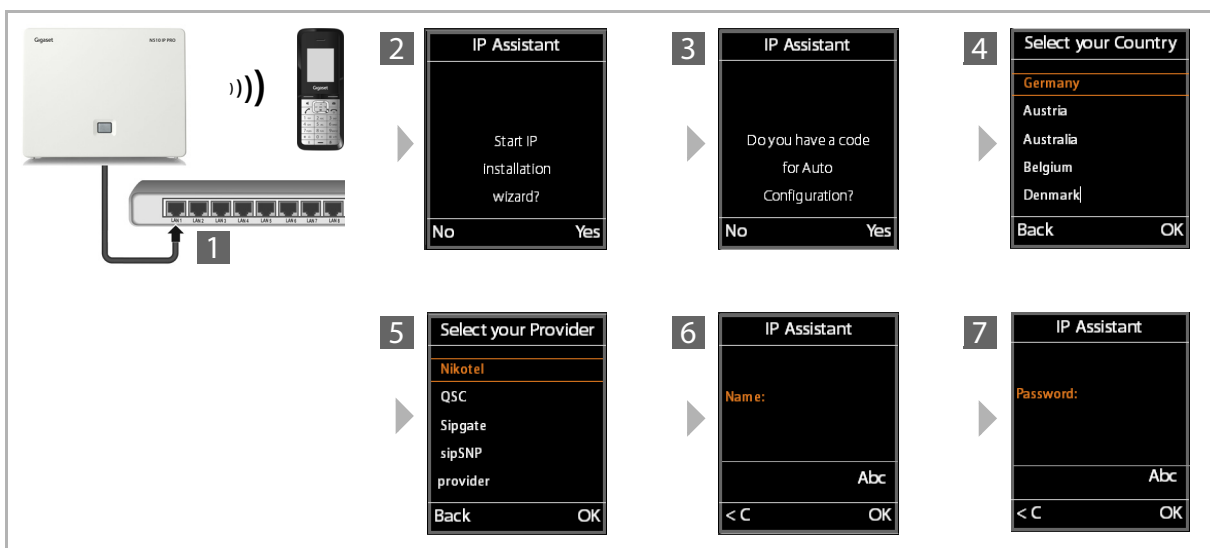
The following methods are available for automatic provisioning (see also introduction on [page 6](#)):

- ◆ For providing the provisioning server URL
 - Gigaset redirection service (→ [page 10](#))
 - SIP multicast mechanism (→ [page 22](#))
 - DHCP option 114 (→ [page 23](#))
- ◆ For auto-provisioning
 - Mac-based auto-provisioning (→ [page 23](#))
 - Auto-provisioning based on an activation code (→ [page 27](#))
- ◆ For profile update
 - SIP check-sync mechanism (→ [page 32](#))

Manual Gigaset VoIP phone set-up – standard procedure

A Gigaset VoIP phone can be set up manually via the phone's user interface – in the case of a DECT VoIP phone via the handset. The use of a PC is not required but possible using a Web user interface which is available for all Gigaset VoIP phones.

The default manual registration procedure for a Gigaset VoIP phone is as follows:



Standard manual registration procedure

- ◆ The end-user connects the phone to the network **1**. Internet access is required.
- ◆ The connection assistant is started **2**.
- ◆ If available the activation code supplied by the provider is entered **3** (→ [page 27](#)).
- ◆ If the phone establishes a connection to the Gigaset server to download a provider profile, this download is carried out in two steps:
 - All countries for which a provider profile is available are listed for the user to select the location **4**.
 - All providers of the selected country for which a profile is available are displayed for the user to select the provider **5**.

Provisioning methods

- ◆ The profile is loaded from the Gigaset server onto the VoIP phone. In this case Gigaset acts as a provisioner for general (user-independent) SIP settings.
- ◆ The user enters the authentication name **6** and the password **7** according to the rules given by the provider profile.

Methods for providing the provisioning server URL

Setting up redirection information using the web user interface

To add the redirection data to the redirection database, Gigaset provides a web user interface for provisioners.

You need a user account (user name and password) which has to be provided by Gigaset Communications GmbH.



There is also an XML-RPC interface available to provide redirection data. The XML-RPC calls are described in detail → **Page 13**.

- ▶ Open the web user interface:

<http://prov.gigaset.net>

- ▶ Login using the user name and password provided by Gigaset.

If the login is successful, the main menu is opened.

The following functions are available:

- ◆ Registration, control and deregistration of single devices
- ◆ Display of devices list
- ◆ Upload of prepared XML files

Registering VoIP phones

- ▶ To register a Gigaset VoIP phone, enter the **MAC ID** of the device, the **URL** of the provisioning server and the **Provider** for the device configuration.

URL and **Provider** can be entered manually or selected from a list of known provisioner URLs and providers.

- ▶ Click on the **Register** button to save the entry.

The corresponding parameters are checked and – if approved – saved in the Gigaset redirection database. The provisioner is informed accordingly.

Searching for and deregistering VoIP phones

The **Deregister** tab can be used to query single redirection data records using the MAC ID or MAC address.

- ▶ To search for a specific redirection data set, open the **Deregister** tab, enter the MAC ID or MAC address of the device and click on the **Search** button.

MAC-ID:

MAC address:	7C2F8016F926
Provider's name:	Provider
Provisioning URL:	http://192.168.2.161:82/gigaset/cgi/ap?mac=%MACD
Last request:	2011-05-11

If the MAC ID or MAC address matches, the redirection data for a specific device appears.

- ▶ To deregister the redirection data record for the device click on the **Deregister** button. You need to confirm this action. It is therefore not possible to delete entries accidentally.

List devices

The **List Devices** tab can be used to search for redirection data sets of all devices or all devices of a specific provider.

- ▶ Open the **List Devices** tab
- ▶ Click on the **List** button to list the redirection data sets of all devices.

or

- ▶ Enter the name of a provider or select it from the list and click on the **Search** button.

Provider:

MAC address:	Provider's name:	Provisioning URL:	Last request:
7C2F8016F926	Provider	http://192.168.2.161:82/gigaset/c	2011-05-11
7C2F8016FAF3	Provider	http://192.168.137.106/gigaset/c	2011-09-10
7C2F801D1121	Provider	http://192.168.178.51/chagall/42/	2011-12-14
7C2F80207541	Provider	http://192.168.178.51/gigaset/62/	2011-11-29

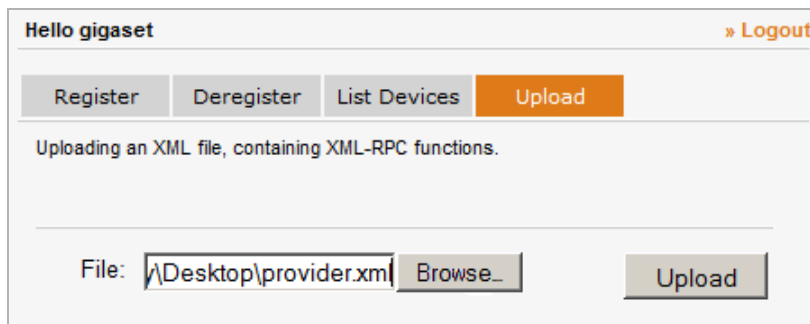
The list shows all devices that have ever been uploaded by the provisioner (possibly dependent on the provider name).

Provisioning methods

Uploading an XML File

It is also possible to upload an XML file containing many redirection data bundles. This option is most useful for provisioners whose end-user profiles are generated automatically by any script and who want to keep using the web user interface instead of direct interfaces for clients. However, more than just uploading redirection data can be carried out. Depending on the structure of the XML file, any function provided by the interfaces may be possible. For detailed information on the XML parameter values please refer to the section **Setting up redirection information using the XML-RPC interface** (→ **page 13**).

- Open the **Upload** tab, browse your file system for the appropriate XML file and click on the **Upload** button.



The screenshot shows a web interface for Gigaset. At the top, it says "Hello gigaset" and has a "Logout" link. Below this are four buttons: "Register", "Deregister", "List Devices", and "Upload" (which is highlighted in orange). Under the buttons, it says "Uploading an XML file, containing XML-RPC functions." Below this is a file input field with the text "File: \Desktop\provider.xml" and a "Browse..." button. To the right of the file input is an "Upload" button.

i

There is also another interface which allows the upload of XML files without using the web user interface.

Commons FileUpload is used (<http://commons.apache.org>) at the server end.

An HTTP client is necessary at the client end.

Principally, this process is similar to the remote procedure calls described in section **Setting up redirection information using the XML-RPC interface** (→ **page 13**). The difference between the versions is that on the one hand XML-RPC clients request remote methods and transfer parameters in order to create an XML file, while on the other hand HTTP clients upload an XML file directly. The latter method – like the XML file upload via the web user interface – is suitable for users whose XML files are created through scripts but who do not want to use the web user interface. Please refer to paragraph **XML syntax** (→ **page 14**) for further details regarding the structure of XML files.

File upload

The HTTP XML file upload can be accessed using the following address:

<https://prov.gigaset.net/apxml/basic.do>

Content type for XML FileUpload

The default content type used in most cases is

`application/x-www-form-urlencoded`

But if a provisioner wants to upload an XML file either via the web user interface or via an HTTP client it would be very inefficient to use the default content type.

For this reason, XML files are uploaded with the content type

`multipart/form-data`

which is suitable for sending large data. Commons FileUpload corresponds exactly to this format.

Setting up redirection information using the XML-RPC interface

The most comfortable Gigaset server interface is the XML-RPC interface. XML-RPC is subject to a procedure call protocol, which means that functions can be accessed at different places. XML-RPC requests are encoded in XML syntax and dispatched via HTTP.

The Gigaset XML-RPC web interface is based on Apache XML-RPC libraries and can be accessed by any XML-RPC client. The client can call different remote procedure methods. The parameter values and the function names are converted into XML syntax simultaneously and transferred to the XML-RPC server via HTTP. The XML file is parsed through the Apache XML-RPC libraries and again, the suitable remote method with its corresponding parameter values is called. The response of this remote method is again converted into XML syntax, sent back to the client and processed by the XML-RPC client libraries.

The XML-RPC service can be accessed using the following address:

`http://prov.gigaset.net/apxml/rpc.do`

An XML-RPC call for an XML-RPC client could be as follows:

```
Object[] params = new Object[]{"#MAC-ID#", "#PROVISIONING_URL#",
    "#PROVIDERNAME#"};
List<Object> list1 = Arrays.asList((Object[])client.execute
    ("autoprov.registerDevice", params));
```

On the Gigaset XML-RPC web interface the implementation appears as follows:

```
public List<Object> registerDevice(String macID, String url, String name) {
    . . .
    use parameter values and create response list
    . . .
    return myList;
}
```

The XML files are invisible on both sides, when effecting the remote procedure call on the client as well as on the server because the XML files are parsed via the XML-RPC libraries. Nevertheless, the plain text (method + parameter) has to be transferred via HTTP in a well- structured manner which makes XML a perfect alternative.

Provisioning methods

XML syntax

Whenever an XML-RPC client requests a remote method on the Gigaset XML-RPC server, an XML file is created consisting of the method's name and the parameters. Both the file and the response are transferred via HTTP. The upload of XML files via *FileUpload* has to look exactly like this as well.

The uploaded XML file, depending on the method for registering a redirection data set, can for example appear as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>autoprov.registerDevice</methodName>
  <params>
    <param>
      <value>FFFFFFFFFFFFFF-1234</value>
    </param>
    <param>
      <value>http://my.provisioning.server.com/gigaset/ap.php</value>
    </param>
    <param>
      <value>MyProvider</value>
    </param>
  </params>
</methodCall>
```

After a successful request, the corresponding XML response file looks as follows :

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param>
      <value><array><data>
        <value><boolean>0</boolean></value>
        <value>mac_already_in_use:FFFFFFFFFFFFFF</value>
      </data></array></value>
    </param>
  </params>
</methodResponse>
```

XML-RPC Commands

If an XML-RPC client calls a remote procedure function on the Gigaset XML-RPC server, an XML file consisting of the XML-RPC method and the corresponding parameter values is created and is transferred via HTTP. The response is transferred the same way.

