Table of contents

1) Introduction 2
2) Characteristics 2
3) Requirements 3
3.1) SMS 3
3.2) Emails 3
3.3) Telegram 4
4) Command line parameters for message2action 4
5) Description of the ini file 4
5.1) General configuration of the program 5
5.2) Sending and receiving SMS 5
5.3) Sending and receiving emails 6
5.4) Sending and receiving Telegram messages 6
5.5) Commands in messages to the Pi 7
5.6) "Build-in" commands 7
6) Commands in messages 8
6.1) SMS 8
6.2) Mail 8
6.3) Telegram 8
7) Sample files 9
7.1) SMS 9
7.2) Mail 9
7.3) Telegram 9
8th) Installation and configuration on the Pi 9
8.1) Source code 10
8.2) SMS tools 10
8.3) Mail, fetchmail 10
8.4) Email, msmtp 10
8.5) Telegram 10
9) systemd services 10
9.1) message2action.service 10
9.2) getmails.service and getmails.timer 11
9.3) sendmails.service 11
9.4) send_telegram_to_bot.service 11
9.5) telegram_check_updates.service 11
10) Compiling and linking the source code 11
11) Security 11
12) Future features 11
13) Message interface 12

13.1) SMS 12
13.1.1) Naming convention and structure of the text files 13
13.2) Mail 14
13.2.1) Naming convention and structure of the text files 14
13.3) Telegram 15
13.3.1) Naming convention and structure of the text files 15
14) Releases of the program 16
14.1) Release 0.27 16
14.2) Release 0.28 16
14.3) Release 0.30 16
14.4) Release 0.31 16
14.5) Release 0.32 16
14.6) Release 0.33 16
14.7) Release 0.34 16
14.8) Release 0.35 16
14.9) Release 0.36 17
14.10) Release 0.39 17
14.11) Release 0.40 17
14.12) Release 0.50 17
FAQ (Frequently asked questions) 17
15) Contact the author of the program 18
16) Legal 18

1) Introduction

What is message2action?
The program processes incoming messages such as SMS, emails, Telegram chat messages and reacts to the content in the messages. This is sometimes also called “server control”.
The program was developed for the Raspberry Pi (hereinafter just “Pi”) - or better: Raspbian OS - in C++. It has never been tested on other platforms.
The message that is sent to the Pi can contain commands such as “stop” or “status” and you can specify in a configuration file (ini file) what should happen when such a command is recognized.
In the current version, message2action uses a file interface. In other words: files are read in directories and then these files are evaluated. It is not an online program.
The trigger why I developed the program was the desire to start my PC but without the PC being accessible via the Internet. The original idea was to send an SMS to the Pi and it would send a WOL (wake on LAN) to the PC. With message2action you avoid having to use Dyndns or port forwarding (in the router). And the PC is still not accessible via the Internet.
With this idea, I thought it would be a good idea to have a standardized interface on a Pi. The interface can also be used for other purposes such as status requests from devices in the home network such as temperature sensors or similar.

2) Characteristics

Here is a list of the most important features:
-Text files are read (emails, SMS and Telegram) and commands are searched for in the text files and then executed. Other programs must take care of receiving these text files.

-Text files are generated (emails, SMS and Telegram) when a response is to be sent back to the sender (for example in response to a status request). Other programs must be responsible for sending the answers to the recipient.
-Information, warnings and errors are recorded in a log file so that you can trace the course of the program even afterwards. This is helpful when troubleshooting.
-The maximum size of the log file can be set.
-The incoming and outgoing text files are stored on the Pi and a maximum age for the files can be specified. Files that are older will be deleted.
-A maximum of 10 email addresses, 10 mobile numbers and 10 chat IDs for Telegram can be specified as contact partners.
-Warning messages are sent when a message from an unknown sender is detected. The unknown sender receives no information.
-A maximum of 10 commands (scripts, programs) can be configured.
-Internal commands are available to stop the program, restart the Pi or shut down.

3) Requirements

What is needed to use message2action?

3.1) SMS

You need a surf stick that can be connected to the Pi via USB. You also need a SIM card for the surfstick to receive and send SMS.
Currently I'm using "SMS Server Tools 3" (http://smstools3.kekekasvi.com) to send and receive SMS with the Surfstick. The software can be installed with this command:

```
sudo apt-get install smstools
```

I connected a Vodafone Surfstick called K3-772Z with a prepaid SIM card from T-Mobile to the Pi. This is a UMTS stick. But to receive or send SMS, the UMTS network is not used but the GSM network. Therefore, I can continue to use the UMTS stick despite the UMTS network being switched off (mid-2021).
To switch the surfstick from data mode to modem mode, you can use the USB Modeswitch program. (http://www.draisberghof.de/usb_modeswitch). You should also install the data package. Both can be installed with this command:

```
sudo apt-get install usb-modeswitch usb-modeswitch-data
```

The “SMS Server Tools” mentioned save received SMS as text files. The default directory is /var/spool/sms/incoming. message2action reads all files in this directory and looks for commands in the files. After evaluation, the processed files are moved to a separate directory (processed files).
If message2action wants to send an SMS, the file is stored in the outgoing directory of SMS Server Tools (default: /var/spool/sms/outgoing).
Note: The directories mentioned are maintained in the ini file and can also have different names.

3.2) Emails

You need an account with a mail provider. This account (more precisely: the associated email address) is the contact for the Pi, in the following Pi account. To send emails to the Pi, another account is required, in the following sender account. So to send emails to the Pi, you send an email from the sender account to the Pi account. And when the Pi wants to send an email to the sender, the Pi account sends it to the sender account. I use fetchmail to read emails from the Pi account. To send emails from the Pi to the sender account, I use msmtp.

```
fetchmail saves all received emails as text files. You can specify the directory
where the files should be stored.
```

message2action reads all files in this directory. After the files are processed, they are moved to a Processed directory.
To install fetchmail you can do this:

```
sudo apt-get install fetchmail
```

In order for message2action to send emails, a text file is created that is used as input for a script. This script is called from sendmails.service (this systemd file is provided by me). The script calls msmtp to send a mail from the Pi account to the sender account. To install msmtp this can be called:

```
sudo apt-get install msmtp
```

Both fetchmail and msmtp still need to be set up by the user.

3.3) Telegram

You need either an app (smartphone) or a web browser to log in to Telegram. First, an account and a bot are set up. For the bot you need the token that is linked to the bot. The token looks something like this:
ABGxGlaeQTbUR_fWZFG2RT8u6MhABCxLwmw
The chat ID with which to communicate is still required. This is a nine-digit number. Please search the internet for instructions on how to find out the chat ID and token. The chat ID and token must be specified in the message2action ini file. This is the file message2action.ini (located in the same directory as the executable program).
Sending Telegram messages is done via curl. If curl is not installed, it can be installed with this call:

```
sudo apt-get install curl
```

To send Telegram messages that message2action has generated, the send_telegram_to_bot.sh script (provided by me) is sent. This script scans the outgoing message directory and sends each one to the bot using curl.
To receive Telegram messages on the Pi and store them as a text file on the Pi, the program telegram_check_updates is used. This program is also provided by me. To start the program when the system starts, the systemd unit telegram_check_updates.service can be used (I provide it).

