Zum Anfang Alles laden

RHASSPY

This module receives, processes and executes voice commands coming from Rhasspy voice assistent.

General Remarks:

- For dialogues, RHASSPY relies on the mechanisms as described in Rhasspy Dialogue Manager documentation.
  So don't expect these parts to work if you use other options than Rhasspy's own dialogue management.
- You may need or want some additional materials to get the best out of RHASSPY in FHEM. So have a look at the additional files and examples provided in svn contrib.
  See especially attributes languageFile and rhasspyIntents for further reference.

Define

```
define <name> RHASSPY <baseUrl> <devspec> <defaultRoom> <language>
<fhemId> <prefix> <useGenericAttrs> <handleHotword> <Babble> <encoding>
```

All parameters in define are optional, most will not be needed (!), but keep in mind: changing them later might lead to confusing results for some of them! Especially when starting with RHASSPY, do not set any other than the first three (or four if your language is neither english nor german) of these at all!

Remark:
RHASSPY uses **parseParams** at quite a lot places, not only in define, but also to parse attribute values.
So all parameters in define should be provided in the *key=value* form. In other places you may have to start e.g. a single line in an attribute with `option:key="value xy shall be z"` or `identifier:yourCode={fhem("set device off")} anotherOption=blabla` form.

Parameters:

- **baseUrl**: http-address of the Rhasspy service web-interface. Optional, but needed as soon as default (`baseUrl=http://127.0.0.1:12101`) is not appropriate.
  Make sure, this is set to correct values (ip and port) if Rhasspy is not running on the same machine or not uses default port!
- **devspec**: All the devices you want to control by Rhasspy **must meet devspec**. If *genericDeviceType* support is enabled, it defaults to `genericDeviceType=.+`, otherwise the former default `devspec=room=Rhasspy` will be used. See as a reference, how to e.g. use a comma-separated list of devices or combinations like `devspec=room=livingroom,room=bathroom,bedroomlamp`.
- **defaultRoom**: Default room name. Used to speak commands without a room name (e.g. "turn lights on" to turn on the lights in the "default room"). Optional, but also recommended. Default is `defaultRoom=default`.
- **language**: Makes part of the topic tree, RHASSPY is listening to. Should (but needs not to) point to the language voice commands shall be spoken with. Default is derived from global, which defaults to `language=en`. Preferably language should be set appropriate in global, if possible.
- **fhemId**: May be used to distinguishe between different instances of RHASSPY on the

Zum Anfang Alles laden

versions - be a criteria to distinguish between different users (e.g. to only allow a subset of commands and/or rooms to be addressed). Not recommended to be set if just one RHASSPY device is defined.
- **prefix**: May be used to distinguishe between different instances of RHASSPY on the FHEM-internal side.
  Might be usefull, if you have several instances of RHASSPY in one FHEM running and want e.g. to use different identifier for groups and rooms (e.g. a different language). Not recommended to be set if just one RHASSPY device is defined.
- **useGenericAttrs**: Formerly, RHASSPY only used it's own attributes (see list below) to identifiy options for the subordinated devices you want to control. Today, it is capable to deal with a couple of commonly used `genericDeviceType` (*switch*, *light*, *thermostat*, *thermometer*, *blind*, *media*, *scene* and *info*), so it will add `genericDeviceType` to the global attribute list and activate RHASSPY's feature to estimate appropriate settings - similar to rhasspyMapping. `useGenericAttrs=0` will deactivate this. (do not set this unless you know what you are doing!). Notes:
  - As some devices may not directly provide all their setter infos at startup time, RHASSPY will do a second automatic devicemap update 2 minutes after each FHEM start. In the meantime not all commands may work.
  - `homebridgeMapping` atm. is not used as source for appropriate mappings in RHASSPY.
- **handleHotword**: Trigger Reading *hotword* in case of a hotword is detected. See attribute rhasspyHotwords for further reference.
- **Babble**: **experimental!** Points to a Babble device. Atm. only used in case if text input from an AMADCommBridge is processed, see rhasspySpeechDialog for details.
- **encoding**: **most likely deprecated!** May be helpfull in case you experience problems in conversion between RHASSPY (module) and Rhasspy (service). Example: `encoding=cp-1252`. Do not set this unless you experience encoding problems!
- **sessionTimeout** **experimental!** timout limit in seconds. By default, RHASSPY will close a sessions immediately once a command could be executed. Setting a timeout will keep session open until timeout expires. NOTE: Setting this key may result in confusing behaviour. Atm not recommended for regular useage, **testing only!** May require some non-default settings on the Rhasspy side to prevent endless self triggering.
- **autoTraining**: **experimental!** deactivated by setting the timeout (in seconds) to "0", default is "60". If not set to "0", RHASSPY will try to catch all actions wrt. to changes in attributes that may contain any content relevant for Rhasspy's training. In case if, training will be initiated after timeout hast passed since last action; see also update devicemap command.

RHASSPY needs a MQTT2_CLIENT device connected to the same MQTT-Server as the voice assistant (Rhasspy) service.

