Kerio Operator Provisioning Reference Guide

for Kerio Operator 2.6.0

Variables

This section presents and overview of all the ”input” variables that come preset when your scripts are run. They are shown together as many variables are available in several different contexts under the same name. A general rule of thumb is that a given variable is available everywhere that it makes sense. E.g.

- Variables describing global Operator configuration (`$TIMEZONE_OFFSET`, `$VOICEMAIL_EXT`) are available everywhere.
- Variables bound to a phone (`$LINES`, `$ADMIN_PASSWORD`, `setPhoneVar()` variables) are available only where an association to a specific phone is known, i.e.:
  - In providers after `associatePhone()`
  - In resync scripts from the start
- Request-related variables (`$PATH`, `$PROTO`, ...) are available only in providers, from the start.

In the following table, the *Avail.* column contains letters indicating the contexts in which a given variable is available:

P Provider.

A Provider after `associatePhone()`.

R Resync script.

T `phone_types` variable expansion.

F Firmware script.

Entries are sorted in a very rough order of importance.

Variable	Avail.	Description
(module persistent globals)	PR	Any variables set with `setModuleVar()`. See its description under *Functions.*
Request Variables		
`$PATH`	P	The filename requested by the phone, including full path. Normalized: starts with a /, double slashes removed, `/./` and `/../` components resolved and removed.
`$PROTO`	P	The protocol using which the request was received. One of the strings `'tftp'`, `'http'` and `'ftp'`.
`$OPERATOR_IP`	P	IP address of the Operator machine. In case of multiple network interfaces, the IP on which the request was receieved (i.e., the one that the phone ”can see”). This is the ”right” address to be set as SIP proxy, NTP server, etc. in the phone configuration).
`$PHONE_IP`	P	The IP adress from which the provisioning request came, i.e., of the phone. Not available in resync scripts, use `$LAST_IP` there. Neither is probably of much use, offered merely for completeness.

... continued on next page

Variable	Avail.	Description
(markers)	P	Any variables set with `setMarker()`. See the *Two-stage configuration; markers* section in the tutorial.
`$HTTP_*`	P	HTTP headers, available under the same names as in CGI, i.e. converted to uppercase, with dashes replaced by underlines and prefixed by `'HTTP_'`. E.g. `$HTTP_USER_AGENT`, `$HTTP_ACCEPT`, etc.
`$OPT_*`	P	TFTP Options (RFC 2347) E.g. `$OPT_blksize`, etc.
Global Operator Configuration		
`$TIMEZONE`	PR	Current Operator timezone as a POSIX/tzdata timezone name. E.g. `Europe/Prague` or `America/New_York`. See Wikipedia for a complete list. In most cases (unless your phone already understands these strings) it's easier to use `$TIMEZONE_OFFSET`.
`$TIMEZONE_OFFSET`	PR	Current Operator UTC offset in seconds. If DST is currently in effect, this value *already includes* the DST offset. This is usually the easiest way to configure phone's timezones – most Kerio's built-in provisioning modules use it.
`$TIMEZONE_OFFSET_BASE`	PR	Same as above, only doesn't include any DST offset.
`$VOICEMAIL_EXT`	PR	The voicemail access extension (usually 50). Can be used to make the "envelope" button present on many phones dial this number for faster voicemail access.
`$DISPLAY_LOGO`	PR	Boolean indicating whether phones should display configured logos. See the chapter on logos in the tutorial.
`$AUTODIAL_TIMEOUT`	PR	Auto dial timeout in seconds, sometimes also known as inter-digit timer. Defines time before a number is automatically dialed.
`$HOSTNAME`	PR	Operator hostname
Module Variables		

`$PHONE_TYPES`	PRF	Array of all the phone types defined in `phone_types`. Keys are type identifiers (first column in `phone_types`), values are arrays containing keys corresponding to variables in the *Phone type variables* section below. Usually used only to test phone type existence (`isset($PHONE_TYPES[$sometype])`), see *Efficiently handling larger number of phone types* in the tutorial. Example:
		
		array(
		  'spa504g' => array(
		    'IDENT' => 'spa504g',
		    'MAX_LINES' => 4,
		    'SECURE_SUPPORT' => false,
		    // ...
		  ),
		  // ...
		)