4) Command line parameters for message2action

If you call message2action without parameters, a small help appears that lists the available commands.
These are the parameters for message2action:

`?`: A little help is displayed.

`start`: Starts the program in standard mode: it searches directories (as specified in the ini file) for text messages and then evaluates them. To stop the program again, you can simply press a so-called readable key on the keyboard. Readable actions include escape, spacebar, all numbers and letters. The Shift and Control keys are not included. Additionally, these 3 options are available:

1) Creating a stop file (see stop parameter)
2) Receipt of a message with the command to stop
3) In another terminal, message2action is called with parameter stop

`daemon`: Starts the program in daemon mode: it searches directories (as specified in the ini file) for text messages and then evaluates them. This mode outputs very little to the screen and is therefore suitable for starting with a systemd unit. In daemon mode, the program cannot be stopped by pressing a key. It can only be stopped in 3 ways:

4) Creating a stop file (see stop parameter)
5) Receipt of a message with the command to stop
6) In another terminal, message2action is called with parameter stop

`stop`: A so-called stop message is generated, which can be evaluated by another instance of the program and thus causes the latter program to stop.

`config`: With this parameter the current configuration is displayed on the screen as it is saved in the ini file. This function is useful to check whether changes to the ini file have been correctly understood by the program. The ini file is only read when the program starts, and no longer afterwards. This means that after every change to the ini file, the program must be stopped and restarted.

5) Description of the ini file

The configuration of the program is stored in a so-called ini file and can be edited with a text editor. The name of the ini file is message2action.ini. This file is in the same directory as the executable program.
Values for parameters can be specified in the ini file, for example in which directory processed files should be stored. Every letter after the character '#' (without quotation marks) is considered a comment. The delivered ini file shows what comments look like.
The '=' (without quotation marks) must appear directly after a parameter. The value must be specified immediately after the '='. See the following examples:
CMD03='/home/pi/demo' is correct. But

CMD03 ='/home/pi/demo' is wrong and
CMD03= '/home/pi/demo' is also incorrect.

5.1) General configuration of the program

`capitalize_values,` Default: `true;` possible values: true, false.
This parameter determines whether all incoming commands should be converted to capital letters (true) or not (false). This parameter is helpful to avoid errors such as “Stop”, “stoP” instead of “stop” when typing commands.

`file_logfile,` Default: `/home/pi/message2action/message2action.log;` possible values: full path to a log file (including the name of the file).
Log information is written to this file while the program is running. The information is provided with a timestamp and tags such as “Info”, “Warning” and “Error”.
"Info": This is simply information such as the command found in an SMS or the status of the program.
"Warning": Something is not working properly and the user should be informed about it. For example, an unknown phone number was recognized as the sender in an SMS.
"Error": An error has occurred that may limit the use of the program. For example, the log file could not be opened for writing or moving the processed file failed due to the necessary rights.

`max_size_logfile,` default: `500;` possible values: any integer.
This value determines the maximum size of the log file in kBytes (1 kByte = 1024 bytes). When this size is reached, the log file is divided into 2 halves and the older part is deleted. This means that the most recent part of the content always remains.
Practice has shown that the 500 kByte log file contains approximately xyz lines.

`garbage_collection_days,` default: `30;` possible values: any integer.
This parameter sets the maximum age, expressed in whole days, for processed files. The program searches the directories where processed files are stored. All files older than the set number of days will then be deleted.
This function protects against the space in the file systems being exhausted because any old data is still there.

5.2) Sending and receiving SMS

`receivingfrom_cell_phone00 to receivingfrom_cell_phone09,` default: `4999;` possible values: Any international telephone number without a leading '+' sign (without quotation marks).
The 10 telephone numbers determine who may process SMS messages and to whom responses (e.g. the status of the program) may be sent back. Perhaps you want to allow family members or members of a team to allow the program to process text messages from them. Every telephone number has the same rights; there is no role model for the telephone numbers. This function protects against unauthorized persons being able to execute commands.
If a telephone number was recognized that is not listed in the ini file, the command is ignored and a warning is sent to all telephone numbers. This warning looks like this:
"Warning: invalid sender-phonenumber in incoming SMS: %s!". The variable %s contains the recognized telephone number from which the SMS came.

`path_incoming_SMS,` default: `/var/spool/sms/incoming/;` possible values: full path where incoming SMS can be found.
This path indicates where incoming SMS can be found as a text file. The program looks for text files in this path.

`path_outgoing_SMS,` default: `/var/spool/sms/outgoing/;` possible values: complete path in which outgoing SMS should be stored.
This is the path in which the program stores text files to be sent via SMS.

`path_processed_SMS,` default: `/var/local/message2action/processedsms/;` possible values: full path where processed SMS can be found.

The program moves the received files into this directory after the program has processed the content (regardless of the status). This means that the input directory is always emptied. This way you can check whether the program is processing the incoming SMS.

`wait_time_next_sms`, default: `2;` possible values: all integers greater than 0.
The value indicates the number of seconds the program should wait after searching the incoming directory for SMS messages in text format and processing the last file. So you can set a waiting time until the next search.
Note: The program for generating the SMS as a text file also has a waiting time and these two waiting times should be coordinated.

`inbound_sms_file_flag`, default: `true;` possible values: true, false.
This flag determines whether incoming SMS should be processed (true) or not (false).

5.3) Sending and receiving emails

`mail_address_00 to mail_address_09`, default: dummyuser@pathfinder-outlander.lit; possible values: any email address.
A maximum of 10 email addresses can be specified from which text messages will be accepted for processing. This procedure is the same as for telephone numbersProcessing SMS. If an email address that is not specified here was recognized as the sender, a warning will be sent to all specified email addresses. This warning looks like this: "Warning: invalid mail address in incoming mail: %s!". The variable %s contains the recognized email address. This function ensures that only known mail senders can execute commands.
Tip: I also recommend specifying the email address from which your mail provider sends emails with an “unknown email address”. For example, for Deutsche Telekom's T-Online, this address is mailer-daemon@t-online.de .

`path_incoming_mail`, default: `/home/pi/mail/;` possible values: full path in which incoming emails can be found as text files.
This is the input directory in which the program looks for emails.

`path_outgoing_mail`, default: `/home/pi/mail/outgoing/;` possible values: full path in which outgoing emails can be found as text files.