Examples for defining an MQTT2_CLIENT device and the Rhasspy device in FHEM:

- **Minimalistic version** - Rhasspy running on the same machine using it's internal MQTT server, MQTT2_CLIENT is only used by RHASSPY, language setting from *global* is used:

```
defmod rhasspyMQTT2 MQTT2_CLIENT localhost:12183
attr rhasspyMQTT2 clientOrder RHASSPY
attr rhasspyMQTT2 subscriptions setByTheProgram

define Rhasspy RHASSPY defaultRoom=Livingroom
```

- **Extended version** - Rhasspy running on remote machine using an external MQTT server on a third machine with non-default port, MQTT2_CLIENT is also used by

Zum Anfang Alles laden

```
attr rhasspyMQTT2 clientOrder RHASSPY MQTT_GENERIC_BRIDGE
MQTT2_DEVICE
attr rhasspyMQTT2 subscriptions hermes/intent/+
hermes/dialogueManager/sessionStarted hermes/dialogueManager
/sessionEnded hermes/nlu/intentNotRecognized hermes/hotword
/+/detected <additional subscriptions for other MQTT-Modules>

define Rhasspy RHASSPY baseUrl=http://192.168.1.210:12101
defaultRoom="Büro Lisa" language=de
devspec=genericDeviceType=.+,device_a1,device_xy handleHotword=1
```

Additionals remarks on MQTT2-IOs:

Using a separate MQTT server (and not the internal MQTT2_SERVER) is highly recommended, as the Rhasspy scripts also use the MQTT protocol for internal (sound!) data transfers. Best way is to either use MQTT2_CLIENT (see above) or bridge only the relevant topics from mosquitto to MQTT2_SERVER (see e.g. http://www.steves-internet-guide.com/mosquitto-bridge-configuration for the principles). When using MQTT2_CLIENT, it's necessary to set `clientOrder` to include RHASSPY (as most likely it's the only module listening to the CLIENT it could be just set to `attr <m2client> clientOrder RHASSPY`)

Furthermore, you are highly encouraged to restrict subscriptions only to the relevant topics:

```
attr <m2client> subscriptions setByTheProgram
```

In case you are using the MQTT server also for other purposes than Rhasspy, you have to set `subscriptions` manually to at least include the following topics additionally to the other subscriptions desired for other purposes.

```
hermes/intent/+
hermes/dialogueManager/sessionStarted
hermes/dialogueManager/sessionEnded
hermes/nlu/intentNotRecognized
hermes/hotword/+/detected
```

Important: After defining the RHASSPY module, you are supposed to manually set the attribute *IODev* to force a non-dynamic IO assignement. Use e.g. `attr <deviceName> IODev <m2client>`.

Note: RHASSPY consolidates a lot of data from different sources. The **final data structure RHASSPY uses at runtime** will be shown by the list command. It's highly recommended to have a close look at this data structure, especially when starting with RHASSPY or in case something doesn't work as expected!
After changing something relevant within FHEM for either the data structure in

- **RHASSPY** (this form is used when reffering to module or the FHEM device) or for
- **Rhasspy** (this form is used when reffering to the remote service),

you have to make sure these changes are also updated in RHASSPYs internal data structure and (often, but not always) to Rhasspy. See the different versions provided by the update command.

Zum Anfang Alles laden

- **update**

  Various options to update settings and data structures used by RHASSPY and/or Rhasspy.
  Choose between one of the following:

  - **devicemap**
    When having finished the configuration work to RHASSPY and the subordinated devices, issuing a devicemap-update is required. You may do that manually in case you have deactivated the "autoTraining" feature or do not want to wait untill timeout is reached. Issueing that command will get the RHASSPY data structure updated, inform Rhasspy on changes that may have occured (update slots) and initiate a training on updated slot values etc., see remarks on data structure above.
  - **devicemap_only**
    This may be helpfull to make an intermediate check, whether attribute changes have found their way to the data structure. This will neither update slots nor (immediately) initiate any training towards Rhasspy.
  - **slots**
    This may be helpfull after checks on the FHEM side to immediately send all data to Rhasspy and initiate training.
  - **slots_no_training**
    This may be helpfull to make checks, whether all data is sent to Rhasspy. This will not initiate any training.
  - **language**
    Reinitialization of language file.
    Be sure to execute this command after changing something within in the language configuration file!
  - **intent_filter**
    Reset intent filter used by Rhasspy dialogue manager. See intentFilter in *rhasspyTweaks* attribute for details.
  - **all**
    Surprise: means language file and full update to RHASSPY and Rhasspy including training and intent filter.

  Example: `set <rhasspyDevice> update language`

- **play <siteId and path+filename>**

  Send WAV file to Rhasspy.
  siteId and *path and filename* are required!
  You may optionally add a number of repeats and a wait time in seconds between repeats.
  wait defaults to 15, if only *repeats* is given.

  Examples:
  ```
  set <rhasspyDevice> play siteId="default" path="/opt/fhem/test.wav"
  set <rhasspyDevice> play siteId="default" path="./test.wav"
  repeats=3 wait=20
  ```

- **speak <siteId and text>**

  Voice output over TTS.
  Both arguments (siteId and text) are required!

  Example:

Zum Anfang Alles laden

Send a text command to Rhasspy.

Example:
```
set <rhasspyDevice> textCommand turn the light on
```

- **fetchSiteIds**

  Send a request to Rhasspy to send all siteId's. This by default is done once, so in case you add more satellites to your system, this may help to get RHASSPY updated.