... continued on next page

Variable	Avail.	Description
`$FIRMWARE_TYPES`	PRF	Array of all the firmware slots defined in `firmware types`. Keys are the slot identifiers (first column), values are associative arrays with the following keys: `'FW_IDENT'` The slot identifier, once again. `'FW_CONTENT_ID'` A unique hexadecimal ID regenerated each time a new file is uploaded to the slot (even if it's identical to the previous one). This is useful in generating firmware filenames for phones which download new firmware only when the filename changes. See *Firmware and Logos -> Arbitrary Filenames* in the tutorial. `FW_UPLOADED` True if a file is uploaded in the slot, false otherwise. `FW_ORIGNAME` The original name (without path) of the file uploaded in the slot. The same as the `$FW_ORIGNAME` variable available in the firmware preprocessing script. `FW_SIZE` The disk space occupied by the preprocessed firmware, in bytes. Displayed in Administration, probably useless in scripts. `FW_SCRIPT` The name of the script used to preprocess files for this slot, without path or extension, i.e., the same as what is passed for `SCRIPT=` in `firmware_types` *Plus any extra NAME=VALUE assignments from "firmware_types".*
`$MODULE_ID`	PRF	ID under which the module is internally stored. Useful for constructing module-private filenames for auxiliary provisioning files (firmwares, logos, additional configuration files, etc.) for which a name can be chosen arbitrarily. See *Arbitrary Filenames* under *Firmwares and Logos* in the tutorial.
`$MODULE_UUID`	PRF	The UUID set in the `info` file.
`$MODULE_VERSION`	PRF	Module version set in the `info` file.
`$DIALPLAN_(name)`	PRF	The dialplan string generated by the `dialplans/(name).php` script. See *Dialplans* in the tutorial.
Phone Type Variables		
`$IDENT`	ART	The phone type identifier (first column in `phone_types`).
`$MAX_LINES`	ART	The phone type's maximum number of lines as set in `phone_types`.
(phone type options)	ART	Any `name=value` options set in `phone_types`. E.g. `$RESYNC`, `$REVIEW`, ... Usually uppercase. See the *Phone Type Options* section for a list.
(custom phone type variables)	ART	Any `name=value` definitions from `phone_types` that have no special meaning for Operator are available as-is. Customarily named lowercase. See *Efficiently Handling Larger Number of Phone Types* in the tutorial.
Phone Configuration		
`$MAC`	AR	The phone's MAC address in normalized format (lowercase, without colons).

... continued on next page

Variable	Avail.	Description
`$LINES`	AR	An array of phone's assigned lines. Each item is an array with the following keys: **`TELNUM`** The extension number (e.g. `10`) **`SIP_USERNAME`** The SIP username. May differ from `TELNUM` when using multiple registration. E.g. `10`, `10p1` or `10home`. Alphanumeric, lowercase. **`SIP_PASSWORD`** The SIP password. Usually generated. **`SECURE`** Whether secure SIP and RTP is enabled for this line. Only allowed if the phone type indicates `SECURE_SUPPORT=1` in its options. **`USER_FULLNAME`** Full name of the Operator user to whom this extension is assigned. **`USERNAME, USER_EMAIL`** The username and e-mail of this user. Probably uninteresting.
`$ADMIN_PASSWORD`	AR	The password that should be used to protect the phone's web administration. If master password is enabled, this is the master phone password, otherwise the password set in provisioned phone properties (generated by default).
`$PHONE_LABEL`	AR	The label for the phone as a whole as opposed to its individual lines. Also known as *station name*. Should be displayed on the phone's screen if possible.
`$PHONE_DESCRIPTION`	AR	Admin's description of the phone. For internal use, should NOT be displayed anywhere. Available only for completeness, probably useless.
`$DIALPLAN`	AR	The dialplan for this phone, based on `$DIALPLAN_TYPE` set in `phone_types`. E.g. when `$DIALPLAN_TYPE == 'spa'`, this variable has identical content to `DIALPLAN_spa`, i.e., whatever string was generated by the `dialplans/spa.php` script. See *Dialplans* in the tutorial.
Last Read Phone Configuration (useful mainly in resync)		
`$LAST_IP`	AR	Phone's last know IP address. Our "best shot" when trying to resync the phone.
`$LAST_CONFIG`	AR	The values of variables from the **Phone configuration** section at the last time that the phone has read the configuration. These should be the configuration that the phone "has seen" and currently uses. See *Implementing custom resync mechanisms* in the tutorial for usage example. Contains e.g. `$LAST_CONFIG['LINES']`, `$LAST_CONFIG['ADMIN_PASSWORD']`, etc.
Firmware Script Variables		
`$FW_IDENT`	F	Identifier (first column from `firmware_types`) of the slot the file is uploaded to. Useful when one script handles several similar slots.
`$FW_ORIGNAME`	F	The original name (without path) of the file uploaded by the user. E.g. `'spa942-6-1-5a.bin'`.
`$FW_INFILE`	F	The physical path of a temporary file to which the uploaded file was saved. This is the file that you can read, copy, convert, unzip or otherwise. Do not inspect its filename, it will be some random gibberish. Use `$FW_ORIGNAME` for that purpose.
`$FW_OUTDIR`	F	The physical path of a freshly-created directory to which the firmware script should put the preprocessed output. The directory will be preserved as-is (until the firmware is delete or replaced) and made available to the provider script, which can access its files using `getFirmwareFile()`.