The respective XML-RPC methods as well as the relevant parameter values, return values and XML formats are illustrated below:

autoprov.registerDevice

Registering a device at the provisioning server:

Call: `autoprov.registerDevice(String macID, String url, String name)`

<code>macID</code>	MAC ID of the device
<code>url</code>	URL of the provisioning server
<code>name</code>	Provider name

Return: Return value (1) (Boolean) 1 | 0 1 = true, 0 = false

 Return value (2) (String) if true: OK:password

 if false: mac_already_in_use:

 mac_invalid:

 url_invalid:

 name_invalid:

Request: `<?xml version="1.0" encoding="UTF-8"?>`

```

<methodCall>
  <methodName>autoprov.registerDevice</methodName>
  <params>
    <param>
      <value>FFFFFFFFFFFF-1234</value>
    </param>
    <param>
      <value>https://my.provisioning.server.com/gigaset/ap.php</value>
    </param>
    <param>
      <value>MyProvider</value>
    </param>
  </params>
</methodCall>

```

Response: `<?xml version="1.0" encoding="UTF-8"?>`

```

<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param>
      <value><array><data>
        <value><boolean>0</boolean></value>
        <value>mac_already_in_use:FFFFFFFFFFFF</value>
      </data></array></value>
    </param>
  </params>
</methodResponse>

```

Provisioning methods

autoprov.deregisterDevice

Deregistering a device from the provisioning server.

Call: autoprov.deregisterDevice(String mac)
 mac MAC ID or MAC address of the device

Return: Return value (1) (Boolean) 1 | 0 1 = true, 0 = false
 Return value (2) (String) if true: OK
 if false: mac_not_found:
 mac_invalid:

Request: <?xml version="1.0" encoding="UTF-8"?>
 <methodCall>
 <methodName>autoprov.deregisterDevice</methodName>
 <params>
 <param>
 <value>FFFFFFFFFFFF</value>
 </param>
 </params>
 </methodCall>

Response: <?xml version="1.0" encoding="UTF-8"?>
 <methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
 <params>
 <param>
 <value><array><data>
 <value><boolean>1</boolean></value>
 <value>OK</value>
 </data></array></value>
 </param>
 </params>
 </methodResponse>

autoprov.listDevices - list devices for all providers

Supply MAC address, provider name, provisioning server URL and registration date for all registered devices:

Call: `autoprov.listDevices()`

Return: Return value (0-n)
 `(Object[])`
 `[MAC, NAME, URL, DATE]`

Request: `<?xml version="1.0" encoding="UTF-8"?>`
 `<methodCall>`
 `<methodName>autoprov.listDevices</methodName>`
 `<params />`
 `</methodCall>`

Response: `<?xml version="1.0" encoding="UTF-8"?>`
 `<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">`
 `<params>`
 `<param>`
 `<value><array><data>`
 `<value><array><data>`
 `<value>BBBBBBBBBBBB</value>`
 `<value>MyProvider1</value>`
 `<value>https://my.provisioning.server.com/gigaset/ap.php</value>`
 `<value>2009-11-29</value>`
 `</data></array></value>`
 `<value><array><data>`
 `<value>EEEEEEEEEEEE</value>`
 `<value>MyProvider2</value>`
 `<value>https://my.provisioning.server.com/gigaset/ap.php</value>`
 `<value>2009-11-27</value>`
 `</data></array></value>`
 `</data></array></value>`
 `</param>`
 `</params>`
 `</methodResponse>`

Provisioning methods

autoprov.listDevices - list devices of a specific provider

Supply MAC address, provider name, provisioning server URL and registration date for all registered devices of a given provider:

Call: `autoprov.listDevices (String name)`
 `name` Provider name

Return: Return value (0-n)
 `(Object [])`
 `[MAC, NAME, URL, DATE]`

Request: `<?xml version="1.0" encoding="UTF-8"?>`
 `<methodCall>`
 `<methodName>autoprov.listDevices</methodName>`
 `<params />`
 `</methodCall>`

Response: `<?xml version="1.0" encoding="UTF-8"?>`
 `<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">`
 `<params>`
 `<param>`
 `<value><array><data>`
 `<value><array><data>`
 `<value>BBBBBBBBBBBB</value>`
 `<value>MyProvider1</value>`
 `<value>https://my.provisioning.server.com/gigaset/ap.php</value>`
 `<value>2009-11-29</value>`
 `</data></array></value>`
 `</data></array></value>`
 `</param>`
 `</params>`
 `</methodResponse>`

autoprov.checkDevice

Supply provider name, provisioning server URL and registration date for a specific device:

Call: `autoprov.checkDevice(String mac)`
 `mac` MAC ID or MAC address of the device

Return: Return value (1) (Boolean) 1 | 0 1 = true, 0 = false
 Return value (2) (String) if true: [MAC]
 if false: max_not_found:
 mac_invalid:
 Return value (3) (String) if true: [NAME]
 Return value (4) (String) if true: [URL]
 Return value (5) (String) if true: [DATE]

Request: <?xml version="1.0" encoding="UTF-8"?>
 <methodCall>
 <methodName>**autoprov.checkDevice**</methodName>
 <params>
 <param>
 <value>**BBBBBBBBBBBBBB**</value>
 </param>
 </params>
 </methodCall>

Response: <?xml version="1.0" encoding="UTF-8"?>
 <methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
 <params>
 <param>
 <value><array><data>
 <value><boolean>**1**</boolean></value>
 <value>**BBBBBBBBBBBBBB**</value>
 <value>**MyProvider1**</value>
 <value>**https://my.provisioning.server.com/gigaset/ap.php**</value>
 <value>**2009-11-29**</value>
 </data></array></value>
 </param>
 </params>
 </methodResponse>

Provisioning methods

autoprov.registerDeviceList

Supply provider name, provisioning server URL and registration date for a list of registered devices:

Call:

```
autoprov.registerDeviceList(List<String> macList,  
String url, String name)  
macList      List of MAC IDs  
url          URL of the provisioning server  
name         Provider name
```

Return:

Return value (1) (Boolean)	1 0	1 = true, 0 = false
Return value (2-n) (String)	if true:	OK: Passwort mac_already_in_use: mac_not_exist:
	if false:	mac_invalid: url_invalid: name_invalid:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>  
<methodCall>  
  <methodName>autoprov.registerDeviceList</methodName>  
  <params>  
    <param>  
      <value><array><data>  
        <value>111111111111-ABCD</value>  
        <value>222222222222-BCDE</value>  
        <value>333333333333-CDEF</value>  
        <value>444444444444-DEFA</value>  
      </data></array></value>  
    </param>  
    <param>  
      <value>https://my.provisioning.server.com/gigaset/ap.php</value>  
    </param>  
    <param>  
      <value>MyProvider</value>  
    </param>  
  </params>  
</methodCall>
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>  
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">  
  <params>  
    <param>  
      <value><array><data>  
        <value><boolean>1</boolean></value>  
        <value>OK</value>  
      </data></array></value>  
    </param>  
  </params>  
</methodResponse>
```

Another Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param>
      <value><array><data>
        <value>mac_already_in_use:111111111111</value>
        <value>mac_already_in_use:333333333333</value>
      </data></array></value>
    </param>
  </params>
</methodResponse>
```

autoprov.deregisterDeviceList

Deregister a list of registered devices:

Call:

```
autoprov.deregisterDeviceList(List<String> macList)
macList          List of MAC IDs or MAC addresses
```

Return: Return value (1) (Boolean) 1 | 0 1 = true, 0 = false
 Return value (2-n) (String) if true: OK
 mac_not_found:
 if false: mac_invalid:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>autoprov.deregisterDeviceList</methodName>
  <params>
    <param>
      <value><array><data>
        <value>111111111111</value>
        <value>222222222222</value>
        <value>333333333333</value>
        <value>444444444444</value>
      </data></array></value>
    </param>
  </params>
</methodCall>
```

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param>
      <value><array><data>
        <value><boolean>1</boolean></value>
        <value>OK</value>
      </data></array></value>
    </param>
  </params>
</methodResponse>
```

Another <?xml version="1.0" encoding="UTF-8"?>

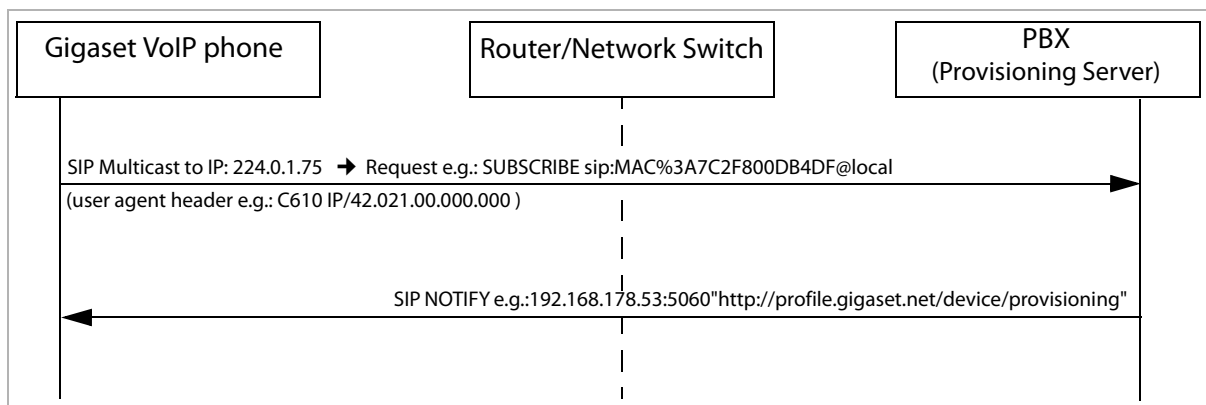
Response: <methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
 <params>
 <param>
 <value><array><data>
 <value>mac_not_found:111111111111</value>
 <value>mac_not_found:333333333333</value>
 </data></array></value>
 </param>
 </params>
</methodResponse>

Providing the provisioning server URL via the SIP multicast mechanism

This mechanism is an easy method for loading the URL of the provisioning server, on which the configuration files (profiles) and/or the firmware files of the different Gigaset VoIP phones are located. The mechanism is designed for VoIP PBXs offering an own provisioning server for the connected VoIP phones.

Before an answer is sent to the initiator of the SIP multicast, the PBX (or SoftSwitch) has to identify the phone type. This is done via the SIP User-Agent header which starts with the product name.

The following example flow chart shows the principle of this mechanism:

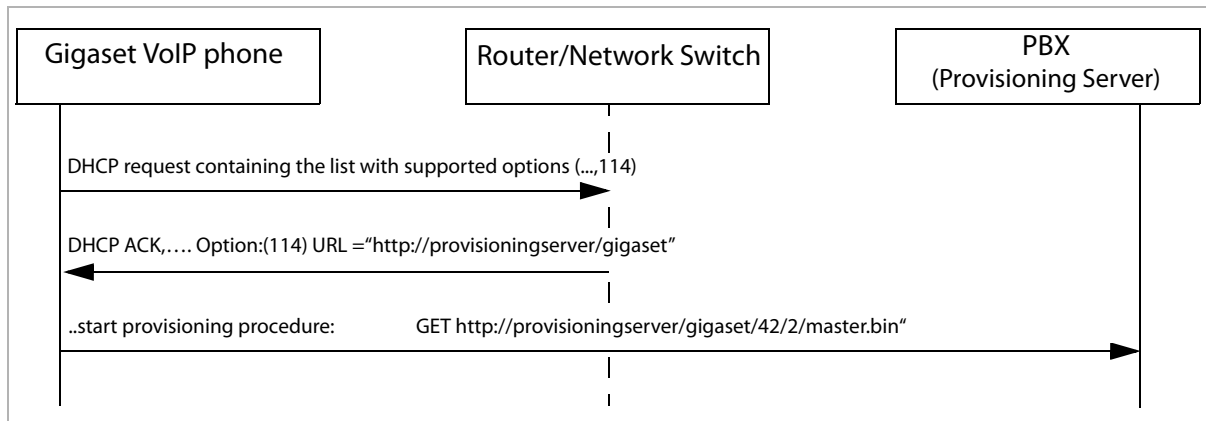


The SIP multicast mechanism is supported by most of the Gigaset VoIP phones and by the Gigaset T300 PRO and Gigaset T500 PRO PBX as well (it is also supported by some products from other companies).