  Example:
  ```
  set <rhasspyDevice> fetchSiteIds
  ```

- **trainRhasspy**

  Sends a train-command to the HTTP-API of the Rhasspy master
  Might be removed in the future versions in favor of the update features

  Example:
  ```
  set <rhasspyDevice> trainRhasspy
  ```

- **volume <float value>**

  Sets volume of given siteId between 0 and 1 (float)
  Both arguments (siteId and volume) are required!

  Example:
  ```
  set <rhasspyDevice> siteId="default" volume="0.5"
  ```

- **customSlot <parameters>**

  Creates a new - or overwrites an existing slot - in Rhasspy
  Provide slotname, slotdata and (optional) info, if existing data shall be overwritten and training shall be initialized immediately afterwards.
  First two arguments are required, third and fourth are optional.
  overwrite defaults to *true*, setting any other value than *true* will keep existing Rhasspy slot data.

  Examples:
  ```
  set <rhasspyDevice> customSlot mySlot a,b,c overwrite training
  set <rhasspyDevice> customSlot slotname=mySlot slotdata=a,b,c
  overwrite=false
  ```

- **activateVoiceInput**

  Activate a satellite for voice input. *siteId*, *hotword* and *modelId* may be provided (either in order of appearance or as named arguments), otherwise some defaults will be used.

Get

- **export_mapping <devicename>**

  Exports a "classical" rhasspyMapping attribute value for the provided device. You may find

Zum Anfang Alles laden

- **test_file <path and filename>**

  Checks the provided text file. Content will be sent to Rhasspy NLU for recognition (line by line), result will be written to the file '<input without ending.txt>_result.txt'. ***stop*** as filename will stop test mode if sth. goes wrong. No commands will be executed towards FHEM devices while test mode is active.

  Note: To get test results, RHASSPY's siteId has to be configured for intent recognition in Rhasspy as well.

- **test_sentence <sentence to be analyzed>**

  Checks the provided sentence for recognition by Rhasspy NLU. No commands to be executed as well.

  Note: wrt. to RHASSPY's siteId for NLU see remark get test_file.

Attributes

Note: To get RHASSPY working properly, you have to configure attributes at RHASSPY itself and the subordinated devices as well.

RHASSPY itself supports the following attributes:

- **languageFile**

  Path to the language-config file. If this attribute isn't set, a default set of english responses is used for voice responses.
  The file itself must contain a JSON-encoded keyword-value structure (partly with sub-structures) following the given structure for the mentioned english defaults. As a reference, there's one in the additionals files available in german (note the comments there!), or just make a dump of the English structure with e.g. (replace RHASSPY by your device's name): `{toJSON($defs{RHASSPY}->{helper}{lng})}`, edit the result e.g. using https://jsoneditoronline.org and place this in your own languageFile version. There might be some variables to be used - these should also work in your sentences.
  languageFile also allows combining e.g. a default set of german sentences with some few own modifications by using "defaults" subtree for the defaults and "user" subtree for your modified versions. This feature might be helpful in case the base language structure has to be changed in the future.

  Example (placed in the same dir fhem.pl is located):

  ```
  attr <rhasspyDevice> languageFile ./rhasspy-de.cfg
  ```

- **response**

  Note: Using this attribute is no longer recommended, use options provided by the languageFile attribute instead.

  Optionally define alternative default answers. Available keywords are `DefaultError`, `NoActiveMediaDevice` and `DefaultConfirmation`.

  Example:

Zum Anfang Alles laden

- **rhasspyIntents**

  Defines custom intents. See Custom Intent erstellen.
  One intent per line.

  Example:

  ```
  attr <rhasspyDevice> rhasspyIntents
  SetCustomIntentsTest=SetCustomIntentsTest(siteId,Type)
  ```

  together with the following myUtils-Code should get a short impression of the possibilities:

  ```
  sub SetCustomIntentsTest {
  my $room = shift;
  my $type = shift;
  Log3('rhasspy',3 , "RHASSPY: Room $room, Type $type");
  return "RHASSPY: Room $room, Type $type";
  }
  ```

  The following arguments can be handed over:

  - NAME => name of the RHASSPY device addressed,
  - DATA => entire JSON-$data (as parsed internally), encoded in JSON
  - siteId, Device etc. => any element out of the JSON-$data.

  If a simple text is returned, this will be considered as response, if return value is not defined, the default response will be used.
  For more advanced use of this feature, you may return either a HASH or an ARRAY data structure. If ARRAY is returned:

  - First element of the array is interpreted as response and may be plain text (dialog will be ended) or HASH type to continue the session. The latter will keep the dialogue-session open to allow interactive data exchange with *Rhasspy*. An open dialogue will be closed after some time, (configurable) default is 20 seconds, you may alternatively hand over other numeric values as second element of the array.
  - Second element might either be a comma-separated list of devices that may have been modified (otherwise, these devices will not cast any events! See also the "d" parameter in *rhasspyShortcuts*), or (if first element is HASH type) a nummeric value as timeout.
  - If HASH type data (or $response in ARRAY) is returned to continue a session, make sure to hand over all relevant elements, including especially *intentFilter* if you want to restrict possible intents. It's recommended to always also activate *CancelAction* to allow user to actively exit the dialoge.

  See also additionals files for further examples on this.

- **rhasspyShortcuts**

  Define custom sentences without editing Rhasspys sentences.ini
  The shortcuts are uploaded to Rhasspy when using the updateSlots set-command.
  One shortcut per line, syntax is either a simple and an extended version.

