Inhaltsverzeichnis

1) Einleitung

Was ist message2action?
Das Programm verarbeitet eingehende Nachrichten wie SMS, Emails, Telegram-Chat-Nachrichten und reagiert auf den Inhalt in den Nachrichten. Gelegentlich wird dies auch „Server-Steuerung“ genannt.
Das Programm ist für den Raspberry Pi (im folgenden nur „Pi“) - oder besser: Raspbian OS – in C++ entwickelt worden. Es wurde nie auf anderen Plattformen getestet.
In der Nachricht, die an den Pi gesendet wird, können Kommandos wie „stop“ oder „status“ stehen und man kann in einer Konfigurationsdatei (ini-Datei) festlegen, was passieren soll, wenn ein solches Kommando erkannt wird.
In der aktuellen Fassung nutzt message2action eine Dateischnittstelle. In anderen Worten: es werden in Verzeichnissen Dateien gelesen und dann werden diese Dateien ausgewertet. Es ist kein Online-Programm.
Der Auslöser, warum ich das Programm entwickelt habe, war der Wunsch, meinen PC zu starten aber ohne, dass der PC über das Internet erreichbar ist. Die Ursprungsidee war, eine SMS an den Pi zu senden und dieser sendet ein WOL (wake on LAN) an den PC. Mit message2action wird also vermieden, dass Dyndns oder Portforwarding (im Router) verwendet werden muss. Und der PC ist nach wie vor nicht über das Internet erreichbar.
Mit dieser Idee dachte ich, es wäre eine gute Idee, ein standardisiertes Interface auf einem Pi zu haben. Das Interface kann auch für andere Zwecke genutzt werden wie zum Beispiel Statusanfragen von Geräten im Heimnetz wie Temperatursensoren oder ähnlich.

2) Eigenschaften

Hier ist eine Liste der wichtigsten Eigenschaften:
-Es werden Textdateien gelesen (Mails, SMS und Telegram) und in den Textdateien nach Kommandos gesucht und dann ausgeführt. Den Erhalt dieser Textdateien müssen andere Programme übernehmen.

-Es werden Textdateien erzeugt (Mails, SMS und Telegram) wenn eine Antwort an den Absender zurückgesendet werden soll (zum Beispiel als Antwort auf eine Statusanfrage). Das Versenden der Antworten an den Empfänger müssen andere Programme übernehmen.
-In einer Logdatei werden Informationen, Warnungen und Fehler mitgeschrieben, damit man den Programmverlauf auch im Nachhinein nachvollziehen kann. Dies ist bei der Fehlersuche hilfreich.
-Die maximale Größe der Logdatei kann eingestellt werden.
-Die ein- und ausgegangenen Textdateien werden auf dem Pi gespeichert und es kann ein maximales Alter für die Dateien angegeben werden. Dateien, die älter sind, werden gelöscht.
-Maximal 10 Mailadressen, 10 Mobilnummern und 10 Chat-IDs für Telegram können als Kontaktpartner angegeben werden.
-Es werden Warnungs-Nachrichten versendet, wenn eine Nachricht von einem unbekannten Absender erkannt wird. Der unbekannte Absender erhält keine Information.
-Maximal 10 Kommandos (Skripte, Programme) können konfiguriert werden.
-Interne Kommandos sind vorhanden, um das Programm zu stoppen, den Pi neu zu starten oder auch herunterzufahren.

3) Anforderungen

Was wird benötigt, um message2action zu nutzen?

3.1) SMS

Man benötigt einen Surfstick, der per USB an den Pi angeschlossen werden kann. Für den Surfstick benötigt man noch eine SIM-Karte, um SMS zu empfangen und zu versenden.
Aktuell benutze ich "SMS Server Tools 3" (http://smstools3.kekekasvi.com) um SMS mit dem Surfstick zu versenden und zu empfangen. Die Software kann mit diesem Kommando installiert werden:

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

Ich habe einen Vodafone Surfstick namens K3-772Z mit einer Prepaid SIM-Karte von T-Mobile am Pi angeschlossen. Das ist zwar ein UMTS-Stick. Aber um SMSen zu empfangen oder zu versenden, wird das UMTS-Netz nicht genutzt sondern das GSM-Netz. Daher kann ich den UMTS-Stick trotz abgeschalteten UMTS-Netz (Mitte 2021) weiternutzen.
Um den Surfstick von dem Daten-Modus in den Modem-Modus umzuschalten, kann man das Programm USB Modeswitch verwenden. (http://www.draisberghof.de/usb_modeswitch). Man sollte sich das Data-Package auch installieren. Beides kann mit diesem Kommando installiert werden:

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

Die erwähnten "SMS Server Tools" speichern eingegangene SMS als Textdateien ab. Das Default-Verzeichnis lautet `/var/spool/sms/incoming`.
message2action liest alle Dateien in diesem Verzeichnis und sucht nach Kommandos in den Dateien. Nach dem Auswerten werden die bearbeiteten Dateien in ein gesondertes Verzeichnis geschoben (Processed-Dateien).
Wenn message2action eine SMS versenden will, wird die Datei in dem Ausgangsverzeichnis von SMS Server Tools abgelegt (Default: `/var/spool/sms/outgoing`).
Beachte: Die erwähnten Verzeichnisse werden in der Ini-Datei gepflegt und können auch anders lauten.