DHCP option (dhcp_url)

As an alternative option for assigning the provisioning server URL to the VoIP phones, the DHCP option 114 (`dhcp_url`) can be used.

The following example flow chart shows the principle of this mechanism:



MAC based provisioning

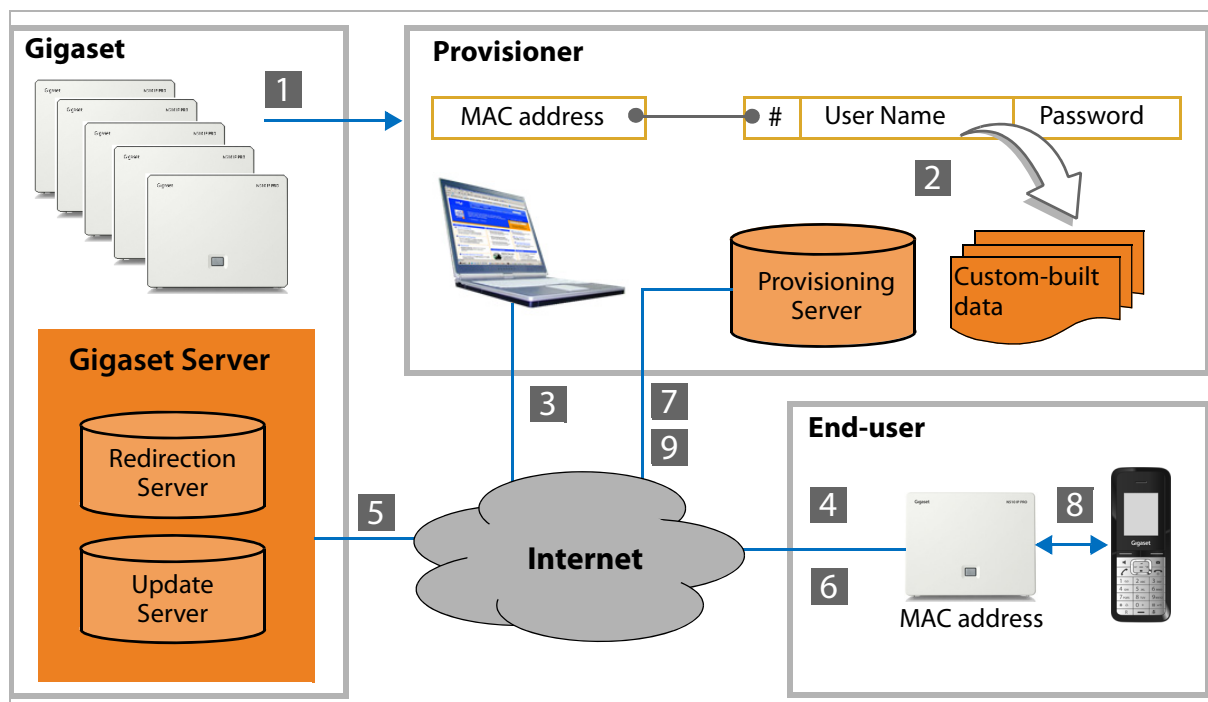
User-specific profiles are loaded without any user interaction. The database containing the user-specific data can be hosted on a provider/PBX related server. When the phone contacts the provisioning server, it identifies itself by means of its own MAC address. Each phone has a globally unique MAC address that was assigned to it in the factory.

This method has the advantage that, ideally, the end-user does not even have to be aware of the need for provisioning: he simply plugs in the phone, and the rest happens automatically in the background.

For this to be successful, the provisioner has to create the association between the MAC address and the end-user prior to delivering the phone to the end-user. This is done efficiently if the MAC ID (containing MAC address and a 4-digit part of a unique password) is printed on the phone's box as barcode, so that the provisioner can scan the address, assign this with the provisioner's URL on the Gigaset server (optional) and enter it in its database.

Because the phone periodically queries for configuration updates (every 24h), this method gives the provisioner the means to effectively control the end-user's phone configuration.

Provisioning methods



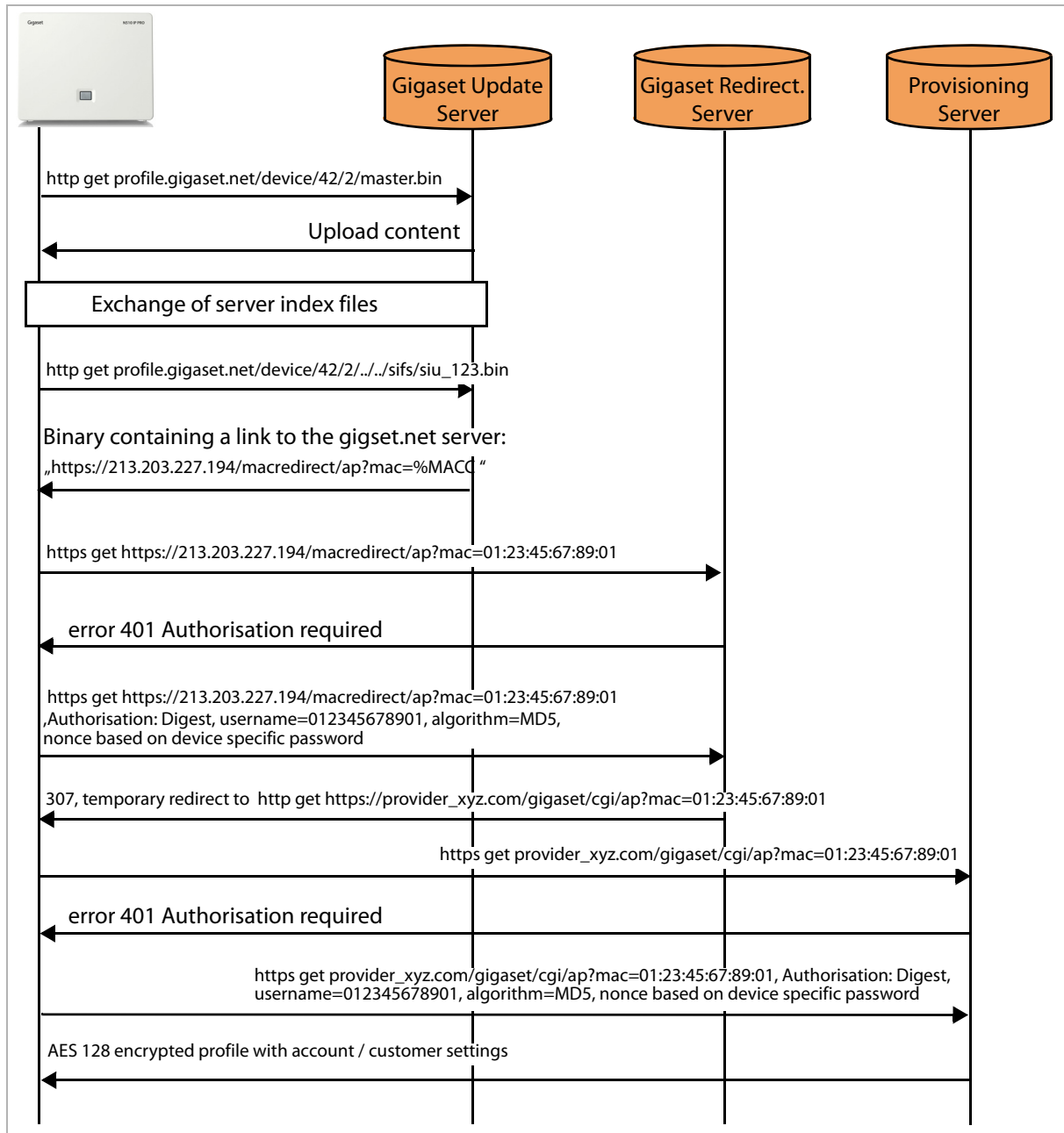
Auto-provisioning via MAC address

Principle of operation – MAC-based provisioning with redirection

- 1 Gigaset prints the MAC address barcode on the housing and the box.
- 2 The provisioner scans the MAC address and associates it with the user data, e.g. the SIP account. The MAC ID consists of the MAC address plus a random ID (4 characters) that is added to the MAC address and has the following syntax:
<MAC-address>-<ID>, e.g. 001122334455-ABCD
- 3 The provisioner enters the redirection data on the Gigaset server. The device's MAC ID and the URL of the provisioning server have to be provided.
Gigaset therefore provides a web interface as well as an XML-RPC interface. Redirection data describes a connection between the MAC address, the provisioning server's URL and the provider's name. This configuration bundle is created by the provisioner and has to be stored on the Gigaset redirection database.
- 4 The end-user connects the phone to the network and the phone contacts the Gigaset server.
- 5 The Gigaset server checks the MAC ID. If the MAC ID is available in the Gigaset provisioning database of the redirection server it transfers the provisioning server address for this device to the phone.
- 6 The phone connects to the provisioning server providing its MAC address.
- 7 The provisioning server uploads the custom-built data to the phone. The provisioner is responsible for creating the custom-built data to store this information on the provisioning server and to keep it up-to-date.
- 8 The phone is now ready to initiate the first call.
- 9 Periodically, the phone connects to the provisioning server in order to check if there is new custom-built data available (once a day).

Message flow

The following diagram shows – in a simplified manner – the message flow between a Gigaset VoIP phone and the involved servers from the auto-provisioning point of view.



The message flow illustrates the steps **4** to **7** of the image on **page 24**.

Communication takes place by means of HTTP requests.

Profiles are stored in a binary format on the provisioning server depending on the device variant. The Gigaset server uses the phone's variant ID to upload the matching configuration. For example, *http://profile.gigaset.net/device/42/2/master.bin* refers to the configuration files of a Gigaset N510 IP PRO phone (→ **page 49**).

The Gigaset redirection server uses the phone's MAC ID to search for the responsible provisioning server for this phone.

Provisioning methods

HTTP request

When the phone contacts the provisioning server in order to download the auto-provisioning file, it performs an `HTTP:GET` for a URL with the following format:

```
http://<server domain>/<directory>/<ap>?mac=<mac address>
```

<code><server domain></code>	DNS name (or IP address) of the provisioning server.
<code><directory></code>	Path to the auto-provisioning script within the server domain. The script is a CGI application, i.e. it runs inside the HTTP server (→ page 41). It constructs a personalised XML configuration file for the requesting phone, then calls an application which creates the binary configuration file (<i>gigaset_profile_gen</i> , → Page 48), and finally returns the configuration data to the phone.
<code><ap></code>	Name of the auto-provisioning script: <i>ap</i> (auto-provision).
<code><mac address></code>	MAC address of the VoIP phone generating the request, in the 6 hex digit-pair textual representation, with or without a colon between each digit pair; for example: 06:55:AF:3A:05:AA or 0655AF3A05AA

Example of a request:

```
http://my.server.domain.com/gigaset/cgi/ap?mac=06:55:AF:3A:05:AA
```

URI format

The URL where the *ap* script is performed is determined by an additional query to get a so-called URI format string. The server for this query is the same server which is used for firmware updates (normally – not necessarily, but recommended – the Gigaset server). The query is done with a 3-digit provisioner code. This code is preprogrammed in the factory and is used by the phone to get the file with the format string. This format string is also used by the phone to build the command to get the profile from the provisioner's server. The format string contains fixed text (used for the query as it is) and format specifier (with a leading %) which will be replaced by the phone. The relevant configuration possibilities of the URI-format string are the following:

Format specifiers:

<code>%DVID</code>	Device ID, composed by build variant and provisioning ID. Example: 42/2
<code>%MACC</code>	MAC with colons. Example: .00:01:E3:12:34:56.
<code>%MACD</code>	MAC without colons. Example: .0001E3123456.
<code>%%</code>	To represent the percent character.

Example:

The URI-format string:

```
http://my.server.domain.com /%DVID/cgi/ap?mac=%MACC
```

Leads to a request with the command:

```
GET http://my.server.domain.com/42/2/cgi/ap?mac=00:01:E3:12:34:56
```

Auto-provisioning via activation code

With this method the end-user has to enter a unique code – the activation code – on the phone when setting it up.