  Examples:

Zum Anfang Alles laden

```
text='Thanks a lot, you are even more exciting!'"
i="mute off" p={fhem ("set $NAME mute off")} n=amplifier2 c="Please
confirm!"
i="i am hungry" f="set Stove on" d="Stove" c="would you like roast
pork"
```

Abbreviations explanation:

- **i** => intent
  Lines starting with "i:" will be interpreted as extended version, so if you want to use that syntax style, starting with "i:" is mandatory.
- **f** => FHEM command
  Syntax as usual in FHEMWEB command field.
- **p** => Perl command
  Syntax as usual in FHEMWEB command field, enclosed in {}; this has priority to "f=".
- **d** => device name(s, comma separated) that shall be handed over to fhem.pl as updated. Needed for triggering further actions and longpoll! Note: When calling Perl functions, the return value of the called function will be used if no explicit device is provided.
- **r** => Response to be send to the caller. If not set, the return value of the called function will be used.
  Response sentence will be parsed to do "set magic"-like replacements, so also a line like `i="what's the time for sunrise" r="at [Astro:SunRise] o'clock"` is valid.
  You may ask for confirmation as well using the following (optional) shorts:
  - **c** => either numeric or text. If numeric: Timeout to wait for automatic cancellation. If text: response to send to ask for confirmation.
  - **ct** => numeric value for timeout in seconds, default: 15.
    See *here* for more info about confirmations.

- **rhasspyTweaks**

  Place for additional settings to influence RHASSPY's global behavior on certain aspects.

  - **timerLimits**

    Used to determine when the timer should response with e.g. "set to 30 minutes" or with "set to 10:30"

    ```
    timerLimits=90,300,3000,2*HOURSECONDS,50
    ```

    Five values have to be set, corresponding with the limits to *timerSet* responses. so above example will lead to seconds response for less then 90 seconds, minute+seconds response for less than 300 seconds etc.. Last value is the limit in seconds, if timer is set in time of day format.

  - **timerSounds**

    Per default the timer responds with a voice command if it has elapsed. If you want to use a wav-file instead, you can set this here.

    ```
    timerSounds= default=./yourfile1.wav eggs=3:20:./yourfile2.wav
    potatoes=5:./yourfile3.wav
    ```

Zum Anfang Alles laden

default is optional. If set, this file will be used for all labeled timer without match to other keywords.
The two numbers are optional. The first one sets the number of repeats, the second is the waiting time between the repeats.
repeats defaults to 5, *wait* to 15
If only one number is set, this will be taken as *repeats*.

- **timeouts**

  Atm. keywords *confirm* and/or *default* can be used to change the corresponding defaults (15 seconds / 20 seconds) used for dialogue timeouts.

  Example:

  ```
  timeouts: confirm=25 default=30
  ```

- **confidenceMin**

  By default, RHASSPY will use a minimum *confidence* level of *0.66*, otherwise no command will be executed. You may change this globally (key: default) or more granular for each intent specified.
  Example:

  ```
  confidenceMin= default=0.6 SetMute=0.4 SetOnOffGroup=0.8
  SetOnOff=0.8
  ```

- **confirmIntents**

  This key may contain <*Intent*>=<*regex*> pairs beeing

  - *Intent* one of the intents supporting confirmation feature (all set type intents) and
  - *regex* containing a regular expression matching to either the group name (for group intents) or the device name(s) - using a full match lookup. If intent and regex match, a confirmation will be requested. Example:

    ```
    confirmIntents=SetOnOffGroup=light|blinds SetOnOff=blind.*
    ```

  To execute any action requiring confirmation, you have to send an *Mode:OK* value by the *ConfirmAction* intent. Any other *Mode* key sent to *ConfirmAction* intent will be interpretad as cancellation request. For cancellation, you may alternatively use the *CancelAction* intent. Example:

  ```
  [de.fhem:ConfirmAction]
  ( yes, please do it | go on | that's ok | yes, please
  ){Mode:OK}
  ( don't do it after all ){Mode}
  [de.fhem:CancelAction]
  ( let it be | oh no | cancel | cancellation ){Mode:Cancel}
  ```

- **confirmIntentResponses**

  By default, the answer/confirmation request will be some kind of echo to the originally spoken sentence ($rawInput as stated by *DefaultConfirmationRequestRawInput* key in *responses*). You may change this for each intent specified using $target, ($rawInput) and $Value als parameters.

Zum Anfang Alles laden

```
$target $Value" SetOnOff="confirm setting $target $Value"
```

$Value may be translated with defaults from a *words* key in languageFile, for more options on *$Value* and/or more specific settings in single devices see also *confirmValueMap* key in rhasspySpecials.

- **ignoreKeywords**

  You may have also some technically motivated settings in the attributes RHASSPY uses to generate slots, e.g. *MQTT, alexa, homebridge* or *googleassistant* in *room* attribute. The key-value pairs will sort the given *value* out while generating the content for the respective *slot* for *key* (atm. only *rooms* and *group* are supported). *value* will be treated as (case-insensitive) regex with need to exact match.
  Example:

  ```
  ignoreKeywords=room=MQTT|alexa|homebridge|googleassistant|logi
  cs-.*
  ```

- **gdt2groups**

  You may want to assign some default groupnames to all devices with the same genericDeviceType without repeating it in all single devices.
  Example:

  ```
  gdt2groups= blind=rollläden,rollladen thermostat=heizkörper
  light=lichter,leuchten
  ```

- **mappingOverwrite**

  If set, any value set in rhasspyMapping attribute will delete all content detected by automated mapping analysis (default: only overwrite keys set in devices rhasspyMapping attributes.