`wait_time_next_mail`, default: `30;` possible values: all integers
The value specifies the number of seconds the program should wait after searching the incoming directory for text-format mail messages and processing the last file. So you can set a waiting time until the next search.
Note: The program for creating the email as a text file also has a waiting time and these two waiting times should be coordinated.

`inbound_mail_file_flag`, default: `true;` possible values: true or false.
This flag determines whether incoming emails should be processed (true) or not (false).

5.4) Sending and receiving Telegram messages

`telegram_bot_token`, Default: `my_telegram_bot_token;` possible values: the token for Telegram.
This token refers to the bot in Telegram. Currently only 1 bot can be configured for the program. But up to 10 chats can be specified. The token is required with the chat ID to collect and send chat content.

`telegram_chat_id_00 to telegram_chat_id_09`, default: `999999999;` possible values: any allowed chat ID.
That's a maximum of 10 chat IDs that the bot can be invited to. If a chat ID that was not specified in the ini file was detected when fetching chat messages, the command will be ignored and a warning will be sent to all chats. The warning looks the same as for SMS and emails.
This feature prevents someone else from executing commands on the Pi.

`path_incoming_telegram`, default: `/home/pi/telegram/`; possible values: full path where incoming Telegram messages can be found as text files.

`path_outgoing_telegram`, default: `/home/pi/telegram/outgoing_to_bot/`; possible values: full path where outgoing Telegram messages can be found as text files.

`path_processed_telegram`, default: `/home/pi/message2action/telegramprocessed/` ; possible values: full path where processed Telegram messages can be found as text files.

`wait_time_next_telegram`, default: `2`; possible values: all integers greater than 0
The value indicates the number of seconds the program should wait after searching the incoming directory for Telegram messages in text format and processing the last file. So you can set a waiting time until the next search.
Note: The program for generating the Telegram message as a text file also has a waiting time and these two waiting times should be coordinated.

`inbound_telegram_file_flag`, default: `true`; possible values: true or false.
This flag determines whether incoming Telegram messages should be processed (true) or not (false).

5.5) Commands in messages to the Pi

First of all, a distinction is made between so-called “built-in” and “external” commands that can be sent to the Pi. Built-in commands are those that are built into the program. Examples include stop, shutdown and WOL (Wakeup on LAN). Keywords that should be used to execute the built-in commands can be specified in the ini file.
External commands are those that allow programs or scripts to be executed on the Pi. The full path to such a program is a value to the external command.

5.6) "Build-in" commands

The following possible values for the built-in commands are: 1 word of any length is permitted.

`message2action_stop_command`, default: `stop`
If you send the command “stop” (without quotation marks) to the Pi, the program ends. If you don't like the word, you could also set it to "stop".
If you have identified any security concerns, it is useful to pause processing of further messages.

`shutdown_pi_command`, default: `shutdown`
If you send the command "shutdown" (without quotation marks) to the Pi, the entire Pi will shut down.
The purpose for this program is the same for the message2action_stop_command. If concerns are identified, you can shut down the entire Pi.

`reboot_pi_command`, default: `reboot`
This command ends the program and restarts the entire Pi.

`status_pi_command`, default: `status`
The status of the program can be queried with this command. The status contains a message for each channel, the number of messages processed, warnings and errors since the program was started.

`message2action_help_command`, default: `help`
This command sends back a small help text that lists the possible commands.

`message2action_questionmark_command`, default: `?`
The question mark triggers the same thing as the help command. Maybe the question mark is easier to remember and it's also shorter.

`message2action_config_command`, default: `config`
This command outputs the current configuration of the program. The result is the same as entering “config” (without quotes) as a parameter on the command line.

`WOL00 to WOL09, default:`
WOL00=1A:1B:1C:1D:1E:1F;host=summerrain
WOL01=2A:2B:2C:2D:2E:2F;host=skyfall
WOL02=3A:3B:3C:3D:3E:3F;ip=10.10.10.10
WOL03=4A:4B:4C:4D:4E:4F;ip=10.10.10.11

The values behind the parameters WOL00 to WOL09 are MAC addresses of network adapters. The MAC address is used to send the so-called magic packet (Wake on LAN) to a system via UDP broadcast. Receiving the magic package triggers the receiving device to wake up from sleep mode or cold start, for example. A maximum of 10 MAC addresses can be set.
The MAC address can be followed by a ';' (without quotation marks) either a TCP-IP hostname or an IPV4-IP address can be assigned separately.
A hostname is specified by:
host=myhostname
An IP address is specified by:
ip=123.123.123.123
The hostname or IP address can be used to check the online status after sending the WOL command. This is done by calling ping.

`online_status_command_suffix`, default: `stat`
This suffix can be appended to the WOL command (WOL00 to WOL09) to query the online status of a device.
Example: You want to know whether the device that belongs to the address of WOL04 is online. Then you send WOL04stat to the program. The program checks the online status of the device using a ping command and sends back the answer.

`check_online_status_waittime`, default: `30`
The number indicates the waiting time in seconds between sending a WOL command and querying the online status. Some devices, such as a PC, need several seconds after waking up or starting cold until the network stack is available and can respond.
With the value 0 this function is switched off and you get no response.

`CMD00 to CMD09`, default for CMD00: `date > /tmp/test.log`.
With these parameters you can execute up to 10 freely defined commands, programs or scripts by sending CMD00 to CMD09 to the Pi. This can be useful, for example, to restart other programs such as a web server or a database. You can expand the functionality of the program in this way.

6) Commands in messages

The way you send commands to the Pi depends on the type of messages. The types are presented below.

6.1) SMS

When you send an SMS to the Pi, you simply write the command (for example “status” {without quotation marks} in the SMS.

6.2) Mail

The command only appears in the subject and not in the body of the email. The main text can be left completely blank as it will not be evaluated.

6.3) Telegram

The command is entered into the chat with the bot. Here you can see an example of 2 commands:

First status was entered and the response was the status of the program. The Wake on LAN for device 00 was then entered and the online status of the device was returned.

7) Sample files

In the samples directory I have stored sample files for incoming messages that can be used for testing. With these example files you do not need to have an SMS device installed, no mail programs installed and no need to configure Telegram with a chat bot. Instead, you can simply copy these sample files into the input directory and see what happens.

7.1) SMS

All example files for SMS are called GSM1.*.
Example: You want to start a program that is located on the Pi under /home/pi/testdirectory/testprogram.sh and is to be called with the parameters invoice tell. The program should be triggered with the command CMD03. This must be adjusted in the ini file:

```
CMD03=/home/pi/testdirectory/testprogram.sh invoice tell
```

Then adapt the example file so that the CMD03 command can be found there. You can use the file GSM1.CMD03 for this. The first line must contain a telephone number that is maintained in the ini file.
The command CMD03 is entered in the last line. As a final step, this file is copied to the directory for incoming SMS (default: /var/spool/sms/incoming).
Note: There are no plausibility checks for the commands CMD00 to CMD09.