For the provisioner, this method is similar to the MAC address one, but it has the advantage that the phones does not need to be handled before sending it to end-user. It just has to be ensured that a unique activation code is created by which the end-user can be identified unambiguously. For the Gigaset VoIP phones the activation code can be a numeric string with a maximum of 32 characters. Depending on the configuration, the phone downloads the configuration parameters only once ("one-shot provisioning". The end-user thus has the freedom to modify all configuration parameters and/or load other profiles. Alternatively, the phone downloads the profile periodically using the activation code entered initially and stores the parameter when the profile is changed.

The activation code

The activation code is realised as a key. It consists of two parts that are concatenated as follows:

```

Activation Code  = <Gigaset part><Provisioner part>
Gigaset part    = Identifies the provisioner for the related phone.
                  3 decimal numbers.
Provisioner part = <user id>[#<password>]
                  Max. 29 decimal numbers.
                  user id    Identifies the end-user unambiguously.
                  #password  Optional password, which is preceded by the # character
                              as separator. The parameter is used for the HTTP digest
                              authentication algorithm.

```

Examples of activation codes:

```

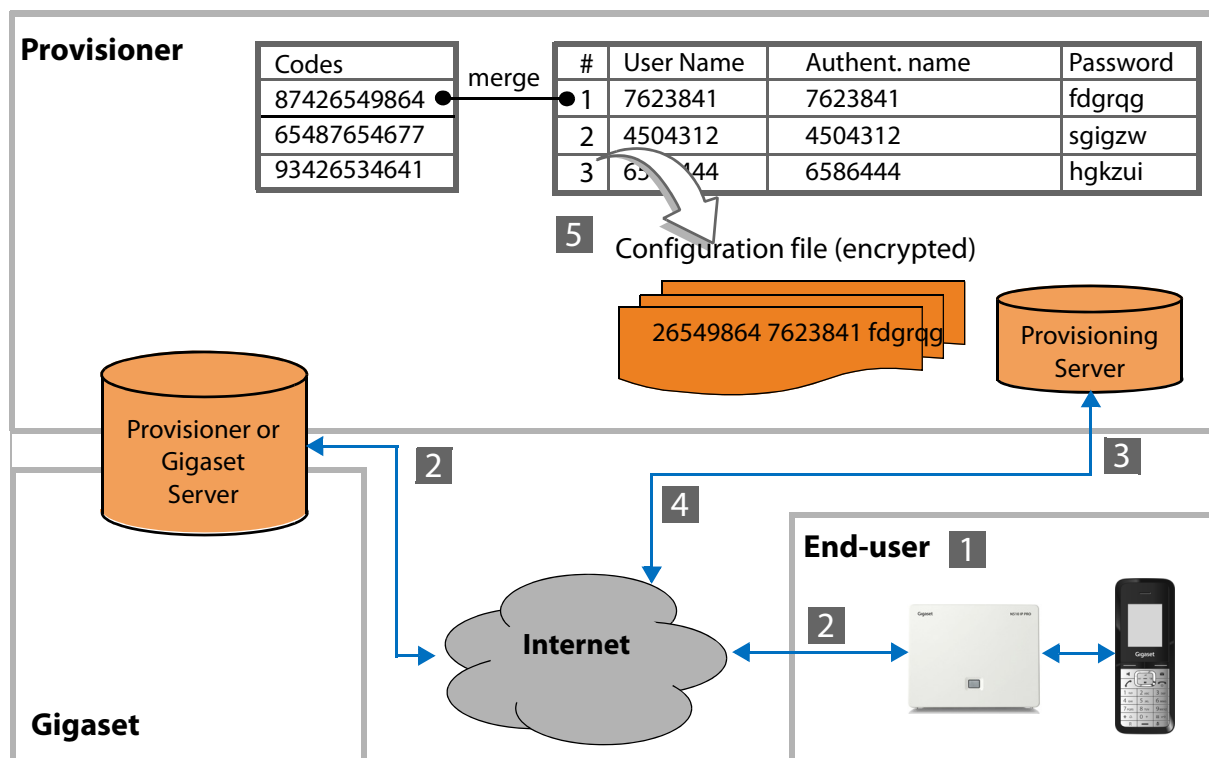
12387426549864#0815    provider = 123, user id = 87426549864, password = 0815.
0159039885893          provider = 015, user id = 9039885893, no password

```

The end-user receives the complete activation code directly from the provisioner, by eMail or during registration at the provisioner's web site, for example.

When generating a new activation code, the provisioner must always prepend the Gigaset part (which is defined by Gigaset Communications when a new provisioner requests this feature). Apart from respecting the basic syntax described above, the provisioner is free to design the provisioner part of the code as desired.

Provisioning methods



Auto-provisioning via activation code

Principle of operation

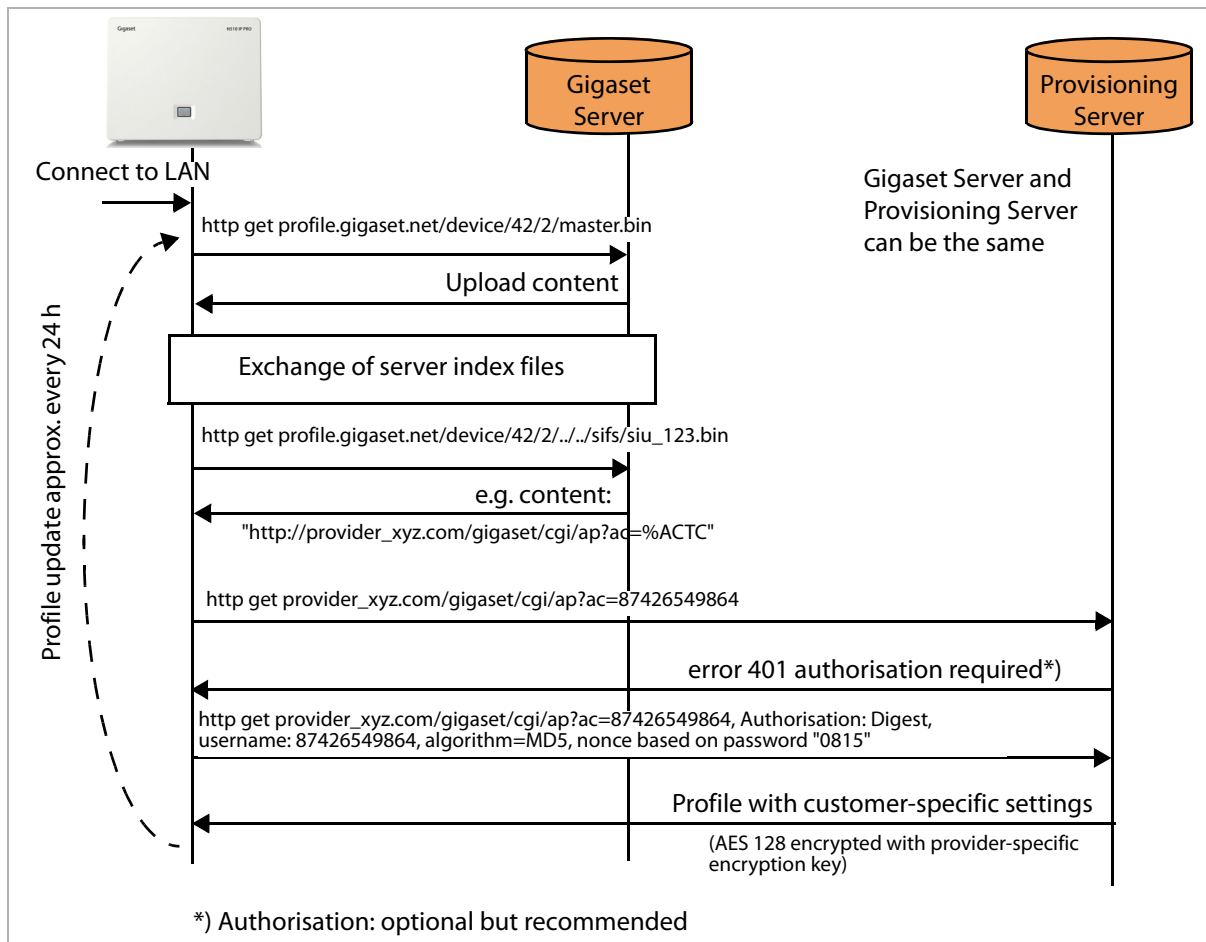
- 1** The end-user buys a phone, installs it and enters the activation code either via the handset procedure (installation assistant) or via the web user interface.
- 2** The phone extracts the Gigaset part of the activation code and uses it to request the URL of the provisioning server, normally at the central (Gigaset) server, where the download profiles and firmware files are located. The provisioning (Gigaset) server sends the provisioner's URL to the phone.
- 3** The phone now sends the *user id* part of the activation code to the provisioner's URL.
- 4** If the provisioner requests authentication of the user, the request is denied with 401 (Unauthorised). In this case the phone must re-issue the request and provide the authorisation header – the content of which must be calculated using the *password* part of the activation code. The password part is of course never transmitted directly.
- 5** The provisioner uses the *user id* to feed a CGI application which searches its database for the given users and then constructs and returns the encrypted user profile.

The received profile also contains the name of the profile which is stored in the phone. From now on, the phone will periodically check for a changed profile by using the stored profile name ("one-shot provisioning"). This means that the profile, (normally) located at the Gigaset server, can be the same one used for manually downloading the profile. It therefore has to be guaranteed that the name of the received profile is identical with the one on the (Gigaset) server. Depending on the configuration within the downloaded profile the phone can alternatively download the profile periodically using the activation code entered initially.

Message flow

The following diagram shows – in a simplified manner – the message flow between a Gigaset VoIP phone and the involved servers from the auto-provisioning point of view.

Use Case: The VoIP phone is prepared for auto-provisioning with an activation code (locked or non-locked [One-Shot]) and the customer feeds in the activation code 12387426549864#0815 via the handset or WEB-UI.



The message flow illustrates steps **3** to **7** of the image on **page 28**.

Communication takes place by means of HTTP requests.

Profiles are stored in a binary format on the provisioning server depending on the phone variant. The Gigaset server uses the phone's variant ID to upload the matching configuration. For example, *http://profile.gigaset.net/device/42/2/master.bin* refers to the configuration files of a Gigaset N510 IP PRO phone (→ **page 49**).

Provisioning methods

HTTP request

When the phone contacts the provisioning server in order to download the auto-provisioning file, it performs an `HTTP :GET` for a URL with the following format:

```
http://<server domain>/<directory>/<ap>?ac=<activation code>
```

<code><server domain></code>	DNS name (or IP address) of the provisioning server.
<code><directory></code>	Path to the auto-provisioning script within the server domain. The script is a CGI application, i.e. it runs inside the HTTP server (→ page 41). It constructs a personalised XML configuration file for the requesting phone, then calls an application which creates the binary configuration file (<i>gigaset_profile_gen</i> , → Page 48), and finally returns the configuration data to the phone.
<code><ap></code>	Name of the auto-provisioning script: <i>ap</i> (auto-provision).
<code><activation code></code>	Activation code of the VoIP phone generating the request.

Example:

```
http://my.server.domain.com/gigaset/cgi/ap?ac=0159039885893
```

URI format

The URL, where the *ap* script is performed, is determined by an additional query to get a so-called URI format string. The server for this query is the same server which is used for firmware updates (normally – not necessarily, but recommended – the Gigaset server). The query is done with a 3-digit provisioner code. This code is preprogrammed in the factory or is part of the activation code which was entered manually and is used by the phone to get the file with the format string. This format string will be used again by the phone to build the command to get the profile from the provisioner's server. The format string contains fixed text (used for the query as it is) and a format specifier (with a leading %) which will be replaced by the phone. The relevant configuration possibilities of the URI-format string are the following:

Format specifiers:

<code>%ACTC</code>	Provisioner part of the activation code. Example: 87426549864
<code>%DVID</code>	Device ID, composed of build variant and provisioning ID. Example: 42/2
<code>%MACC</code>	MAC with colons. Example: .00:01:E3:12:34:56.
<code>%MACD</code>	MAC without colons. Example: .0001E3123456.
<code>%%</code>	To represent the percent character.