... continued on next page

Variable	Avail.	Description
`$FW_VERIFY`	F	If `1`, the script should perform usual verification of the uploaded file. If false, it should fail only if it's physically impossible to preprocess the uploaded file (e.g. trying to unzip a file that's not a zip archive).

Tabular Files

On several occasions the provisioning modules need to describe lists of things (currently phone types and firmware slots). For this purpose, so-called *tabular files* are used. Their syntax is described throughout the tutorial, the advanced parts especially in *Efficiently Handling Larger Number of Phone Types*. Everything described there (variable expansions, `@set`, etc.) applies all the tabular files.

In the table below, options that are written in a fixed column without specifying a name have the column number in the *Pos.* column. Those written as `NAME=value` have the name in the *Name* column. Positional options can also be written as named (e.g. `IDENT=spa504g`) but this is discouraged.

`phone_types` options

Pos.	Name	Default	Description
1	(`IDENT`)	*(none)*	A module-unique identifier for the phone type. Preferrably lowercase, may contain underscores (e.g. `spa504g`).
2	(`MAX_LINES`)	*(none)*	Maximum number of lines that can be assigned to a phone of this type.
3	(`DESC`)	*(none)*	A human-readable name for the phone type, including manufacturer. E.g. `Cisco SPA504G`. Show on the *Provisioned Phones* screen in Administration.
	`RESYNC`	*(none)*	The method used to resync the phone. Either a built-in one (named uppercase, see *Built-In Resync Methods*) or the name of a resync script (customarily lowercase), which should then be located in `"resync/$RESYNC.php"` inside your module.
	(resync options)	*(none)*	Resync methods may take additional options. For `SIPNOTIFY`, they are named `SIPNOTIFY_something`, see *Butilt-In Resync Methods* .
	`REVIEW`	`''`	A whitespace-separated list of configuration files that belong to phones of this type. Use variable expansion to put things like MAC address (`$MAC`) or phone type (`$IDENT`) in the filenames. Refer to the *The Download Configuration feature* and *Variable expansion in* `phone_types` sections in the tutorial for details.
	`SECURE_SUPPORT`	0	Set to 1 when the phone supports Secure SIP and SRTP. If you indicate security support, your module MUST honor the SECURE flag for phone lines and MUST ensure secure connection is actually used (the users will count on it).
	`FW`	*(empty)*	A comma-separated list of identifiers of firmware slots associated with this phone. This serves two purposes: • It is used by `phoneFirmwares()` to return only slots relevant to the current phone's type. • When a file is uploaded to a slot, all phones whose types list the slot in their `FW` will be marked as requiring a resync.

... continued on next page

Pos.	Name	Default	Description
	(custom options)	*(none)*	You can add arbitrary custom options to `phone_types` (customarily named lowercase to avoid clashes with future Operator options). Any `name=value` pair you include in `phone_types` will be available as a variable in the A, R, T contexts as explained in Variables. See the *Efficiently handling larger number of phone types* for usage of this feature.