7.2) Mail

All example files for emails are called mail.*.
Example: If you want to execute the same command as in the example for SMS, you have to send an email to the email address that is read by the email fetch program (default: fetchmail). When the pickup program has picked up the email, you will receive a text file. In the chapter about fetchmail describes how to set up the program.
In the samples directory you can find several sample files that were received via fetchmail. These files can be customized and then copied to the directory where incoming emails for the program can be found (default: emails in the home directory).

7.3) Telegram

All example files are called telegram.*.
The procedure is the same as for SMS and emails. The customized file is copied to the directory for incoming Telegram messages (default: telegram/outgoing_to_bot in the home directory).

8) Installation and configuration on the Pi

On my Pi I installed the program in my home directory under message2action. All following statements and commands refer to this directory.
Run this command in your home directory to download the repository:

```
git clone https://github.com/Sunblogger/message2action
```

Now several files appear in the message2action directory:
`message2action:` the executable program
`message2action.ini:` ini file containing the configuration of the program. This file needs to be customized.

`message2action_de.pdf, message2action_en.pdf:` German and English documentation.
sendmails.sh: This script is called by the systemd service sendmails.service to send mails generated by message2action.

These directories are created:
`fetchmail:` Configuration of fetchmail that must be done by the user.
`msmtp:` Configuration of msmtp that must be done by the user.
`telegram:` Directory for storing incoming Telegram messages.
`mailprocessed:` Directory for storing emails processed by message2action.
`smsprocessed:` Directory for storing emails processed by message2action.
`telegramprocessed:` Directory for storing Telegram messages processed by message2action.
`samples:` Sample files for SMS, emails and Telegram messages. See Chapter Sample files. These files can be used for testing.
`systemd:` Files for setting up systemd services, among other things, for automatically starting and stopping message2action and other helper programs. For details see chapter systemd.
There is an installation file called installservicetimer.sh in the directory. This script can be used to install the systemd services.

8.1) Source code

`message2action.cpp:` Source code in C++ for the program.

8.2) SMS tools

If you use the SMS tools, the /etc/smsd.conf file must be adapted. The device for sending and receiving SMS must be set up. Incoming SMS must be allowed with incoming = yes.

8.3) Mail, fetchmail

If you use fetchmail to collect emails, the following applies:
In the fetchmail directory:
The fetchmailparser.sh file does not require any adjustments.
`fetchmailrc:` The mail server, the account plus password and the ssl fingerprint must be entered here.

8.4) Email, msmtp

If you use msmtp to send emails, you have to adapt the .msmtprc file in the msmtp directory: Enter the email account plus password and more.

8.5) Telegram

In the telegram directory:
`send_telegram_to_bot.sh:` No customization required.
`telegramparser.sh:` No customization required.

A program called telegram_check_updates is delivered to extract text messages from a Telegram chat and save them as a text file. If the program is called without parameters, you will receive a little help. The telegram_check_updates.txt file contains further explanations.

9) systemd services

The systemd directory contains service and timer files that are delivered optionally and are not absolutely required for operation. The following sections explain the files. To install the service files, the installservicetimer.sh script can be used. The script copies the service files to the /lib/systemd/system directory. After copying, each service file is registered with systemctl.

9.1) message2action.service

`message2action.service` starts the program when the operating system starts and stops the program when the operating system shuts down.
To check the executability of the service file, this can be entered:

```
sudo systemctl status message2action
```

If the program should be started as a service, this can be entered:

```
sudo systemctl start message2action
```

To stop the program as a service, you can enter this:

```
sudo systemctl stop message2action
```

9.2) getmails.service and getmails.timer

`getmails.service` calls fetchmail to retrieve mails and saves the mails as text files in the incoming mail directory. The fetchmail directory contains a configuration file for fetchmail and a parser script that is called in the configuration file. The parser script is used to only get the sender and the subject of the email (the main text of the email is discarded). It also ensures that the emails are saved under a unique file name.
`getmails.timer` is used to call getmails.service periodically. By default this happens every 5 minutes. Note: The mail provider may react unpleasantly if you retrieve emails too often.

9.3) sendmails.service

`sendmails.service` calls a script called `sendmails.sh` with the start parameter to send emails in a loop. `sendmails.service` can stop this by calling the `sendmails.sh` script with the stop parameter (this can also be done manually). `sendmails.sh`, like the service files, is supplied as an example of how to send emails with the msmtp program. sendmails.sh runs in a loop until it finds a file called `/tmp/sendmails.stop` (a so-called stop file). You can create this stop file by calling sendmails.sh with the stop parameter.

9.4) send_telegram_to_bot.service

This service is used to send Telegram messages (generated by message2action) to the bot's chat ID. The send_telegram_to_bot.sh script is called in the service file, which looks for text files in the home directory, which are then sent to the chat using curl.

9.5) telegram_check_updates.service

The service telegram_check_updates.service must be adapted so that its own token is entered in the line starting with “ExecStart=”.

10) Compiling and linking the source code

The source code can be adapted according to your own needs. To create the executable file, the makefile named makefile in the message2action directory can be used.
If you have made any adjustments, I would be happy to be informed about them. Maybe I'll make these adjustments.
With the define called DEVSTAGE in the source code, the program outputs more details about the ongoing operations. This can be helpful for debugging. The define can be found at the very beginning of message2action.cpp.