Example:

The URI-format string:

```
http:// my.server.domain.com/gigaset/cgi/ap?ac=%ACTC
```

and the activation code 12387426549864#0815 leads to a request with the command:

```
GET http://my.server.domain.com/gigaset/cgi/ap.cgi?ac=87426549864
```

Security aspects

- ◆ HTTPS (TLS) is supported by most of the Gigaset VoIP phones. Server root certificates are used.
- ◆ The profile can be encrypted with AES 128 using a specific encryption key.
- ◆ The MAC address is sent during the provisioning process and this MAC address can be used for comparing the address with a data base containing all allowed MAC addresses. Therefore only known phones receive provisioning data.
- ◆ The activation code string contains an optional password.
- ◆ The use of HTTP Digest Authentication is possible.

<i>i</i>	It is not possible to use TLS client certificates.
-----------------	--

Client authentication via HTTP Digest Authentication

Gigaset VoIP phones can use https (HTTP over TLS) but without client certification only. Consequently, there is no information to check the authenticity of the client. Anyone can send a request to the Gigaset redirection server or a provisioning server with a randomly faked MAC address. If the random MAC address is correct, that person would be redirected to a provisioning server and receive an end-user profile configuration.

In order to avoid such a scenario, an additional HTTP digest authentication as defined in RFC 2617 can be used by the phones. The use of HTTP digest authentication is strongly recommended for all networks that are easily eavesdropped, because otherwise a hacker could easily access the user account information by simply recording and replicating the phones GET request to the provisioning server.

Because the HTTP digest authentication is based on a shared secret (which is contained in the *password* component of the activation code), an attacker who wishes to hijack the phone by providing a fake provisioner's URL will still not be able to find out what the shared secret is – because it is never transmitted in clear text.

The HTTP digest authentication algorithm implemented in Gigaset VoIP phones only supports the following context and therefore should be taken into consideration by the provisioner's HTTP implementation:

`algorithm = "MD5" and qop = "auth".`

Furthermore, the `uri=` part of the authorisation header is built in the format described in RFC 2616 (i.e., without the domain portion of the URI), e.g.:

`uri="/phone/cgi/ap?ac=87426549864"`

Updating the configuration data

For transferring changed settings from e.g. a VoIP PBX to a VoIP phone an additional method is necessary for triggering the provisioning procedure, because the real profile download can only be started from the VoIP phone.

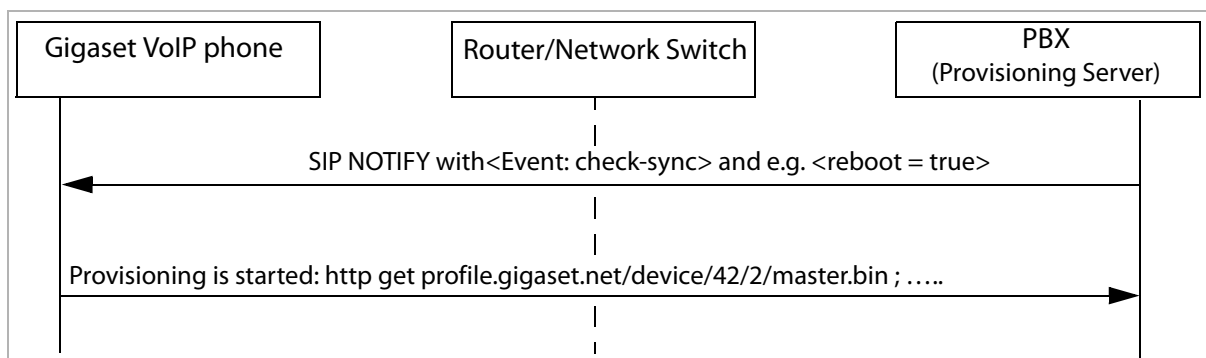
The Gigaset VoIP phones automatically ask for new updates when they are restarted. To initiate a configuration update from the provisioner's side the SIP check-sync mechanism can be used.

SIP check-sync mechanism

The check-sync mechanism allows the initiation of a profile (configuration file) download from remote.

A SIP NOTIFY has to be used for sending the check-sync command. As additional information the protocol includes the switch, if a reboot has to take place (some operating systems need a reboot to make the new settings valid).

The following pictures show the principle of this mechanism and protocol details:



```
Session Initiation Protocol
Request-Line: NOTIFY sip:1002.N510IP@172.29.1.51:5060 SIP/2.0
Method: NOTIFY
Request-URI: sip:1002.N510IP@172.29.1.51:5060
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK69448a0c;rport
From: "Gigaset" <sip:Gigaset@127.0.0.1>;tag=as39b0fcab
To: <sip:1002.N510IP@172.29.1.51:5060>
Contact: <sip:Gigaset@127.0.0.1>
Call-ID: 5f7fe6183735ef2b607bd7da0aaf12d0@127.0.0.1
CSeq: 102 NOTIFY
User-Agent: Gigaset PRO
Max-Forwards: 70
Event: check-sync;reboot=true
Content-Length: 0
```


The XML provisioning file

Configuration data for Gigaset VoIP phones is provided by means of XML files.

Currently, the XML file is used as input for the *gigaset_profile_gen* application which converts the configuration into a binary format comprehensible by Gigaset VoIP phones. In future, the Gigaset VoIP phones will accept XML as input format directly (→ [page 3](#)).

XML files

XML files can be created based on templates that are delivered by Gigaset and stored within the *gigaset* file system. A general template is provided in the subordinate */gigaset/cgi/shop* directory (→ [page 40](#)).

The following template files are available:

<code>template.xml</code>	Template for auto-provisioning using the MAC method.
<code>actc_template.xml</code>	Template for auto-provisioning using an activation code.

Device-specific templates are available in the device's *cgi* subdirectories (→ [page 49](#)). The XML file syntax may be different depending on the device's functionality and date of manufacture. To get information on the correct XML syntax for a specific device please refer to the appropriate template in the device's subdirectory.

Gigaset supplies a template XML file which has to be adapted by the auto-provisioning script to provide the real provisioning data.

XSD schema files

To make sure that the XML file contains only configuration parameters the phone understands, it has to be validated against a fixed schema file (referred to inside the XML file). Schema files are also provided by Gigaset and available in the general *cgi* directory as well as the device-specific *cgi* directories (→ [page 49](#)).

The following schema files are available:

<code>provider.xsd</code>	XML schema file for validating the <i>template.xml</i> file.
<code>actc_provider.xsd</code>	XML schema file for validating the <i>actc_template.xml</i> file.

Configuration parameters

The parameters used for provisioning are described in general on [page 8](#).

Each Gigaset IP phone has a lot of further configuration parameters which can be provided by the XML input file additionally to the provider data.

Details of the parameters mentioned here and a list of all possible configuration parameters are available at <http://wiki.gigaset.com>.

<i>i</i>	<p>The list of modifiable parameters can change if further features are added to the phone in the future. Therefore, please refer to the template files (<i>template.xml</i> / <i>actc_template.xml</i>) and the schema files (<i>provider.xsd</i> / <i>actc_provider.xsd</i>) which are supplied.</p> <p>Attention:</p> <p>You should never change the provider schema file – any changes you might desire in the schema file must be provided by Gigaset. The schema file is the only guarantee that the XML file is compatible with the Gigaset phone you are marketing.</p>
-----------------	---

The XML provisioning file

XML template – example

The following is a short extract from the *template.xml* file. To view the total file or the *actc_template.xml* file you can open it from the *gigaset/cgi/shop* directory (→ [page 49](#)).

The template provided by Gigaset for the XML file contains two types of parameters:

- ◆ A large set of parameters which have to be adapted for a specific provider.
- ◆ A smaller set of parameters intended to be adapted (by the provisioner's *ap.cgi* script) to tailor the file for an individual end customer. The latter set of parameters has been highlighted in the fragment shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ProviderFrame xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="provider.xsd">
  <Provider>
    <MAC_ADDRESS value="insert MAC_ADDRESS here"/>
    <VERSION value="insert VERSION here"/>
    <PROFILE_NAME class="string" value="insert PROFILE_NAME here"/>
    <S_SIP_LOGIN_ID class="string" value="insert S_SIP_LOGIN_ID here"/>
    <S_SIP_PASSWORD class="string" value="insert S_SIP_PASSWORD here"/>
    <S_SIP_USER_ID class="string" value="insert S_SIP_USER_ID here"/>
    <S_SIP_DOMAIN class="string" value="192.168.2.1"/>
    <!-- optional (and obsolete)
      <S_SIP_REALM class="string" value=""/>
    -->
    <S_SIP_SERVER class="string" value="192.168.2.1"/>
    <I_SIP_SERVER_PORT class="integer" value="5060"/>
    <S_SIP_REGISTRAR class="string" value="192.168.2.1"/>
    <I_SIP_REGISTRAR_PORT class="integer" value="5060"/>
    <B_SIP_USE_STUN class="boolean" value="false"/>
    <S_STUN_SERVER class="string" value=""/>
    <I_STUN_SERVER_PORT class="integer" value="3478"/>
    <!-- optional
      <I_NAT_REFRESH_TIME class="integer" value="20"/>
    -->
    <I_OUTBOUND_PROXY_MODE class="string" value="auto"/>
    <S_OUTBOUND_PROXY class="string" value="192.168.2.1"/>
    <I_OUTBOUND_PROXY_PORT class="integer" value="5060"/>
    <I_RE_REGISTRATION_TIMER class="integer" value="180"/>
    <I_RE_STUN_TIMER class="integer" value="240"/>
    <!-- optional; loudness values: 0 = normal, 1 = loud, 255 = low
      <I_LOUDNESS_1 class="integer" value="0"/>
    -->
    <!-- more optional parameters for account 1
      <S_SIP_DISPLAYNAME class="string" value="anything"/>
    -->
    <S_SIP_PROVIDER_NAME class="string" value="PBX"/>
    <!-- more optional parameters for account 1
      <I_SIP_ACCOUNT_MT_RCV_1 class="integer" value="63"/>
      <I_SIP_ACCOUNT_MT_SND_1 class="integer" value="1"/>
    -->
    <B_SIP_ACCOUNT_IS_ACTIVE_1 class="boolean" value="true"/>
    <!-- more optional parameters for account 1
      <S_VOIP_NET_AM_NUMBER_1 class="string" value=""/>
      <B_VOIP_NET_AM_ENABLED_1 class="boolean" value="true"/>
    -->
```