`firmware_types` options

Pos.	Name	Default	Description
1	(`FW_IDENT`)	*(none)*	A module-unique identifier for this slot.
2	(`DESC`)	*(none)*	A humand-readable name for the slot, shown in the *Firmwares and Logos* dialog in Administration.
3	(`ROLE`)	*(none)*	The role of the slot. Can be one of `firmware`, `logo` or `other`. It is used to display the slot on the correct tab of the firmwares dialog, it can be used as a filter in `phoneFirmwares()` and `moduleFirmwares()` but otherwise has no special significance. You could use a `logo` slot for firmwares if you really wanted to.
	`SCRIPT`	`$FW_IDENT`	The name of the script used to preprocess files uploaded to this slot, without path or extensions. Operators tries to run the files `firmware/$SCRIPT.sh` and `firmware/$SCRIPT.php` from the module directory, in this order.
	`LONGDESC`	*(empty)*	A long textual description (several lines) of what the user is supposed to upload to the slot. For firmware it's good to directly include download links. Any URL is automatically converted into a clickable hyperlink. The description is displayed in the upload dialog (i.e., when one clicks on *Edit* on the slot).
	(custom options)	*(none)*	You can add any arbitrary `name=value` options (preferrably named lowercase to avoid clashes with future Operator names). They will be in the preprocessing script (as global variables) and in the `$FIRMWARE_TYPES` entry for this slot, accessible from the provider.

Built-in Resync Methods

SIPNOTIFY

Functions

Functions marked `*` are "use only when you know what you are doing" functions.

General-Use Functions

`transliterate($str)` [**AR**] Replace accented characters from `$str` with their ASCII equivalents. The exact transliteration rules are set in *Operator Administration > Advanced Options > Transliteration*.

`normalizeMac($mac)` [**ART**] Normalize a MAC address: remove punctuation and make lowercase.

`getStaticFile($name)` Returns a full path to the installed copy of the `misc/$name` file from your module.

`setModuleVar($name, $val)` **[PR]** Store a persistent global variable. After calling `setModuleVar('somevar', 5)`, variable `$somevar` will be available preset in all contexts to the value of 5, similarly to how e.g. `$TIMEZONE` is available. Assigning to `$somevar` DOES NOT change the persistently stored value. That can be only done by `setModuleVar()`. There is no way to "unset" a module variable; you can set it to null, though. This allows modules to maintain state. However, we haven't found a use for it ourselves yet. Each module has its own namespace so there is no risk for naming conflicts.

`setPhoneVar($name, $val)` **[AR]** The same, only the variable is bound to a phone, each phone has its own namespace.

Provider Functions

Output Functions

A provider script can end two ways: successfully and unsuccessfully.

A successful exit means that the module generates a configuration file and sends it to the phone. It can write its output to STDOUT as usual (with `echo` and friends), however, using one of the provided output functions, escpecially `sendTemplate()`, is **strongly recommended** as it allows end users to override the templates with custom configuration modifications.

An unsuccessful exit consists of returning an error status (NotFound, PassNext) to the provisioning engine by calling one of the corresponding functions below. A module that exists unsucessfully MUST NOT write anything to STDOUT (i.e., with `echo`).

If your script reaches its end or raises an exception, `passNext()` is called implicitly. In case your module has already written anything to STDOUT, this partial output will get mixed with output of the following modules. Another reason why manual output is discouraged.

Unsuccessful Exit Functions Each of these functions returns the corresponding exit status to the provisioning engine and **terminates the script**.

`passNext()` Call when you don't know this file or are not interested in it. It passes the request on to the remaining modules in the list to try to handle it.

`* sendNotFound()` Return a "file not found" error to the phone immediately, without trying other modules. Use this for files that *belong* to your phone but you don't want them to exist, e.g. files to store user configuration (ringer volume and suchlike) that would overwrite whatever was set using the phone menu if they existed. To put it simply: do not use this function unless you know what you are doing.

Successful Exit Functions Each of the following functions sends some output and/or returns some status and **terminates the script**.

`sendTemplate($name)` Render a template `"templates/$name.tpl"` and send the result to the phone. Equivalent to `renderTemplate($name); sendOk();`. Using `sendTemplate()` is preferred.