11) Security

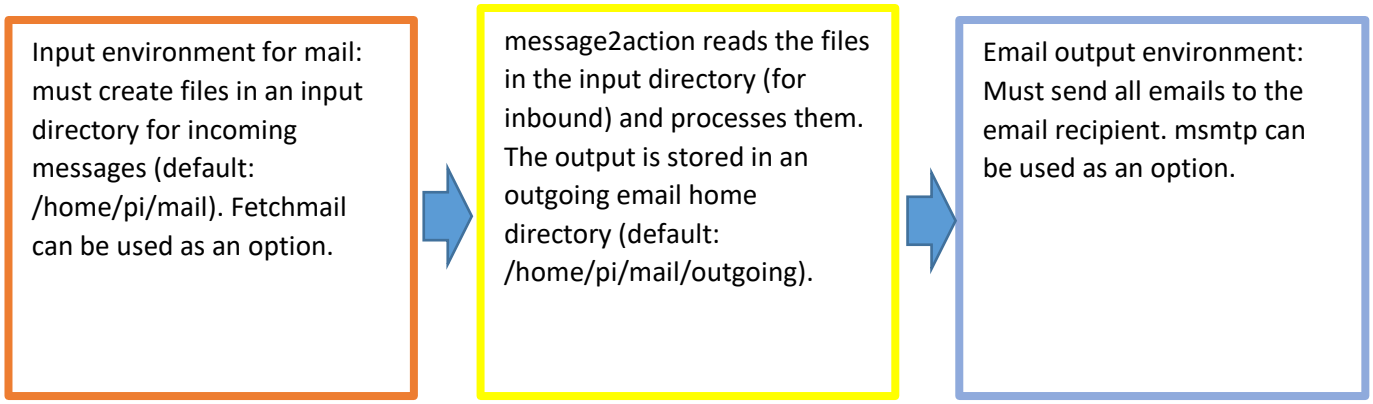
The “SMS Server Tools 3” writes the mobile phone number into the SMS file. The program checks this cell phone number for validity. In the ini file you can specify 10 permitted mobile phone numbers. This means that if someone else sends a text message to your Pi, the program will not work with it. All valid mobile numbers will receive a message with a warning when an invalid phone number is detected. The same applies to emails. You can specify 10 valid email addresses in the INI file. If your account receives an email from an unknown account, all email addresses will receive a warning email. This can be a problem if you receive spam emails because every time a spam email is received, an email is sent to all registered email accounts in the .ini file. The same applies to Telegram messages. In the ini file you can specify 10 valid chat IDs. If your bot still receives a message from a chat that is not specified, all chat IDs will receive a warning message.

12) Future features

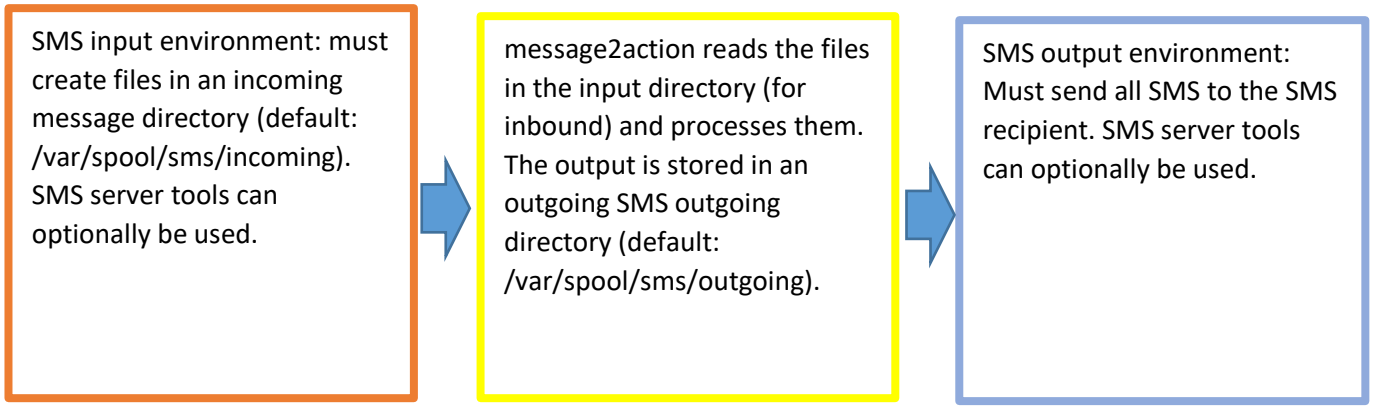
a) It might be interesting to query the GPIOs and, for example, short-circuit pins via command.
b) Currently you have to use a text editor to adapt the ini file. A program with a GUI could be an alternative to this.
c) Maybe it makes sense to save the contents of the log file (message2action.log) in the database instead of in a text file. This could be helpful when analyzing errors.
d) It could be interesting to also use the mailaccounts built into the Raspberry Pi OS for communication. This would allow you to send a message to the program directly from the Pi without having to go via SMS or the Internet.
e) Currently, SMS, emails and Telegram messages are handled on a file basis. In the future, it could also be interesting to handle emails and Telegram messages online, so that you no longer have a file interface

13) Message interface

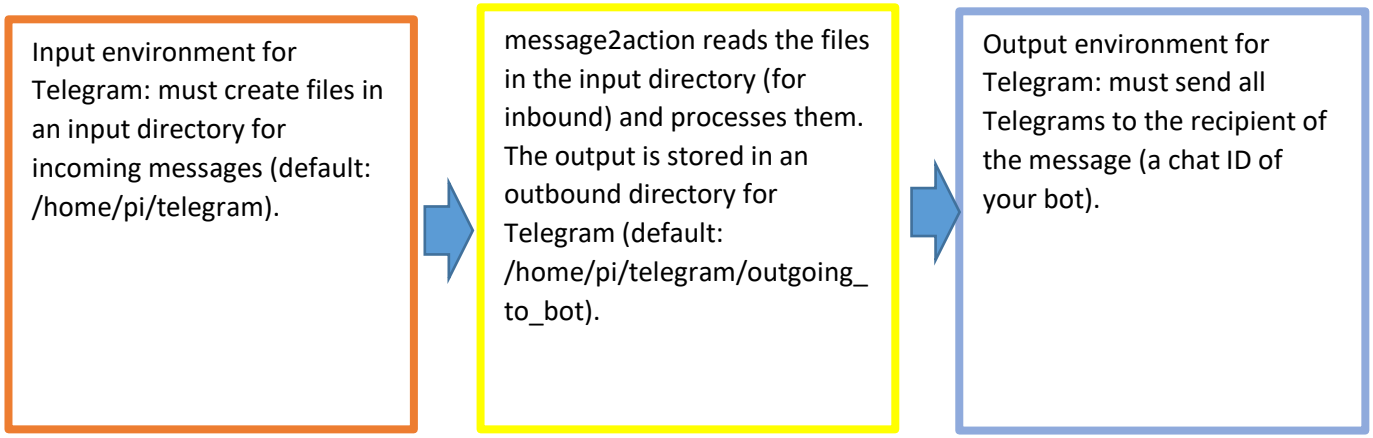
The interfaces to message2action are asynchronous and file-based. This means that all messages are files sent to message2action. And all messages returned by message2action are also files. The consequence of this design is that you need to set up an environment to create the files that you want message2action to process. And your environment must be able to accommodate the files set up by message2action in order to deliver them to the intended recipient. For a general overview of email, see the diagram below:



Here is the corresponding diagram for SMS:



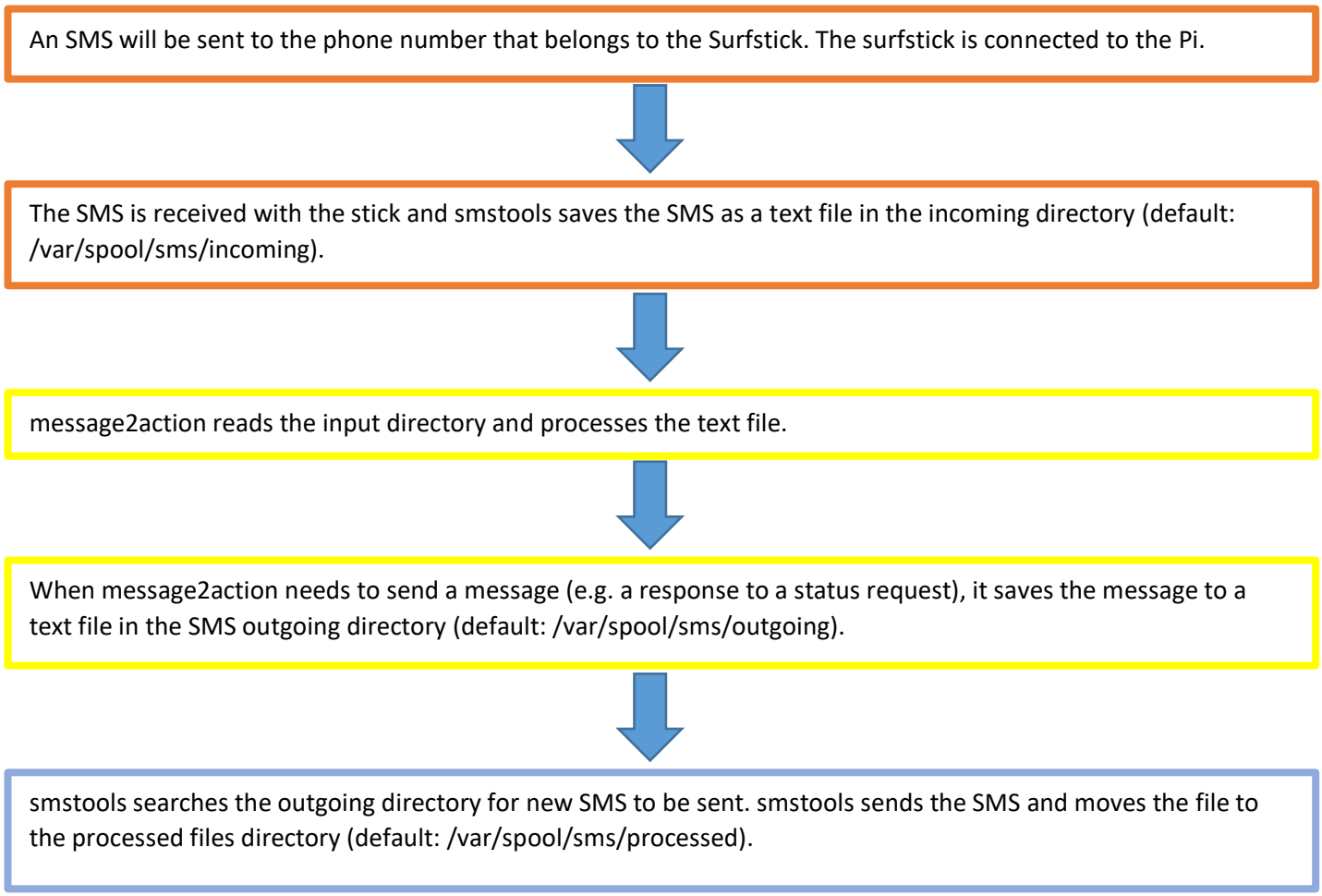
Here is the corresponding diagram for Telegram messages:



This design makes it easier to integrate the next type of messages to be processed by messag2action. Online processing would be more modern and also easier for the user. But then all channels would have to be implemented in the software. Instead, ready-made programs such as smstools, fetchmail, msmtp, curl, etc. can simply be used.

13.1) SMS

This is the flow for processing SMS on your Pi using message2action. The colors of the input environment, message2action and the output environment are maintained as above.



13.1.1) Naming convention and structure of the text files

Here is an example of a file created by SMStools for an incoming SMS. (Real values are replaced by “DUMMY”):

```
From: 4999
From_TOA: DUMMY
From_SMSC: DUMMY
Sent: DUMMY
Received: DUMMY
Subject: DUMMY
Modem: DUMMY
IMSI: DUMMY
Report: DUMMY
Alphabet: DUMMY
Length: 5

CMD01
```

After the keyword From:, the first line contains the cell phone number from which the SMS comes. This value is used in message2action to check whether an SMS came from a valid mobile phone. The very last line contains the text that was sent in the SMS. This is the command you send to your Pi and is read by message2action. Before the very last line you will find a blank line.
The files are saved by SMStools with this naming convention: GSM1.abxyz, where abxyz represents a random 5-character string. Here is an example of a file sent by message2action:

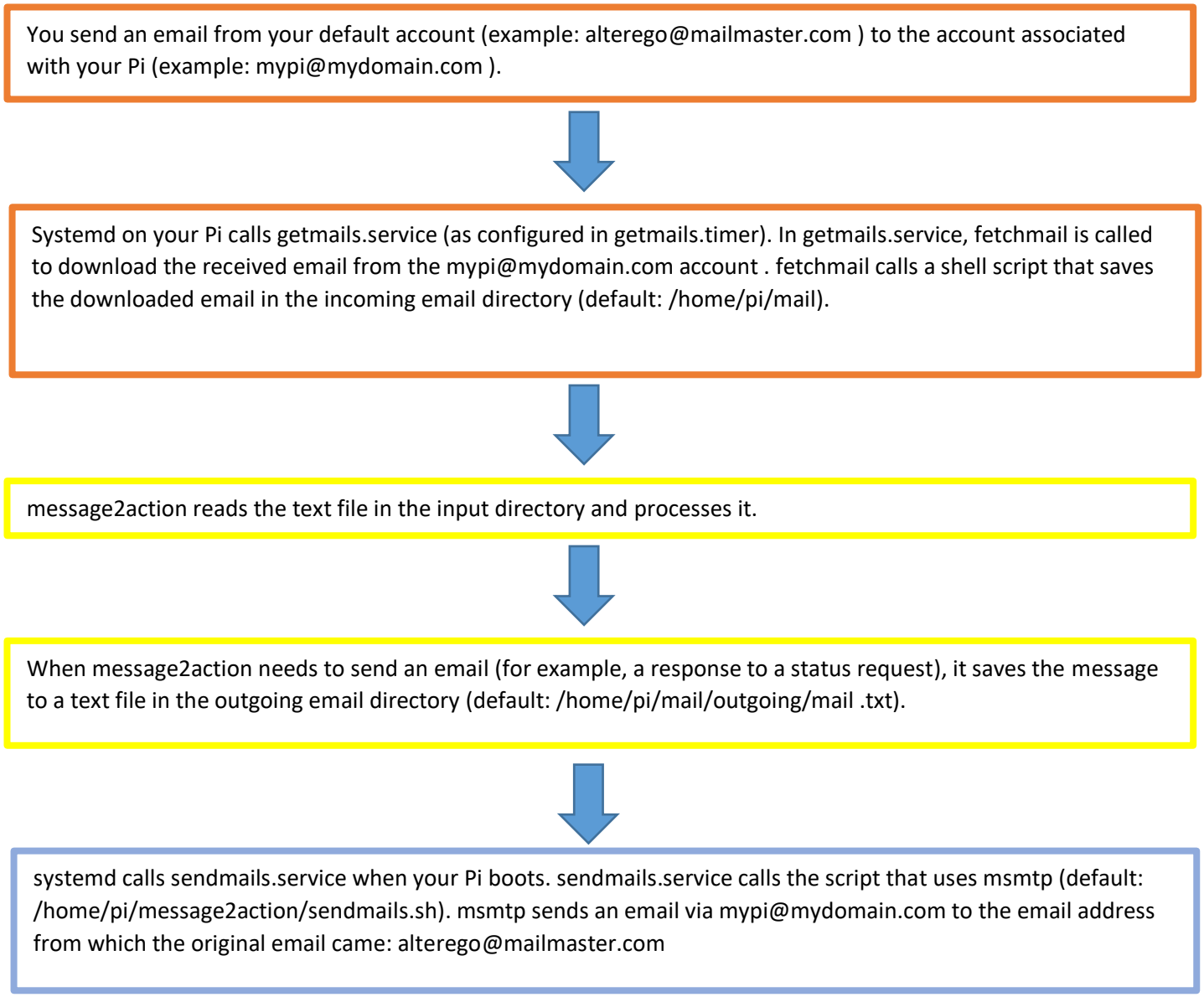
```
To: 4999

Your text to your cell phone
```