```

    <!-- optional; codec list for account 1 with 5 elements, values: 0 = PCMU G.711 µ law,
1 = PCMA G.711 a law, 2 = G726, 3 = G729, 4 = G726 AAL2 , 5 = G722
    <I_SIP_PREFERRED_VOCODER class="array">
        <ARRAYELEMENT class="integer" value="5"/>
        <ARRAYELEMENT class="integer" value="1"/>
        <ARRAYELEMENT class="integer" value="0"/>
        <ARRAYELEMENT class="integer" value="2"/>
        <ARRAYELEMENT class="integer" value="3"/>
    </I_SIP_PREFERRED_VOCODER>
-->
    <!-- optional: parameters for account 2
    ...
-->
    <!-- optional; codec list for account 2 with 5 elements, values: 0 = PCMU G.711 µ law,
1 = PCMA G.711 a law, 2 = G726, 3 = G729, 4 = G726 AAL2 , 5 = G722
    ...-->
    <!-- optional: parameters for account 3
    ...
--> ...
    <!--Bit-Masks for I_DTMF_TX_MODE_BITS: Audio=1, RFC2833=2, SIP-INFO=4-->
    <I_DTMF_TX_MODE_BITS class="integer" value="1"/>
    <I_DTMF_TX_RTP_PAYLOAD_TYPE class="integer" value="101"/>
    <B_SHOW_USERID_DURING_WIZARD class="boolean" value="true"/>
    <!-- optional parameters:
        <S_DATA_SERVER class="string" value="gigaset.siemens.com/gigaset"/>
        <S_EMAIL_SERVER class="string" value="192.168.2.55"/>
        <S_MESSENGER_SERVER class="string" value="192.168.2.55"/>
        <I_MESSENGER_SERVER_PORT class="integer" value="1"/>
        <S_TIME_NTP_SERVER class="string" value="192.168.2.55"/>
        <B_REDIRECT_EMERGENCY_TO_PSTN class="boolean" value="true"/>
    -->
    <!-- optional; allows to add a text to the UA header in SIP messages:
        <S_USERAGENT_STRING class="string" value="additional_text"/>
    -->
    <!-- optional-->
        <B_SIP_SHC_ACCOUNT_IS_ACTIVE class="boolean" value="true"/>
    <!-- optional online phonebook settings -->
        <S_TDS_SERVICE_URL class="string" value=""/>
        <S_TDS_LOGIN class="string" value=""/>
        <S_TDS_LOGIN_PASS class="string" value=""/>
        <S_TDS_TXT_MENU class="string" value=""/>
        <S_TDS_TXT_HEADER class="string" value=""/>
        <S_TDS_TXT_SEARCH_1 class="string" value=""/>
        <S_TDS_TXT_SEARCH_2 class="string" value=""/>
        <S_TDS_TXT_DEP_MENU class="string" value=""/>
        <S_TDS_TXT_DEP_HEADER class="string" value=""/>
        <S_TDS_TXT_DEP_SEARCH_1 class="string" value=""/>
        <S_TDS_TXT_DEP_SEARCH_2 class="string" value=""/>
        <S_TDS_TXT_PTD_MENU class="string" value=""/>
        <S_TDS_TXT_PTD_HEADER class="string" value=""/>
        <S_TDS_TXT_PTD_SEARCH_1 class="string" value=""/>
        <S_TDS_TXT_PTD_SEARCH_2 class="string" value="0"/>
        <I_TDS_CAPABILITIES class="integer" value="0"/>
        <S_TDS_ISO_3166_1 class="string" value=""/>
        <B_TDS_TRANSMIT_MAC_ADDRESS class="boolean" value="1"/>
    <!-- optional online phonebook settings end-->
</Provider>
</ProviderFrame>

```

The XML provisioning file

Example: XML file fragment for sipgate

In the case of the MAC method:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ProviderFrame xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="mac_provider.xsd">
  <Provider>
    <MAC_ADDRESS value="filled out by the ap script"/>
    <VERSION value="insert VERSION here"/>
    <PROFILE_NAME class="string" value="insert PROFILE_NAME here"/>
    <S_SIP_LOGIN_ID class="string" value="insert S_SIP_LOGIN_ID here"/>
    <S_SIP_PASSWORD class="string" value="insert S_SIP_PASSWORD here"/>
    <S_SIP_USER_ID class="string" value="insert S_SIP_USER_ID here"/>
    <S_SIP_PROVIDER_NAME class="string" value="Sipgate"/>
    <S_SIP_DOMAIN class="string" value=""/>
    <S_SIP_REALM class="string" value="sipgate.de"/>
    <S_SIP_SERVER class="string" value="sipgate.de"/>
    <I_SIP_SERVER_PORT class="integer" value="5060"/>
    <S_SIP_REGISTRAR class="string" value="sipgate.de"/>
    <I_SIP_REGISTRAR_PORT class="integer" value="5060"/>
    ...
  </Provider>
</ProviderFrame>
```

In the case of the activation code method:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ProviderFrame xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="actc_provider.xsd">
  <Provider>
    <ACTIVATION_CODE value="filled out by the ap script"/>
    <VERSION value="insert VERSION here"/>
    <PROFILE_NAME class="string" value="insert PROFILE_NAME here"/>
    <S_SIP_LOGIN_ID class="string" value="insert S_SIP_LOGIN_ID here"/>
    <S_SIP_PASSWORD class="string" value="insert S_SIP_PASSWORD here"/>
    <S_SIP_USER_ID class="string" value="insert S_SIP_USER_ID here"/>
    <S_SIP_PROVIDER_NAME class="string" value="Sipgate"/>
    <S_SIP_DOMAIN class="string" value=""/>
    <S_SIP_REALM class="string" value="sipgate.de"/>
    <S_SIP_SERVER class="string" value="sipgate.de"/>
    <I_SIP_SERVER_PORT class="integer" value="5060"/>
    <S_SIP_REGISTRAR class="string" value="sipgate.de"/>
    <I_SIP_REGISTRAR_PORT class="integer" value="5060"/>
    ...
  </Provider>
</ProviderFrame>
```

How to manage the VERSION parameter

The VERSION parameter in the XML file is somewhat special because it is not really a configuration item, but instead is used by the phone to detect whether there have been any changes in the configuration since the last time it was changed.

The parameter is a time string with the following mandatory format:

`ddmmyyhhmm`

where *dd*, *mm*, *yy*, *hh* and *mm* represent the decimal numeric values for day, month, year, hours and minutes respectively. Because the phone converts this string into an equivalent integer value, it is important to choose valid date & time values.

In the phone, the change detection algorithm works as follows:

- ◆ Whenever the phone has requested and processed an auto-provisioning file, it converts the string contained in the VERSION parameter into an integer, which is saved in the non-volatile RAM.
- ◆ In future, the phone compares this saved integer with the value of the integer calculated from the VERSION parameter contained in any newly requested auto-provisioning file.
- ◆ As long as the two integers are equal, the phone assumes that the auto-provisioning file is identical with the old one, and will ignore it.
- ◆ If, however, the integers are different, the phone assumes that a new auto-provisioning file has been issued and updates its configuration accordingly, replacing its saved copy of the integer with the one calculated from the new file.

For the auto-provisioning system, this means that whenever it wishes the phone to update its configuration, it must generate a new value of the VERSION parameter.

The design assumes that the auto-provisioning system will store this version string in the database entry belonging to a given end-customer, thus indicating the last time the configuration for that particular end-customer was changed.

<i>i</i>	Changing the VERSION string too often (e.g. on a daily basis or even more often) is not to be recommended, as it will force the phone to update its non-volatile memory unnecessarily. As this memory is implemented using FLASH or EEPROM technology, which has a limited number of write cycles, the phone's useful lifetime will suffer!
-----------------	---

Setting up an own provisioning server

The general procedure for setting up auto-provisioning of Gigaset IP phones is as follows:

- ◆ Installing a Linux server

Currently, systems with i386 architecture (and newer, compatible ones) are supported. The installation file is distributed as a Linux RPM file, i.e. the Linux system used, must support this packaging format.
- ◆ Preparing the Linux server
 - Installing an Apache HTTP server

The Apache HTTP server is an open source web server for UNIX and Windows systems. It is used to handle the HTTP requests of the Gigaset IP phones.
 - Adapting the web server configuration file for connecting the Gigaset auto-provisioning script

To enable the auto-provisioning script (→ [page 41](#)) to be executed successfully the web server configuration file has to be adapted as follows: Add the *gigaset/cgi* directory to the Apache directories.
 - Installing PHP script language

PHP is a general scripting language designed for web development to produce dynamic web pages. The code is interpreted by a web server with a PHP processor module which generates the resulting web page. It also has evolved to include a command-line interface capability and can be used in standalone graphical applications.
 - Setting up a MySQL server (optional)

MySQL is the recommended instrument for providing the auto-provisioning script with the user-related data.
- ◆ Installing the auto-provisioning application
 - Preparing the file system
 - Installing the application
- ◆ Creating the auto-provisioning script or adapting the sample script (→ [page 41](#))

Installing the auto-provisioning application

Preparing the file system

Before installing the auto-provisioning application the file system has to be prepared. i.e. you have to create the necessary directories and copy the files into the file system structure.

Creating directories

- ▶ Create a new directory *gigaset/cgi/shop* within the web server document path, e.g.

/var/www/html:

<code>cd /var/www/html/</code>	Change to the web server document directory
<code>mkdir gigaset</code>	Create a new directory <i>gigaset</i>
<code>cd gigaset</code>	Change to the <i>gigaset</i> directory
<code>mkdir cgi</code>	Create a new directory <i>cgi</i>
<code>cd cgi</code>	Change to the <i>cgi</i> directory
<code>mkdir shop</code>	Create a new directory <i>shop</i>
<code>cd /</code>	Return to the <i>root</i> directory

Setting up an own provisioning server

Copying the files to the provisioning server

- Copy the following files from the CD into the appropriate directories:

File	Copy into the directory:
ap	→ var/www/html/gigaset/cgi
key	→ var/www/html/gigaset/cgi/shop
provider.xsd	→ var/www/html/gigaset/cgi/shop
template.xml	→ var/www/html/gigaset/cgi/shop
gigaset_profil_gen-<version>.i386.rpm	→ /usr/bin



Please note, that the web server path may be different in other Linux and Apache installations.

Required libraries

Besides the usual *libc* libraries required by any standard Linux application, *gigaset_profile_gen* needs the *libxml2* library.

You can download the latest version of *libxml2* from

<http://xmlsoft.org>

The RPM installation will inform you if the required libraries are not installed on your system.

Installing the *gigaset_profile_gen* application

To install the application perform the following steps:

- Change to the */usr/bin* directory.

```
cd /usr/bin
```

- Install the software.

```
rpm -Uhv gigaset_profile_gen-<version>.i386.rpm
```

To enable the Perl script to start the generator, you need to create a link to the application in the *shop* directory.

- Change to the *shop* directory.

```
cd /var/www/html/gigaset/cgi/shop
```

- Create the link

```
ln -s /usr/bin/gigaset_profile_gen
```

Setting the access rights for the auto-provisioning files and script

When the package is installed, the following files are available in the */gigaset* directory located in the HTTP server document directory.

File	Description
<code>cgi/ap</code>	Auto-provisioning script.
<code>cgi/shop/template.xml</code>	Template for auto-provisioning using the MAC method containing the configuration data for the phone.
<code>cgi/shop/actc_template.xml</code>	Template for auto-provisioning using an activation code containing the configuration data for the phone.
<code>cgi/shop/gigaset_profile_gen</code>	Auto-provisioning tool.
<code>cgi/shop/key</code>	Contains the secret key used to encrypt the configuration file to be sent to the phone.
<code>cgi/shop/provider.xsd</code>	XML schema file required by the <code>gigaset_profile_gen</code> application in order to validate the <code>template.xml</code> file.
<code>cgi/shop/actc_provider.xsd</code>	XML schema file required by the <code>gigaset_profile_gen</code> application in order to validate the <code>actc_template.xml</code> files.

To enable the generator to run successfully you need to set the access rights for the files in the *cgi* and *shop* directories correctly.

<i>i</i>	<p>Access rights are set using the <code>chmod 755</code> command.</p> <p>This sets the access rights as follows: Read, write and execute rights for the owner (root), read and execute for all other user users.</p>
-----------------	--

- Change the directory.

```
cd <HTTP server directory>/gigaset/cgi
```
- Change the access rights for the auto-provisioning script.

```
chmod 755 ap
```
- Change the directory.

```
cd <HTTP server directory>/gigaset/cgi/shop
```
- Change the access rights for the *key*, *template* and *provider* files.