  Example:

  ```
  mappingOverwrite=1
  ```

- **extrarooms**

  You may want to add more rooms to what Rhasspy can recognize as room. Using this key, the comma-separated items will be sent as rooms for preparing the room and mainrooms slots.
  Example:

  ```
  extrarooms= barn,music collection,cooking recipies
  ```
  Note: Only do this in case you really know what you are doing! Additional rooms only may be usefull in case you have some external application knowing what to do with info assinged to these rooms!

- **updateSlots**

  Changes aspects on slot generation and updates.

  ```
  noEmptySlots=1
  ```

Zum Anfang Alles laden

to this type. If set to 1, no empty slots will be generated.

`overwrite_all=false`

By default, RHASSPY will overwrite all generated slots. Setting this to *false* will change this.

- **intentFilter**

  Atm. Rhasspy will activate all known intents at startup. As some of the intents used by FHEM are only needed in case some dialogue is open, it will deactivate these intents (atm: *ConfirmAction, CancelAction, ChoiceRoom* and *ChoiceDevice*(including the additional parts derived from language and fhemId))) at startup or when no active filtering is detected. You may disable additional intents by just adding their names in *intentFilter* line or using an explicit state assignment in the form *intentname=true* (Note: activating the 4 mentionned intents is not possible!). For details on how *configure* works see Rhasspy documentation.

- **rhasspyHotwords**

  Define custom reactions as soon as a specific hotword is detected (or with "global": a toggle command is detected). This does not require any specific configuration on any other FHEM device.
  One hotword per line, syntax is either a simple and an extended version. The "hotword" *global* will be treated specially and can be used to also execute custom commands when a *toggle* event is indicated.

  Examples:

```
bumblebee_linux = set amplifier2 mute on
porcupine_linux = livingroom="set amplifier mute on" default=
{Log3($DEVICE,3,"device $DEVICE - room $ROOM - value $VALUE")}
global = { rhasspyHotword($DEVICE,$VALUE,$DATA,$MODE) }
```

  First example will execute the command for all incoming messages for the respective hotword, second will decide based on the given *siteId* keyword; $DEVICE is evaluated to RHASSPY name, $ROOM to siteId and $VALUE to the hotword. Additionally, in "global key", $DATA will contain entire JSON-$data (as parsed internally, encoded in JSON) and $MODE will be one of *on*, *off* or *detected*
  . You may assign different commands to *on*, *off* and *detected*. *default* is optional. If set, this action will be executed for all *siteIds* without match to other keywords.
  Additionally, if either *rhasspyHotwords* is set or key *handleHotword* in DEF is activated, the reading *hotword* will be filled with *hotword* plus *siteId* to also allow arbitrary event handling.
  NOTE: As all hotword messages are sent to a common topic structure, you may need additional measures to distinguish between several *RHASSPY* instances, e.g. by restricting subscriptions and/or using different entries in this attribute.

- **rhasspyMsgDialog**

  If some key in this attribute are set, RHASSPY will react somehow like a msgDialog device. This needs some configuration in the central msgConfig device first, and additionally for each RHASSPY instance a siteId has to be added to the intent recognition service.

Zum Anfang Alles laden

(comma-separated device names). This ist the only **mandatory** key to be set.
- *open* A keyword or expression used to initiate a dialogue (will be converted to a regex compatible notation)
- *sessionTimeout* timout limit in seconds (**recommended**). All sessions will be closed automatically when timeout has passed. Timer will be reset with each incoming message .
- *close* keyword used to exit a dialogue (similar to open) before timeout has reached
- *hello* and *goodbye* are texts to be sent when opening or exiting a dialogue
- *msgCommand* the fhem-command to be used to send messages to the messenger service.
- *siteId* the siteId to be used by this RHASSPY instance to identify it as satellite in the Rhasspy ecosystem
- *querymark* Text pattern that shall be used to distinguish the queries done in intent MsgDialog from others (for the future: will be added to all requests towards Rhasspy intent recognition system automatically; not functional atm.)

Remarks on rhasspySpeechDialog and Babble:
Interaction with Babble and AMAD.*-Devices is not approved to be propperly working yet. Further tests may be needed and functionality may be subject to changes!

- **rhasspySpeechDialog experimental!**

  Optionally, you may want not to use the internal speach-to-text engine provided by Rhasspy (for one or several siteId's), but provide simple text to be forwarded to Rhasspy for intent recognition. Atm. only "AMAD" is supported for this feature. For generic "msg" (and text messenger) support see rhasspyMsgDialog
  Note: You will have to (de-) activate these parts of the Rhasspy ecosystem for the respective satellites manually!