The files are stored by message2action with this naming convention: GSM1.<date>_<time>_<number of CPU ticks>. The <number of CPU ticks> helps make the name of the file unique.

13.2) Mail

This is the flow for processing email on your Pi with message2action using the optional fetchmail and msmtp software:



13.2.1) Naming convention and structure of the text files

Here is an example of a text file for an incoming email:

```
From: dummyuser@pathfinder-outlander.lit
Subject: status
```

The email address is used by message2action to verify that we have received a message from a valid email account. The text after Subject: contains the command you are sending to your Pi.

Here is an example of a text file containing an outgoing email:

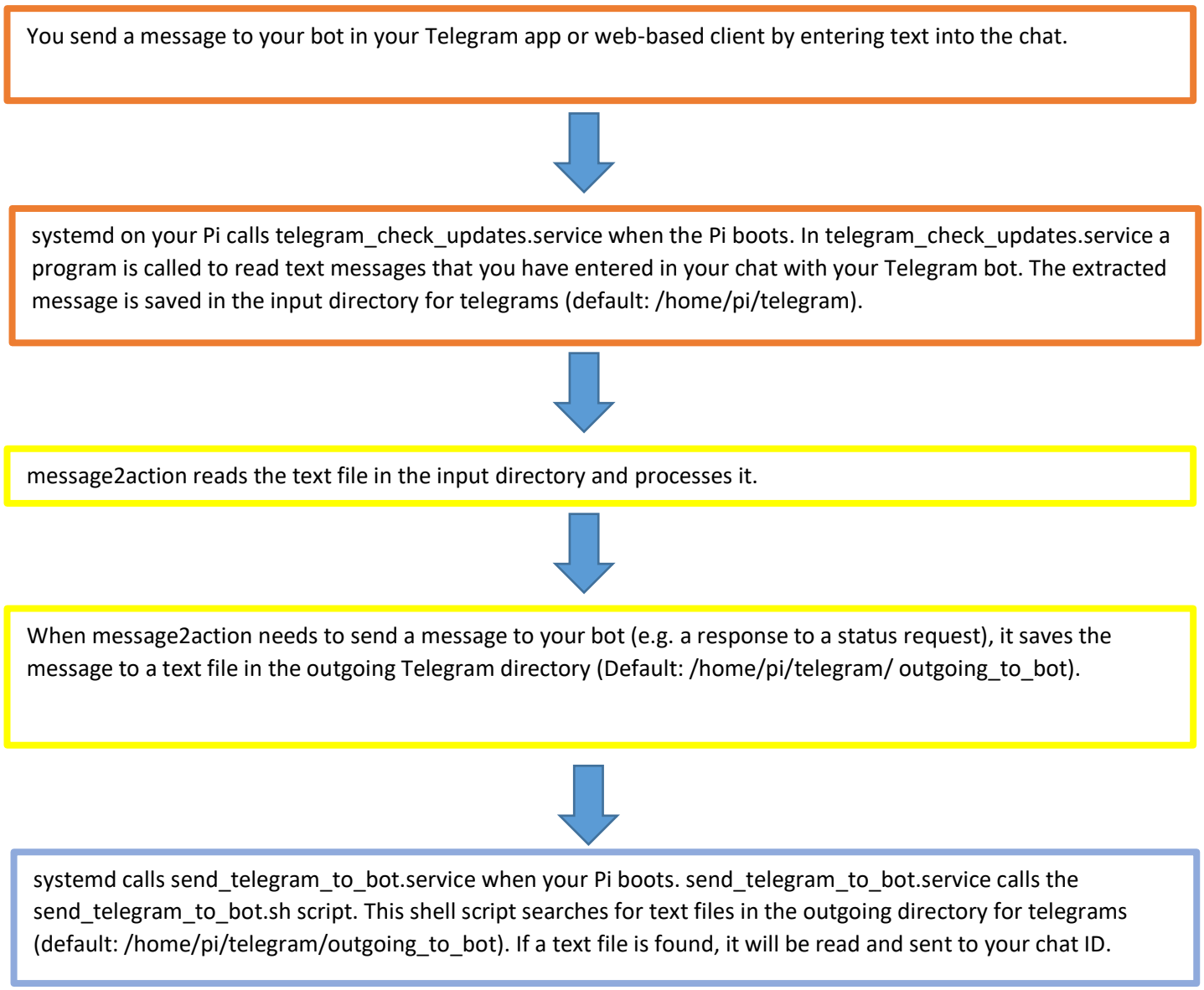
```
To:DUMMYUSER@DUMMY-DOMAIN.INT
Subject: Info: no further information for you

Message from message2action, see subject for details.
```

The files are saved with this naming convention: <date>_<time>_<number of CPU ticks>.mail. The <number of CPU ticks> helps make the name of the file unique.