```
chmod 755 key
chmod 755 template.xml
chmod 755 provider.xsd
```


Auto-provisioning example script

The auto-provisioning script (*ap*) is responsible for the following tasks:

- 1 Looking for the given MAC address or activation code and finding the corresponding user and profile data.
- 2 Editing a copy of the template XML file with the given user data.
- 3 Running the *gigaset_profile_gen* program which generates the desired configuration file (→ [page 47](#)).
- 4 Sending the configuration file to the requesting phone.

Gigaset provides an example script written in Perl because it has powerful text manipulation instructions and is the usual choice for writing CGI scripts. This script can be found in the */gigaset/cgi* directory.

Our simple script does all of the above. However, our task 1 is very primitive, because it maps only a few MAC addresses to fixed users. In real life, this task will probably be implemented by a database application.

Here is the sample script. It is stored in the */gigaset/cgi* directory:

```
#!/usr/bin/perl
#*****
#** Copyright (c) 2008
#** Gigaset Communications GmbH
#** Author(s): LC, HJL / GC PD D SD RP
#
#       This is a DEMO CGI script that shows how to interface
#       the Gigaset application 'gigaset_profile_gen' inside
#       a CGI application written in Perl.
#
# ** Redistribution and use in source and binary forms, with or without
# ** modification, are permitted provided that the following conditions
# ** are met:
#
# ** 1. Redistributions of source code must retain the above copyright
# ** notice, this list of conditions and the following disclaimer.
# ** 2. Redistributions in binary form must reproduce the above copyright
# ** notice, this list of conditions and the following disclaimer in the
# ** documentation and/or other materials provided with the distribution.
# **
# ** THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR
# ** IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
# ** WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# ** DISCLAIMED.
# ** IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
# ** INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
# ** NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# ** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# ** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# ** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
# ** THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#*****
use CGI;
$q = new CGI();
use File::Temp qw/ tempfile /;
use File::stat;
# -----

# 0. Change to 'shop' directory where actual processing takes place:
chdir './shop' or die "Can't cd to shop: ${\n}";
```

```
# 1. Get the requesting device's MAC address or activation code

$MacAddress = $q->param(mac);
$ActivationCode = $q->param(ac);

if ($MacAddress eq '' and $ActivationCode eq '')
{
    # no parameter, assume MAC-method
    $MacAddress = 'FF:FF:FF:FF:FF:FF';# for testing
}

# 2. Associate a user profile depending on the provisioning method

if ($MacAddress ne '')
{
    # 2a In case of MAC Autoprovisioning
    # print("MAC method!!\n");
    # simple check for a valid MAC-address (with or without colons)
    if(length($MacAddress) != 17)
    {
        if(length($MacAddress) != 12)
        {
            die "invalid MAC-format";
        }
        # convert a MAC-address without colons (e.g. 123456789012)
        # to a MAC-address with colons (e.g. 12:34:56:78:90:12)
        $MacAddress=~ s/..\B/$&:/gi
    }
    # Associate requestor's MAC address with a user profile:
    # Create a temporary xml file from the template:
    ($InFileH, $TheXmlInputFile) = tempfile('TmpXXXXXX', '.xml');
    close($InFileH);
    if (&Customize_For_MAC_Method('template.xml', $TheXmlInputFile, $MacAddress) != 0)
    {
        die "Customize MAC subroutine failed";
    }
}
elseif($ActivationCode ne '')
{
    # 2b In case of Provisioning using Activation Code
    # print("Activation Code method!!\n");

    # Associate requestor's Activation Code with a user profile:

    # Create a temporary xml file from the template:
    ($InFileH, $TheXmlInputFile) = tempfile('TmpXXXXXX', '.xml');
    close($InFileH);
    if (&Customize_For_ActivationCode_Method('actc_template.xml', $TheXmlInputFile,
$ActivationCode) != 0)
    {
        die "Customize Activation Code subroutine failed";
    }
}

# 3.Convert the XML user profile created in 2) into an encrypted
#   binary profile:

# Create a temporary output file in the current directory
($OutputFileH, $TheOutputFileName) = tempfile('TmpXXXXXX', '.bin');
close($OutputFileH);
```

Setting up an own provisioning server

```
# Important: call generator with the -s argument, so it won't output
# anything on stdout!
# Moreover, if you need to generate a non encrypted autoprovisioning file,
# enter the option '-noencrypt', as shown below:
# $ExitStatus = system ("./gigaset_profile_gen $TheXmlInputFile $TheOutputFileName
# -s -noencrypt");
$ExitStatus = system ("./gigaset_profile_gen $TheXmlInputFile $TheOutputFileName -s");
print $ExitStatus
unlink ($TheXmlInputFile);
if ($ExitStatus != 0)
{
    die "Could not generate the binary profile";
    unlink ($TheOutputFileName);
}

# 4. Send the config file to the requesting device:

open (OUTFILE, $TheOutputFileName) or die "The bin file to output could not be found";
print "Content-type: application binary\n\n";

while (1)
{
    $ByteCnt = read (OUTFILE, $Bytes, 100);
    print $Bytes;
    if ($ByteCnt != 100)
    {
        last;
    }
}

close OUTFILE;
unlink ($TheOutputFileName);

#####
### Subroutine to customize the XML template for MAC-Autoprovisioning method #####
#####

sub Customize_For_MAC_Method
{
    my($TemplateFileName, $OutputFileName, $MacAddr) = @_;

    # This subroutine is a *very* primitive example of how to customize
    # the xml template file for a specific end customer.
    # This examples assumes that there are only 4 end customers to
    # service :-))
    # In the real world, this subroutine would interface to a huge
    # database of users who are identified by their MAC address!

    #####
    ##### Customer "Database" for MAC method #####
    #####

    #
    # Customer 1 Customer 2 Customer 3 #
    my(@UserVersion) = ('3011061455', '3011061455' '3011061455' );
    my(@UserMacAddr) = ('00:01:E3:75:F1:72', '00:01:E3:67:60:77', '00:0A:5E:55:02:8F');
    my(@UserLoginId) = ('1234567', '1234567', '1234568' );
    my(@UserPassword)= ('abcdef', 'aa123456', 'bb123456' );
    my(@UserId) = ('1234567', '1234567', '1234568' );
    my(@UserProfName)= ('d_default_de.bin', 'd_default_de.bin', 'd_default_de.bin');
    #####
    #####
```

```
# find user associated with given MAC address:
my $UserIndex = 0;
while ($UserIndex <= $#UserMacAddr)
{
    if (@UserMacAddr[$UserIndex] eq $MacAddr)
    {
        last;
    }
    $UserIndex++
}
if ($UserIndex > $#UserMacAddr)
{
    return -1;          # Failure: no match for MAC addr found
}

# load the whole template:
open (INFILE, $TemplateFileName) or die "Could not open $TemplateFileName";
my($st) = stat($TemplateFileName) or die "No file $!";
read (INFILE, my $Block, $st->size);
close(INFILE);

# substitute the place holders in the template with the
# actual values assigned to the end customer:
$Block =~ s/insert MAC ADDRESS here/@UserMacAddr[$UserIndex]/g;
$Block =~ s/insert VERSION here/@UserVersion[$UserIndex]/g;
$Block =~ s/insert PROFILE_NAME here/@UserProfName[$UserIndex]/g;
$Block =~ s/insert S_SIP_LOGIN_ID here/@UserLoginId[$UserIndex]/g;
$Block =~ s/insert S_SIP_PASSWORD here/@UserPassword[$UserIndex]/g;
$Block =~ s/insert S_SIP_USER_ID here/@UserId[$UserIndex]/g;

# store the customized xml file:
open (OUTFILE, ">$OutputFileName") or die "Could not open $OutputFileName";
print OUTFILE $Block;
close(OUTFILE);
return 0;          # Success
}

#####
### Subroutine to customize the XML template for Activation Code Provisioning ###
### method #####
#####

sub Customize_For_ActivationCode_Method
{
    my($TemplateFileName, $OutputFileName, $ActCode) = @_;

    # This subroutine is a *very* primitive example of how to customize
    # the xml template file for a specific end customer.
    # This examples assumes that there are only 4 end customers to
    # service :-))
    # In the real world, this subroutine would interface to a huge
    # database of users who are identified by their Activation Code!
    ##### Customer "Database" for Activation code method #####
    #####

    #
    # Customer 1 Customer 2 Customer 3 #
    my (@UserActCode) = ('000133676069', '000113676077', '000754550284' );
    my (@UserLoginId) = ('6260854', '1234567', '1234568' );
    my (@UserPassword)= ('dd1234', 'aa123456', 'bb123456' );
    my (@UserId) = ('6260854', '1234567', '1234568' );
    my (@UserVersion) = ('3011061455', '3011061455', '3011061455' );
    my (@UserProfName)= ('d_default_de.bin', 'd_default_de.bin', 'd_default_de.bin');
    #####
    #####

```

Setting up an own provisioning server

```
# find user associated with given Activation Code:
my $UserIndex = 0;
while ($UserIndex <= $#UserActCode)
{
    if (@UserActCode[$UserIndex] eq $ActCode)
    {
        last;
    }
    $UserIndex++
}
if ($UserIndex > $#UserActCode)
{
    return -1;      # Failure: no match for Activation Code found
}

# load the whole template:
open (INFILE, $TemplateFileName) or die "Could not open $TemplateFileName";
my($st) = stat($TemplateFileName) or die "No file $!";
read (INFILE, my $Block, $st->size);
close(INFILE);

# substitute the place holders in the template with the
# actual values assigned to the end customer:
$Block =~ s/insert ACTIVATION_CODE here/@UserActCode[$UserIndex]/g;
$Block =~ s/insert VERSION here/@UserVersion[$UserIndex]/g;
$Block =~ s/insert PROFILE_NAME here/@UserProfName[$UserIndex]/g;
$Block =~ s/insert S_SIP_LOGIN_ID here/@UserLoginId[$UserIndex]/g;
$Block =~ s/insert S_SIP_PASSWORD here/@UserPassword[$UserIndex]/g;
$Block =~ s/insert S_SIP_USER_ID here/@UserId[$UserIndex]/g;

# store the customized xml file:
open (OUTFILE, ">$OutputFileName") or die "Could not open $OutputFileName";
print OUTFILE $Block;
close(OUTFILE);
return 0;# Success
}
```

Adapting the ap script for MySQL database input (optional)

Usually, the user data is imported from a database. The following is an example for implementing MySQL access within the *ap* script.

When the *ap* script should access a MySQL database for importing the user-related data adapt the script analogous to the following:

- At the beginning of the script change the following section:

```
use CGI
$q = new CGI();
use File::Temp qw/ tempfile /;
use File::stat;
```

- Add the following lines:

```
### use Mysql for database connection
use Mysql;
### Encode will be used for encoding and decoding results from database
use Encode;
```

- Add another sub routine to the script. Ensure that database, table and field names match the settings defined for the database.

```
#####
### Subroutine to customize the XML template for MAC-Autoprovisioning method   ###
###                               with Mysql Connection                       ###
#####

sub Customize_For_MAC_Method_MYSQL
{

my($TemplateFileName, $OutputFileName, $MacAddr) = @_ ;

#####
# MySQL Connection
#####

# Mysql Configuration
$host="localhost";
$database="GigasetAutoProvisioning";
$tablename ="UserConnectionValues";
$user="root";
$pw="gigaset";
# MySQL Connet
$connect = Mysql->connect($host,$database,$user,$pw);
#Select DB
$connect ->selectdb($database);
#Define a Mysql Query
$myquery ="SELECT *FROM $tablename WHERE MAC = \'$MacAddress\' LIMIT 0,30";
#Execute the Query Function
$execute =$connect->query($myquery);

$MAC="";
$Version ="";
$Profil ="";
$LOGINIG ="";
$USERID ="";
$PW="";

# Turn results from DB into Variables
while (@results = $execute->fetchrow())
{
$MAC = $results[0];
# convert UTF8 (DB) to Iso-8859-1(Latin) for the template(Endconfinig Iso-8859-1
$octets = encode("utf8", $Mac);
$Mac = decode("iso-8859-1",$octets);
$Version = $results[1];
$octets = encode("utf8", $Version);
$Version = decode("iso-8859-1",$octets);
$Profil =$results[2];
$octets = encode("utf8", $Profil);
$Profil = decode("iso-8859-1",$octets);
$LOGINID =$results[3];
$octets = encode("utf8", $LOGINID);
$LOGINID = decode("iso-8859-1",$octets);
$USERID =$results[4];
$octets = encode("utf8", $USERID);
$USERID = decode("iso-8859-1",$octets);
$PW= $results[5];
$octets = encode("utf8", $PW);
$PW = decode("iso-8859-1",$octets);
}

# load the whole template:
open (INFILE, $TemplateFileName) or die "Could not open $TemplateFileName";
my($st) = stat($TemplateFileName) or die "No file $!";
read (INFILE, my $Block, $st->size);
close(INFILE);

# substitute the place holders in the template with the
# actual values assigned to the end customer:
```

Setting up an own provisioning server