  Keys that may be set in this attribute:
  - *allowed* A list of AMADDevice devices allowed to interact with RHASSPY (comma-separated device names). This ist the only **mandatory** key to be set.
  - *filterFromBabble* By default, all incoming messages from AMADDevice/AMADCommBridge will be forwarded to Rhasspy. For better interaction with Babble you may opt to ignore all messages not matching the *filterFromBabble* by their starting words (case-agnostic, will be converted to a regex compatible notation). You additionally have to set a *Babble* key in DEF pointing to the Babble device. All regular messages (start sequence not matching filter) then will be forwarded to Babble using `Babble_DoIt()` function.
  - *<allowed AMAD-device>* A list of key=value pairs to tweak default behaviours:
    - *wakeword* If set, a wakeword detected message for this wakeword will lead to an "activateVoiceInput" command towards this AMADDevice
    - *sessionTimeout* timeout (in seconds) used if a request (e.g. for confirmation) is open for this AMADDevice (if not set, global default value is used)
    - Remark: This may contain additional keys in the future, e.g., to restrict wakeword effect to a specific siteId.

  Example:

```
allowed=AMADDev_A
filterFromBabble=tell rhasspy
AMADDev_A=wakeword=alexa sessionTimeout=20
```

Zum Anfang Alles laden

modules like MQTT2_DEVICE, even if the topic matches to one of it's own subscriptions. By default, these messages will not be forwarded for better compability with autocreate feature on MQTT2_DEVICE. See also clientOrder attribute in MQTT2 IO-type commandrefs; setting this in one instance of RHASSPY might affect others, too.

For the subordinated devices, a list of the possible attributes is automatically extended by several further entries

The names of these attributes all start with the *prefix* previously defined in RHASSPY - except for genericDeviceType (gDT).
These attributes are used to configure the actual mapping to the intents and the content sent by Rhasspy.

Note: As the analyses of the gDT is intented to lead to fast configuration progress, it's highly recommended to use this as a starting point. All other RHASSPY-specific attributes will then be considered as a user command to **overwrite** the results provided by the automatics initiated by gDT usage.

By default, the following attribute names are used: rhasspyName, rhasspyRoom, rhasspyGroup, rhasspyChannels, rhasspyColors, rhasspySpecials.
Each of the keywords found in these attributes will be sent by update to Rhasspy to create the corresponding slot.

- **rhasspyName**

  Comma-separated "labels" for the device as used when speaking a voice-command. They will be used as keywords by Rhasspy. May contain space or mutated vovels.

  Example:
  `attr m2_wz_08_sw rhasspyName kitchen lamp,ceiling lamp,workspace,whatever`

- **rhasspyRoom**

  Comma-separated "labels" for the "rooms" the device is located in. Recommended to be unique.

  Example:
  `attr m2_wz_08_sw rhasspyRoom living room`

  Note: If you provide more than one room, the first will be regarded as *mainroom*, which has a special role, especially in dialogues.

- **rhasspyGroup**

  Comma-separated "labels" for the "groups" the device is in. Recommended to be unique.

  Example: `attr m2_wz_08_sw rhasspyGroup lights`

- **rhasspyMapping**

  If automatic detection (gDT) does not work or is not desired, this is the place to tell

Zum Anfang Alles laden

Example:

```
attr lamp rhasspyMapping SetOnOff:cmdOn=on,cmdOff=off,response="All
right"
GetOnOff:currentVal=state,valueOff=off
GetNumeric:currentVal=pct,type=brightness
SetNumeric:currentVal=brightness,cmd=brightness,minVal=0,maxVal=255
,map=percent,step=1,type=brightness
GetState:response=The temperature in the kitchen is at
[lamp:temperature] degrees
MediaControls:cmdPlay=play,cmdPause=pause,cmdStop=stop,cmdBack=prev
ious,cmdFwd=next
```

- **rhasspyChannels**

  Used to change the channels of a tv, set light-scenes, etc.
  key=value line by line arguments mapping command strings to fhem- or Perl commands.

  Example:

  ```
  attr TV rhasspyChannels orf eins=channel 201
  orf zwei=channel 202
  orf drei=channel 203
  ```

  Note: This attribute is not added to global attribute list by default. Add it using userattr or by editing the global userattr attribute.

- **rhasspyColors**

  Used to change to colors of a light
  key=value line by line arguments mapping keys to setter strings on the same device.

  Example:

  ```
  attr lamp1 rhasspyColors red=rgb FF0000
  green=rgb 008000
  blue=rgb 0000FF
  yellow=rgb FFFF00
  ```

  Note: This attribute is not added to global attribute list by default. Add it using userattr or by editing the global userattr attribute. You may consider using rhasspySpecials (*colorCommandMap* and/or *colorForceHue2rgb*) instead.

- **rhasspySpecials**

  Options to change a bunch of aspects how a single device behaves when addressed by voice commands. You may use several of the following lines.

  key:value line by line arguments similar to rhasspyTweaks.

  - **group**

    If set, the device will not be directly addressed, but the mentioned group - typically a FHEM structure device or a HUEDevice-type group. This has the advantage of saving RF ressources and/or fits better to already implemented logics.

Zum Anfang Alles laden

put this info in all devices under RHASSPY control in the same external group logic.
All of the following options are optional.

- **async_delay**
  Float nummeric value, just as async_delay in structure; the delay will be obeyed prior to the next sending command.
- **prio**
  Numeric value, defaults to "0". *prio* and *async_delay* will be used to determine the sending order as follows: first devices will be those with lowest prio arg, second sort argument is *async_delay* with lowest value first.
- **partOf**
  Will adress an entire group directly. This group has to exist in FHEM first (could be e.g. a *structure* or a *ZigBee*-group) and needs to be switched with the same command than the single device.