13.3) Telegram

This is the flow for processing Telegram messages on your Pi with message2action using the optional Curl software:



13.3.1) Naming convention and structure of the text files

Here is an example of a text file containing a message from a chat with a bot:

```
Name: message_from_chat
Chat ID: 999999999
Message: status
```

The first line is currently always the same. The second line contains the chat ID with your bot. The last line contains the message you entered in your chat with your bot.
Here is an example of an outbound message sent to the chat ID using your bot:

```
Token: 12345:ffzfzzz55rrdd
Chat ID: 999999999
Message to bot: Info: # of processed telegrams: 1, SMS: 0, mails: 0, errors: 0,
warnings: 0
```

The first line stores your bot's token. The second line stores the chat ID with your bot. The third line stores the message that will be visible in your chat with the bot.

14) Releases of the program

Release 0.13 of its predecessor called “sms2action” was released in August 2016. Here are the changes made in the versions of message2action:

14.1) Release 0.27

- Processing of emails has started.
- The check_online_status_waittime parameter has been added to the ini file.
- If the program is running, you can end it again by pressing a button.
- Systemd services getmails.service, getmails.timer and sendmails.service have been deployed.
- Documentation in PDF format has been published.
- Commands to the Pi are converted to capital letters (example: “status” becomes “STATUS”).
- Various bugs have been fixed.

14.2) Release 0.28

- An error when reading the INI file has been fixed.
- Fixed a bug reading the line in an email containing the command (it was assumed that the new line was always at the end of the line).
- Global variables are now initialized with constant values defined with the define statement.
- The program call was changed in systemd-files.

14.3) Release 0.30

- The fullpath_systemd_shell_mail parameter is no longer needed as this function has been moved to systemd files.
- The fullpath_outgoing_mail parameter is no longer needed as a naming mechanism for filenames has been implemented.
- Fixed some bugs.
- Telegram message processing has been added.
- The naming of the files used to send messages has been changed.

14.4) Release 0.31

- Python pathlib library installation has been added in this documentation.
- A small bug in message2action has been fixed.

14.5) Release 0.32

For SMS, the way the command is searched has been changed. In some cases there is no entry for "IMEI:" in the file and I was sure that this was only the case with my stick. But that was wrong. Now the program simply looks for a blank line in the text file and uses this as a separator for the command.

14.6) Release 0.33

3 new commands have been implemented: message2action_help_command, message2action_questionmark_command and message2action_config_command. Now you can get help by sending “help” or “?” (both values are default). If you want to get the current configuration, you can use “config” as default. The sendmails.sh and send_telegram_to_bot.sh scripts had to be adjusted.

14.7) Release 0.34

- When message2action runs as a systemd service, it attempts to intercept keyboard input. However, this does not make sense and leads to the error message “Error: Terminal settings could not be determined!” and “Error: Terminal settings could not be set to new values!”. To handle this mode, a new command line parameter called “daemon” has been implemented. It can take the values y or n. If you specify daemon=y, the program will no longer attempt to intercept keyboard input.
- Deleting old files is now possible. In the ini file you can use the parameter`garbage_collection_days specify the number of days how old files in the processed directories are allowed to become.`

14.8) Release 0.35

When message2action receives commands such as "reboot" or "shutdown", it now attempts to send back an acknowledgment beforehand to signal that the command has been received and is being executed. This also applies to the commands CMD00 ... CMD09.

14.9) Release 0.36

2 errors in sending WOL to the system have been fixed. Initially it was suppressed from sending a confirmation of the status back to the sender after sending WOL. The second error caused the result of ping (used to check online status) to be misinterpreted and message2action ran in an infinite loop.

14.10) Release 0.39

Release 0.37 and 0.38 were not released. Fixed several bugs related to buffer usage and string comparison. The procedure for shutting down and restarting the Raspberry has been changed: sudo has been added before the command.

14.11) Release 0.40

The ssmtp program is no longer maintained for Raspbian OS “Buster”. So I had to find another program and switched to msmtp. The sendmails.sh file had to be adjusted.

14.12) Release 0.50

The source code was changed from C to C++. This enabled some new features:

- It is no longer necessary to use command line parameters to specify which input channel is used to receive messages. You specify this in the ini file. Now it is clear that a response is sent on the same channel on which it was received.
- Multithreading is used to read multiple inbound channels in parallel. This is helpful if an incoming channel fails. You can use an alternative channel, for example to restart the Pi.
- I replaced using Python to receive Telegram messages with a C++ program. This program is called telegram_check_updates.
- Parameters in the ini file for storing the process ID of an instance are no longer needed.

14.13) Release 1.10

The first version released on Github.

FAQ (Frequently asked questions)

#1: Q: Why are there two ways to stop message2action program?
A: First, the method was implemented with the “stopfile” parameter. This path is slightly slower than the “stopsignal” parameter. When you call the stopfile parameter, a message file is created and message2action reads this file. However, if you specified the number of seconds to wait in the next cycle in the ini file, you have to wait for the cycle to finish. This is realized by the sleep function in C.
If you call the program with the “stopsignal” parameter, the sleep function is interrupted immediately. This is necessary for implementing message2action as a service on your Pi because you want to stop all services very quickly when you shut down or restart your Pi.

#2: Q: What devices and software do I need to test message2action?
A: You don't need any SMS tools, a surf stick or anything like that. You can use and modify the sample files to check whether the program meets your requirements. You don't need an email program on your Pi either, as you can use sample files for this too. The same applies to Telegram.

#3: Q: In which languages can I contact the author?
A: In German and English.

#4: Why is there a getmails.timer but no sendmails.timer?
A: When I started developing the software, I couldn't write scripts in Bash. That has changed a little. I started with getmails.service and added a getmails.timer because I found it easy to let systemd take control of the scheduling. A few months later I created sendmails.service and was then able to write a sendmails.sh that contains the scheduler. I'm not entirely sure which way is better: handing off the scheduling job to systemd or making this call myself. So I left both methods to learn from them.

#5: Is there also support for the Telegram client (telegram-cli)?

A: No, not anymore. I found it very difficult to implement the Telegram CLI and I found it easier to use Telegram bots. I managed to get the Telegram CLI working, but it wasn't a satisfactory solution for me.

#6: Can I process more than one message type in parallel?
A: Yes. All you have to do is set the flags for inbound_sms_file_flag, inbound_mail_file_flag and inbound_telegram_file_flag to true in the ini file.

15) Contact the author of the program

If you have any questions or comments, you can reach me by email at: freeofcharge@t-online.de . Please provide this information to make it easier for us to assist you:

Type of information	source
Release of message2action	Simply enter in a terminal: `./message2action`
Log file from message2action	See the path where the log file is located (default: /home/pi/message2action): message2action.log
Name and version of the operating system (in the vast majority of cases this will be Raspberry OS)	Simply enter in a terminal: `cat /etc/os-release`
Kernel number and CPU name	Simply enter in a terminal: `uname -a`

16) Legal