```
$Block =~ s/insert MAC_ADDRESS here/$MAC/g;
$Block =~ s/insert VERSION here/$Version/g;
$Block =~ s/insert PROFILE_NAME here/$Profil/g;
$Block =~ s/insert S_SIP_LOGIN_ID here/$LOGINID/g;
$Block =~ s/insert S_SIP_PASSWORD here/$PW/g;
$Block =~ s/insert S_SIP_USER_ID here/$USERID/g;
# store the customized xml file:
open (OUTFILE, ">$OutputFileName") or die "Could not open $OutputFileName";
print OUTFILE $Block;
close(OUTFILE);
return 0;
# Success
}
```

i

Database entries have to be converted from Unicode to UTF-8 to be processed by the auto-provisioning tool successfully.

Testing the installation

When the installation and adjustment are finished, you should test to see if the auto-provisioning process works properly. You can use a standard web browser to test the correct setup of the auto-provisioning system.

- ▶ To check if your installation has been successful enter the following URL in your browser:

`http://<provisioning server>/gigaset/cgi/ap?mac=<MAC address>`

`<provisioning server>` Domain name or IP address of the provisioning server

`<MAC address>` MAC address of an IP phone known by the `ap` script.

Example:

Your host has the domain `cfg.provisioner.com`.

- ▶ Enter the following URL:

`http://cfg.provisioner.com/gigaset/cgi/ap?mac=00:11:22:33:44:55`

- ◆ If everything is set up properly, the browser offers you a binary file. The file is encrypted, so it should be quite unintelligible when downloaded and then opened with a hex editor.
- ◆ If the web browser times out, or returns "404", you'll have to verify your setup; make sure you have created the correct directory.
- ◆ If the web browser returns "500", the script has failed, e.g. because an unknown MAC address was entered.

You can then test your auto-provisioning system with the Gigaset phone.

- ▶ Power-up the phone and wait a few minutes for the phone to generate the auto-configuration request.
- ▶ We suggest you use a trace program like Ethereal®, Wireshark etc. to examine the phone requests and the response from your HTTP server.
- ▶ Open the phone's web page to check whether the configuration parameters you programmed have been successfully stored in the phones configuration memory.

The gigaset_profile_gen application

gigaset_profile_gen is a console application intended to be called from a CGI script. Chapter [Auto-provisioning example script](#) (→ [page 41](#)) describes how this script has to be designed and provides an *ap* example script.

Synopsis

```
gigaset_profile_gen XML_INPUT_FILE OUTPUT_FILE [-s] [-noencrypt]
```

Description

Generates the encrypted configuration file from the given *XML_INPUT_FILE* and places it in *OUTPUT_FILE*.

Mandatory arguments

<i>XML_INPUT_FILE</i>	File containing the configuration data for the phone to be provisioned (→ page 34).
<i>OUTPUT_FILE</i>	Binary, encrypted version of the configuration, ready to be sent to the phone.

Optional arguments

<i>-s</i>	<i>Silent operation:</i> suppresses any output by the program to <i>STDOUT</i> ; when called from a CGI application, this argument must be used, because the CGI standard uses <i>STDOUT</i> to gather the response that HTTP will send to the client. Without this parameter, the HTTP response would contain unwanted text, such as the sign-on message generated by the program!
<i>-noencrypt</i>	<i>Don't encrypt the output file</i> This argument must only be used if the phone does not require a secret key – otherwise, it will not understand the configuration file!

Remarks

- ◆ In order to encrypt the configuration file, *gigaset_profile_gen* usually needs a secret key, which is contained in a file named *key*. This file must be located in the same directory as the application. The secret key must coincide with the one used in the phone. Therefore, the key file is customised by Gigaset for the provisioner.
In certain closed network scenarios, the provisioner might desire to work with an unencrypted configuration file. In this case, the phone has to be customised by Gigaset accordingly ("no secret key"), and the argument *-noencrypt* shown above must be given when invoking the *gigaset_profile_gen* application for building the file.
Note that for security reasons Gigaset recommends using encrypted configuration files only.
- ◆ To make sure that the *XML_INPUT_FILE* contains only configuration parameters the phone understands, the *gigaset_profile_gen* validates it against a fixed schema file (referred to inside the XML file). This schema file is provided by the Gigaset. It must be located in the same directory as the *XML_INPUT_FILE*.
If the validation fails, the application returns an error.

Return values

gigaset_profile_gen returns 0 on success. All other values indicate an error. Error messages are sent to *STDERR*. In the case of errors, an empty *OUTPUT_FILE* is generated.

File system structure

Usually, the configuration data for the VoIP phones is provided by Gigaset, taking a determined data structure within the HTTP server file system into consideration.

If the primary URL is changed (e.g. via the DHCP option 114 or via SIP NOTIFY, → [Page 22](#)), the data structure has to be stored at the HTTP server according to the new URL. When using an own provisioning server the data has to be provided by the provisioner itself.

The data structure is as follows:

Within the HTTP server file system the *gigaset* directory is created containing a subdirectory for each device type. The numbers used as directory names correspond to the device or the device variant of the Gigaset IP phones.

gigaset/41	41 = Gigaset DX800A
42	42 = Gigaset C610 IP/N300 (1), Gigaset N510 IP PRO (2)
60	60 = Gigaset DE900 IP PRO
61	61 = Gigaset DE700 IP PRO
62	62 = Gigaset DE410 IP PRO
63	63 = Gigaset DE310 IP PRO
70	70 = Gigaset N720 DM PRO
71	71 = Gigaset N720 IP PRO

For each device variant a master file (*master.bin*) is stored. In addition all files that are necessary for the device, e.g. firmware, language files for the Web GUI, help files, texts for the handset UI, auto-provisioning links, are available.

Gigaset provides the provisioner, e.g. the PBX manufacturer, with the required data structure. The auto-provisioning example application (*ap* script, *gigaset_profile_gen*, XML templates, etc.) is copied to the subdirectories of the device variants.

Provisioning is automatically processed at the location where the example application is stored. This is achieved with the use of wildcards for the server URL, MAC address, etc. in the auto-provisioning URL.

Example for Gigaset N510 IP PRO:

```
gigaset/42/2/ cgi/shop/gigaset_profile_gen
               cgi/shop/key
               cgi/shop/merkur_mac_template.xml
               cgi/shop/merkur_mac_template.xsd
               cgi/ap
               c_0811101406_eng.bin
               c_0811101406_fre.bin
               c_0811101406_ger.bin
               ...
               d_default_de.bin
               l_0811101406.bin
               master.bin
               pde_0811101406_eng.bin
               pde_0811101406_fre.bin
               pde_0811101406_ger.bin
               ...
```

- ◆ The variant/provisioning ID is 42/2
- ◆ The URL assigned using SIP NOTIFY is <http://192.168.1.100/provsioning/gigaset>
The data structure has to be stored within this directory.
- ◆ The total path to the file *master.bin* is:
<http://192.168.1.100/provsioning/gigaset/42/2/master.bin>

- ◆ This is used to load the auto-provisioning URL, e.g.
<http://%DURL/%DVID/cgi/ap?mac=%MACD>
- ◆ The place holders are replaced in sequence:
%DURL="http://192.168.1.100/provsioning/gigaset"
%DVID="42/2"
%MACD=12:34:56:78:90:12
- ◆ The VoIP phone then launches the complete auto-provisioning request:
<http://192.168.1.100/provsioning/gigaset/42/2/cgi/ap?mac=12:34:56:78:90:12>

Hence, the auto-provisioning request is processed at the correct location by means of wildcards without previously knowing the URLs of the auto-provisioning servers.

Of course, instead of using wildcards, you can also apply a fixed URL for the auto-provisioning server. But for the first set-up the procedure described above is helpful.

Index

A

Access rights	40
Activation code	
authentication	27
auto-provisioning method.....	27
format	27
generating	27
Gigaset part	27
provisioner part	27
security hints	31
ap script see auto-provisioning script	
Apache HTTP server	
configuration file.....	38
directories.....	38
installing	38
autoprov.checkDevice.....	19
autoprov.checkDeviceList.....	20
autoprov.deregisterDevice.....	16
autoprov.deregisterDeviceList.....	21
autoprov.listDevices.....	18
autoprov.listDevices, all	17
autoprov.registerDevice.....	15
Auto-provisioning	
general procedure	38
template.....	40
URL	49
via activation code	27
Auto-provisioning script	41
example	41
MySQL connection.....	45
Auto-provisioning via activation code	
message flow.....	29
Auto-provisioning via MAC address	
message flow.....	29
Auto-provisioning, definition	3
Auto-provisioning script.....	26, 30

C

check-sync.....	32
chmod	40
Command	
check-sync	32
chmod.....	40
Configuration file	
Apache	38
encryption key	40
Configuration update	32

D

Database	
character format UTF-8	47
Deregistering	
a list of devices	21
Deregistering a device	
via web UI	11

via XML-RPC call	16
Device	
deregistering a list of.....	21
listing all at a specific provider registered	18
listing all registered	17
listing all registered at a specific	
provider.....	19
listing all registered at a specific provider	20
DHCP option 114	6, 23
dhcp_url	23
Directory	
cgi	38
gigaset.....	38
HTTP server.....	40
shop	38

E

End-user, role in provisioning process	4
--	---

F

File system	
preparing for installation.....	38
File system structure.....	49
device variants	49
Files, provisioning	33
for XML configuration file	33

G

Gigaset	
redirection server	5
update server	5
gigaset directory	38
Gigaset wiki.....	33
gigaset_profile_gen	
arguments.....	48
installing.....	39
link to shop directory	39
required libraries	39
return values	48
usage	48
Gigaset, role in provisioning process.....	4
/gigaset/cgi.....	38
/gigaset/cgi/shop.....	38

H

HTTP Digest Authentication.....	31
HTTP requests	26, 30

I

Installation test	47
Installing	
Apache HTTP server.....	38
gigaset_profile_gen.....	39
PHP	38
IP phone set-up	
manually	9

Index

K

Key 40

L

libc library 39

Libraries..... 39

libxml2 library 39

List devices
 via web user interface..... 11

M

MAC address..... 23

MAC ID..... 23

MAC-based auto-provisioning 23

Main menu 10

main menu 10

Manual IP phone set-up..... 9

master.bin 49

Message flow
 auto-provisioning via activation code.... 29
 auto-provisioning via MAC address 29

MySQL
 in ap script 45

P

Perl script 41

Phone
 variant ID 25, 29

PHP
 install..... 38

Profile..... 3

Provider, role in provisioning process..... 4

Provisioning
 MAC-based..... 23

Provisioner, role in provisioning process..... 4

Provisioning
 get URL..... 26, 30

Provisioning data..... 8

Provisioning files, XML 33

Provisioning methods..... 6

Provisioning server 5

 customised..... 5

Provisioning, definition..... 3

R

Redirection
 web user interface 10

Redirection data
 deregistering via web user interface 11

Redirection data record

 registering via web user interface 10

Redirection server 5

Redirecton service..... 6

Registering
 via web user interface..... 10

Roles, in provisioning process..... 4

S

Schema file for XML template 33

Secret key 48

Security
 using activation code method..... 31

Security aspects 31

Server
 provisioning..... 5
 redirection..... 5
 update..... 5

SIP check-sync mechanism..... 32

SIP multicast mechanism..... 6, 22

T

Template..... 33

Template, auto-provisioning 40

U

Unicode..... 47

Update configuration 32

Update server..... 5

URI format string 26, 30

User account
 for provisioner web UI 10

UTF-8..... 47

V

VERSION parameter, in XML file 37

W

Web user interface..... 10
 for redirection data..... 10

Wiki 33

X

XML file 33
 uploading directly..... 12

 uploading vie web UI 12

XML FileUpload..... 12

XML schema file for template 40

XML template..... 33

XML-provisioning..... 3

 plain 3

 with binary 3

XML-RPC 13

XML-RPC command

 autoprov.checkDevice 19

 autoprov.checkDeviceList..... 20

 autoprov.deregisterDevice..... 16

 autoprov.deregisterDeviceList..... 21

 autoprov.listDevices..... 18

 autoprov.listDevices, all..... 17

 autoprov.registerDevice 15

XML-RPC commands 15

XSD schema file..... 33

Issued by

Gigaset Communications GmbH
Frankenstraße 2a, D-46395 Bocholt

© Gigaset Communications GmbH 2012

All rights reserved. Subject to availability.
Rights of modification reserved.

www.gigaset.com

A31008-M2212-R910-2-7643