Example:

```
attr lamp1 rhasspySpecials group:async_delay=0.3 prio=1
group=lights
```

- **numericValueMap**

Allows mapping of numeric values from the *Value* key to individual commands. Might e.g. usefull to address special positioning commands for blinds.

Example:

```
attr blind1 rhasspySpecials numericValueMap:10='Event Slit'
50='myPosition'
```

Note: will lead to e.g. `set blind1 Event Slit` when numeric value 10 is received in {Value} key.

- **venetianBlind**

```
attr blind1 rhasspySpecials venetianBlind:setter=dim
device=blind1_slats stopCommand="set blind1_slats dim
[blind1_slats:dim]"
```

Explanation (one of the two arguments is mandatory):

- **setter** is the set command to control slat angle, e.g. *positionSlat* for CUL_HM or older ZWave type devices
- **device** is needed if the slat command has to be issued towards a different device (applies e.g. to newer ZWave type devices)
- **CustomCommand** arbitrary command defined by the user. Note: no variables will be evaluated. Will be executed if a regular nummeric command is detected.
- **stopCommand** arbitrary command defined by the user. Note: no variables will be evaluated. Will be executed if a stop command is detected.

If set, the slat target position will be set to the same level than the main device.

- **colorCommandMap**

Allows mapping of values from the *Color* key to individual commands.

Zum Anfang Alles laden

```
120='rgb 00FF00' 240='rgb 0000FF'
```

- **colorTempMap**

  Allows mapping of values from the *Colortemp* key to individual commands.

  Works similar to colorCommandMap

- **colorForceHue2rgb**

  Defaults to "0". If set, a rgb command will be issued, even if the device is capable to handle hue commands.

  Example:

  ```
  attr lamp1 rhasspySpecials colorForceHue2rgb:1
  ```

- **priority**

  Keywords *inRoom* and *outsideRoom* can be used, each followed by comma separated types to give priority in *Set* or *Get* intents. This may eleminate requests in case of several possible devices or rooms to deliver requested info type.

  Example:

  ```
  attr sensor_outside_main rhasspySpecials
  priority:inRoom=temperature
  outsideRoom=temperature,humidity,pressure
  ```

  Note: If there's a suitable "active" device, this will be given an even higher priority in most cases (e.g. "make music louder" may increase the volume on a switched on amplifier device and not go to an MPD device in the same room)

- **confirm**

  This is the more granular alternative to confirmIntents key in rhasspyTweaks (including *confirmIntentResponses*). You may provide intent names only or <*Intent*>=<*response*> pairs like `confirm: SetOnOff="$target shall be switched $Value" SetScene`.

- **confirmValueMap**

  Provide a device specific translation for $Value, e.g. for a blind type device *rhasspySpecials* could look like:
  ```
  confirm: SetOnOff="really $Value $target"
  confirmValueMap: on=open off=close
  ```

- **scenes**

  ```
  attr lamp1 rhasspySpecials scenes:scene2="Kino zu zweit"
  scene3=Musik scene1=none scene4=none
  ```

  Explanation:

Zum Anfang Alles laden

scene from the internal list, setting the combination *all=none* will exclude the entire device from beeing recognized for SetScene, *rest=none* will only include the labeled scenes. These values finally will be what's expected to be spoken to identificate a specific scene.

- **blacklistIntents**

  ```
  attr weather rhasspySpecials blacklistIntents:MediaControls
  ```

  Explanation:

  If set, the blacklisted intents will be deleted after automated mapping analysis.

- **rhasspyMsgCommand**

  Command used by RHASSPY to send messages to text dialogue partners. See also rhasspyMsgDialog attribute.

Intents

The following intents are directly implemented in RHASSPY code and the keywords used by them in sentences.ini are as follows:

- Shortcuts
  (keywords as required by user code)
- SetOnOff
  {Device} and {Value} (on/off) are mandatory, {Room} is optional.
  Note: As **experimental** feature, you may hand over additional fields, like {Device1} ("1" here and in the follwoing keys may be any additonal postfix), {Group}/{Group1} and/or {Room1}. Then the intent will be interpreted as SetOnOffGroup intent adressing all the devices matching the {Device}s or {Group}s name(s), as long as they are in (one of) the respective {Room}s.
  The only restriction is: The intented {Value} (or, for other multicommand intents: Color etc.-value) has to be unique.
- SetOnOffGroup
  {Group} and {Value} (on/off) are mandatory, {Room} is optional, *global* in {Room} will be interpreted as "everywhere".
  Experimental multicommand feature should work also with this intent, (redirecting to itself and adding devices according to the additional keys).
- SetTimedOnOff
  Basic keywords see SetOnOff, plus timer info in at least one of the fields {Hour} (for relative additions starting now), {Hourabs} (absolute time of day), {Min} (minutes) and {Sec} (seconds). If {Hourabs} is provided, {Min} and {Sec} will also be interpreted as absolute values.
- SetTimedOnOffGroup
  (for keywords see SetOnOffGroup)
- GetOnOff
  (for keywords see SetOnOff)
- SetNumeric
  Dependend on the specific surrounding informations, a combination of {Device}, {Value} (nummeric value), {Change} and/or {Type} are sufficient, {Room} is optional. Additional optional field is {Unit} (value *percent* will be interprated as request to calculate, others will be ignored). {Change} can be with one of ({Type})