`sendStatic($name)` Sends the file `"misc/$name"` from your module directory to the phone as-is. Mostly useful for binary and other special files. For static configuration files use `sendTemplate()` so that they can be overriden by the user!

`sendFile($path)` Sends a file given absolute path in the Operator file system. Only ever use this with paths returned by other functions (e.g. `getFirmwareFile()`). Never constructs any physical file paths yourself as Operator's directory layout may change in any way between versions!

`* sendOk()` Return success without outputting anything. Use when output was previously sent manually. May be used in rare cases without any preceding output to force sending an empty file. However, if it is possible to put something in the empty file, it's recommended to create an (empty) template for it so that the user can override the template with some non-empty content.

Output-Only Functions These functions send output but do not terminate the script. Manual output is error-prone. Use only when you have a good reason to!

`* renderTemplate($name)` Send the output of a template without exiting. Can be used to mix templates with other kinds of output, although we can't think of any use case for that. Included for completeness.

Other Functions

`associatePhone($mac_address, $phone_type, $update_type = true)` Inform Operator that the currently generated configuration file belongs to a phone with MAC address `$mac_address` and phone type `$phone_type` (must be a valid identifier from your module's `phone_types`), create a record for this phone in *Provisioned Phones* if one doesn't exist (and generate a line if enabled), fetch its configuration (lines, etc.) and put it in the respective global variables (see the *Phone configuration* section under Variables above). Set update_type to false to use phone_type from the database. If the phone doesn't exist, phone_type variable will still be used to create the phone.

Firmware-Related Functions

`haveFirmware($slot)` Return true if a file is upload to firmware slot `$slot`, false otherwise.

`getFirmwareFile($slot, $file)` Return the physical path where the file created by the firmware preprocessing script as `$FW_OUTDIR/$file` is stored. You can use usual PHP functions (`fopen`, `fread`, ...) to work with this file, but usually you will just pass the path to `sendFile()`. See the tutorial for an example.

If there is nothing uploaded in the slot, `false` is returned. If there is something uploaded but `$file` doesn't exist in the preprocessed directory, `null` is returned.

`moduleFirmwares($role = '', $uploadedOnly = false, $filters = array())` Return a list of module's firmware slots matching criteria given by the arguments:

- `$role`: Return only slots with a given role (`firmware`, `logo` or `other`). By default slots of all roles are returned.
- `$uploadedOnly`: If true, return only slots that have a file uploaded. By default all slots are returned.
- `$filters`: An associative array of additional filters on custom slot properties (as specified in `firmware_types`).

  For example, if your `firmware_types` looks like:

  ```
  fw1         ``Firmware 1''   firmware  subtype=firmware
  fw2         ``Firmware 2''   firmware  subtype=firmware
  bootloader1 ``Bootloader 1'' firmware  subtype=bootloader
  ```

  then `moduleFirmwares('firmware', false, array('subtype' => 'firmware'))` would return:

  ```
  array(`fw1', `fw2')
  ```

The return value is an array of slot identifiers (strings). Only slots matching *all* the specified criteria are included.

`phoneFirmwares($role = '', $uploadedOnly = false, $filters = array())` Same as `moduleFirmwares()` but adds one more implicit criterion: only slots associated with the current phone's type (as specified by `FW=` in `phone_types`) are returned. It must be called after `associatePhone()` so that it knows the phone type.

Resync Script Functions

`urlopenPhone($scheme, $pathAndQuery, $opts = array())` Open the URL `"$scheme://$LAST_IP/$pathAndQuery"` (where `$LAST_IP` is the last known IP of the phone) and return a file handle for reading the contents.

`$opts` may contain the following options:

Name	Default	Description
`user`	*(none)*	Username for HTTP authentication.
`password`	*(none)*	Password for HTTP authentication.
`auth`	`'auto'`	HTTP authentication type: `'basic'`, `'digest'` or `'auto'`. It's recommended to set this to the right type instead if `auto`. In case of basic auth, it will save you one extra request (that would end with 401). In case of digest auth, it will increase security.
`body`	*(none)*	The request body. When present, a POST request is sent instead of the default GET.
`referer`	*(none)*	Set the `Referer` header.
`verifySSlCert`	`true`	Set to `false` to disable SSL certificate verification. Almost always needed when using HTTPS. The default is `true` only for "formal correctness", otherwise useless.