Zum Anfang Alles laden

    - tempUp, tempDown (temperature/desired temp)
    - setUp, setDown (setTarget)
    - cmdStop (applies only for blinds)

  allowing to decide on calculation scheme and to guess for the proper device and/or answer. **experimental multicommand** feature should work also with this intent (switching intent to SetNumericGroup).
- SetNumericGroup
  (as SetNumeric, except for {Group} instead of {Device}).
- GetNumeric
  (as SetNumeric)
- GetState
  To querry existing devices, {Device} is mandatory, keys {Room}, {Update}, {Type} and {Reading} (defaults to internal STATE) are optional. By omitting {Device}, you may request some options RHASSPY itself provides (may vary dependend on the room). {Type} keys for RHASSPY are *generic*, *control*, *info*, *scenes* and *rooms*.
- MediaControls
  {Device} and {Command} are mandatory, {Room} is optional. {Command} may be one of *cmdStop*, *cmdPlay*, *cmdPause*, *cmdFwd* or *cmdBack*
- MediaChannels
  (as configured by the user)
- SetColor
  {Device} and one Color option are mandatory, {Room} is optional. Color options are {Hue} (0-360), {Colortemp} (0-100), {Saturation} (as understood by your device) or {Rgb} (hex value from 000000 to FFFFFF) **experimental multicommand** feature should work as well.
- SetColorGroup
  (as SetColor, except for {Group} instead of {Device}).
- SetScene
  {Device} and {Scene} (it's recommended to use the $lng.fhemId.Scenes slot to get that generated automatically!), {Room} is optional, {Get} with value *scenes* may be used to request all possible scenes for a device prior to make a choice.
- GetTime
- GetDate
- Timer
  Timer info as described in *SetTimedOnOff* is mandatory, {Room} and/or {Label} are optional to distinguish between different timers. {CancelTimer} key will force RHASSPY to try to remove a running timer (using optional {Room} and/or {Label} key to identify the respective timer), {GetTimer} key will be treated as request if there's a timer running (optionally also identified by {Room} and/or {Label} keys). Required tags to set a timer: at least one of {Hour}, {Hourabs}, {Min} or {Sec}. {Label} and {Room} are optional to distinguish between different timers. If {Hourabs} is provided, all timer info will be regarded as absolute time of day info, otherwise everything is calculated using a "from now" logic.
- SetTimer
  Set a timer, required info as mentionned in *Timer*
- GetTimer
  Get timer info as mentionned in *Timer*, key {GetTimer} is not explicitely required.
- ConfirmAction
  {Mode} with value 'OK'. All other calls will be interpreted as CancelAction intent call.
- CancelAction
  {Mode} is recommended.
- Choice
  One or more of {Room}, {Device} or {Scene}
- ChoiceRoom
  {Room}

Zum Anfang Alles laden

- ReSpeak

Readings

There are some readings you may find usefull to tweak some aspects of RHASSPY's logics:

- siteId2room_<siteId>
  Typically, RHASSPY derives room info from the name of the siteId. So naming a satellite *bedroom* will let RHASSPY assign this satellite to the same room, using the group sheme is also supported, e.g. *kitchen.front* will refer to *kitchen* as room (if not explicitly given).
  You may overwrite that behaviour by setting values to siteId2room readings: `setreading siteId2room_mobile_phone1 kitchen` will force RHASSPY to link your satellite *phone1 kitchen* to kitchen as room.
- room2siteId_<room>
  Used to identify the satellite to speak messages addressed to a room (same for playing sound files). Should deliver exactly one possible siteId, e.g. <lingingroom.04>
- siteId2doubleSpeak_<siteId>
  RHASSPY will always respond via the satellite where the dialogue was initiated from. In some cases, you may want additional output to other satellites - e.g. if they don't have (always on) sound output options. Setting this type of reading will lead to (additional!) responses to the given second satellite; naming scheme is the same as for site2room.
- sessionTimeout_<siteId>
  RHASSPY will by default automatically close every dialogue after an executable commandset is detected. By setting this type of reading, you may keep open the dialoge to wait for the next command to be spoken on a "by siteId" base; naming scheme is similar as for site2room. Intent *CancelAction* will close any session immedately.
- siteId2ttsDevice_<siteId>
  experimental! If an AMADDevice TYPE device is enabled for rhasspySpeechDialog, RHASSPY will forward response texts to the device for own text-to-speach processing. Setting this type of reading allows redirection of adressed satellites to the given AMADDevice (device name as reading value, 0 to disable); naming scheme is the same as for site2room.

Perl specials

Wenn Sie einige Aufgaben automatisieren wollen, dann sollten Sie die Befehle at oder notify nutzen. Für komplexere Aufgaben sollten Sie lieber ein SHELL-Skript oder einen PERL "oneliner" als das at/notify argument anwenden. Dieser Abschnitt gibt Ihnen einige Tipps zur Anwendung der PERL-oneliner.

- Um PERL-"oneliner" zu testen, geben Sie diese am "telnet" Prompt (oder in der FHEMWEB Text-Eingabezeile) eingeschlossen von geschweiften Klammern {} in einer Zeile ein. Die letzte Beispielzeile schreibt nur etwas in die Logdatei, während das Ergebnis der anderen Zeilen direkt auf der Webseite sichtbar ist.

  Beispiele:

  { "Hello" }
  { 1+3*4 }
  { `ls /etc` }
  { Log 1, "Hello" }