Uses CURL as backend. Cookies are automagically preserved between requests.

`connectPhone($proto, $port)` Return a socket connected to the phone using protocol `$proto` (`'tcp'` or `'udp'`) on port `$port`.

Equivalent to:

```
$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP); // for tcp
$sock = socket_create(AF_INET, SOCK_DGRAM,  SOL_UDP); // for udp
socket_connect($sock, $LAST_IP, $port);
```

You shouldn't use `socket_create` or `socket_connect` in your scripts directly. Always use `connectPhone()`.

`getRegCallId($lineNo)` Get the `Call-Id` of the registration call of phone's `$lineNo`-th line (0-based, usually 0). Some phones (e.g. some archaic Cisco models) may require this value in resync requests.

`sendSipNotify($opts)` Send a SIP NOTIFY request to the phone. This does exactly the same as the `SIPNOTIFY` built-in resync method. It may be used when you need to compute the SIP NOTIFY options in some more complex way than `RESYNC=SIPNOTIFY` allows. It may also be used to combine resync methods: e.g. the Kerio's built-in SPA module does resync using *both* HTTP (`urlopenPhone`) and SIP NOTIFY (`sendSipNotify`) as each of these methods has its own unreliabilities.

The individual options availables:

Name	Default	Description
`auth`	`'none'`	Whether and how to authenticate the request. Allowed values: `'none'`, `'auto'`, `'basic'` and `'digest'`. Setting a specific type is preferred to `'auto'` for the same reasons as outlined in `urlopenPhone()`.
`username`	`$LAST_CONFIG ['LINES'][0] ['SIP_USERNAME']`	The `To:` username used in the resync request.
`authUser`	(*same as* `username`)	The authentication username for the resync request.
`password`	`$LAST_CONFIG ['LINES'][0] ['SIP_PASSWORD']`	The password used to authenticate the resync request.
`event`	`'check-sync'`	The `Event:` header.
`arg`	*(none)*	Additional arguments for the event. They will be appended after the event name separated by a semicolon. E.g. when `arg` is `'reboot=true'`, the request will contain: `Event: check-sync;reboot=true`

... continued on next page

Name	Default	Description
addHeaders	array()	Additional headers as a list of array($name, $value) pairs.
body	''	The request body. Usually empty.
port	5060	Port to send the packets to.
callId	*(random)*	The Call-Id for the request. Some phones may require setting it to getRegCallId(0). Most don't. Don't touch unless you need to.
cseq	1	You probably shouldn't change this either.

The resulting packet will look like this:

```
NOTIFY sip:{$username}@{$LAST_IP} SIP/2.0``
Via:            SIP/2.0/UDP {$OPERATOR_IP}:{/*source port*/};branch=1
From:           <sip:admin@{$OPERATOR_IP}>
To:             <sip:{$username}@{$LAST_IP}>
Event:          {$event}{$arg ? ''; $arg`` : ''``}
Call-ID:        {$callId}
CSeq:           {$cseq} NOTIFY
Contact:        <sip:admin@{$OPERATOR_IP}>
[Authorization: ...]
Content-Length: {strlen($body)}

{$body}
```

Dialplan Script Functions

dialPlanSimplifyOnedigit($rawData) See *Diaplans* in the tutorial.

Changelog

2.3.1

- $FIRMWARE_TYPES variable.
- Documented the firmware API.

2.2.2

- Changed the dialplan API to use dialplans/*.php scripts and the $DIALPLAN_TYPE phone type variable instead of dialplan-generating functions and generateDialplan() in provider. generateDialplan() (and related functionality) becomes deprecated and will be removed in a future version. (This was necessary in order to be able to trigger "some phones don't have an up-to-date configuration" notifications when dialplan changes.)

2.2.0 beta 3

- Cookie support in HTTP resync.
- Basic support for firmwares and logos (firmware_types, getFirmwareFile(), haveFirmware(), phoneFirmwares(), moduleFirmwares(), preprocessing script API, etc.). Has several bugs with reagard to custom provisioning modules (e.g. firmware scripts must have Unix line ending instead of being autoconverted like all the other files). We recommend using 2.3.1 or above.