3.2) Mails

Man braucht einen Account bei einem Mailprovider. Dieser Account (genauer: die zugehörige Mailadresse) ist der Kontakt zum Pi, im folgenden Pi-Account. Um Mails an den Pi zu senden, ist ein weiterer Account erforderlich, im folgenden Sender-Account. Um also Mails an den Pi zu senden, sendet man eine Mail von dem Sender-Account an den Pi-Account. Und wenn der Pi eine Mail an den Absender schicken will, so sendet der Pi-Account an den Sender-Account.
Um Mails vom Pi-Account auszulesen, nutze ich `fetchmail`. Um Mails vom Pi aus an den Sender-Account zu schicken, nutze ich `msmtp`.
`fetchmail` speichert alle eingegangenen Mails als Textdateien. Man kann das Verzeichnis angeben, wo die Dateien abgelegt werden sollen.
message2action liest alle Dateien in diesem Verzeichnis. Nach dem Verarbeiten der Dateien werden sie in ein Verarbeitet-Verzeichnis verschoben. Zur Installation von `fetchmail` kann dies aufgerufen werden:

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

Damit message2action Mails versenden kann, wird eine Textdatei erzeugt, die als Input für ein Skript verwendet wird. Dieses Skript wird von `sendmails.service` aufgerufen (diese systemd-Datei wird von mir bereitgestellt). Das Skript ruft `msmtp` auf, um eine Mail von dem Pi-Account an den Sender-Account zu senden. Um `msmtp` zu installieren, kann dies aufgerufen werden:

`sudo apt-get install msmtp`
Sowohl `fetchmail` als auch `msmtp` müssen vom Anwender noch eingerichtet werden.

3.3) Telegram

Man braucht entweder eine App (Smartphone) oder einen Webbrowser, um sich in Telegram einzuloggen. Zuerst wird ein Account und ein Bot eingerichtet. Für den Bot braucht man den Token, der mit dem Bot verknüpft ist. Der Token sieht in etwa so aus: `ABGxGlaeQTbUR_fWZFG2RT8u6MhABCxLwmw`
Es wird noch die Chat-ID benötigt, mit der kommuniziert werden soll. Das ist eine neunstellige Nummer. Bitte suchen Sie im Internet nach Anleitungen, wie man die Chat-ID und den Token herausfindet. Die Chat-ID und der Token müssen in der Ini-Datei von message2action angegeben werden. Das ist die Datei `message2action.ini` (liegt im gleichen Verzeichnis wie das ausführbare Programm).
Das Versenden von Telegram-Nachrichten geschieht über `curl`. Falls `curl` nicht installiert ist, kann es mit diesem Aufruf installiert werden:
`sudo apt-get install curl`
Um Telegram-Nachrichten, die message2action erzeugt hat, zu versenden, wird das Skript `send_telegram_to_bot.sh` (wird von mir bereitgestellt) versendet. Dieses Skript durchsucht das Verzeichnis für ausgehende Nachrichten und sendet jede einzelne per `curl` an den Bot.
Um Telegram-Nachrichten auf dem Pi zu empfangen und auf dem Pi als Textdatei abzulegen, wird das Programm `telegram_check_updates` verwendet. Auch dieses Programm wird von mir bereitgestellt. Um das Programm beim Systemstart mit zu starten, kann die systemd-Unit `telegram_check_updates.service` genutzt werden (wird von mir bereitgestellt).

4) Kommandozeilenparameter für message2action

Wenn man message2action ohne Parameter aufruft, so erscheint eine kleine Hilfe, die die verfügbaren Kommandos auflistet.
Dies sind die Parameter für message2action:

`?`: Eine kleine Hilfe wird angezeigt.

`start`: Startet das Programm im Standard-Modus: es durchsucht Verzeichnisse (wie in der ini-Datei angegeben) nach Textnachrichten und wertet diese dann aus. Um das Programm wieder anzuhalten, kann einfach eine sogenannte lesbare Taste auf der Tastatur gedrückt werden. Lesbare Taten sind zum Beispiel Escape, Leertaste, alle Zahlen und Buchstaben. Die Hochstell (Shift) und die Steuerung (Strg bzw. Ctrl)-Taste zählen nicht dazu. *Zusätzlich* stehen diese 3 Möglichkeiten zur Verfügung:

1) Erzeugen einer Stop-Datei (siehe Parameter `stop`)
2) Empfang einer Nachricht mit dem Kommando zum Stoppen
3) In einem anderen Terminal wird message2action mit Parameter `stop` aufgerufen

`daemon`: Startet das Programm im Daemon-Modus: es durchsucht Verzeichnisse (wie in der ini-Datei angegeben) nach Textnachrichten und wertet diese dann aus. Dieser Modus gibt nur sehr wenige Ausgaben auf dem Bildschirm aus und ist deswegen geeignet, mit einer systemd-Unit gestartet zu werden. Im Daemon-Modus kann das Programm nicht durch Drücken einer Taste gestoppt werden. Es kann *nur* noch auf 3 Wege gestoppt werden:

4) Erzeugen einer Stop-Datei (siehe Parameter `stop`)
5) Empfang einer Nachricht mit dem Kommando zum Stoppen
6) In einem anderen Terminal wird message2action mit Parameter `stop` aufgerufen

`stop`: Es wird eine sogenannte stop-Nachricht erzeugt, die von einer anderen Instanz des Programms ausgewertet werden kann und damit das letztgenannte Programm anhalten lässt.

`config`: Mit diesem Parameter wird die aktuelle Konfiguration auf dem Bildschirm ausgegeben, wie sie in der ini-Datei gespeichert ist. Diese Funktion ist nützlich, um zu prüfen, ob Änderungen an der ini-Datei auch korrekt vom Programm verstanden worden sind. Die ini-Datei wird nur zum Start des Programms gelesen, danach nicht mehr. Das bedeutet, dass nach nach jeder Änderung an der ini-Datei das Programm zu stoppen und neu zu starten ist.

5) Beschreibung der ini-Datei

Die Konfiguration des Programmes wird einer sogenannten ini-Datei abgelegt und kann mit einem Texteditor bearbeitet werden. Der Name der ini-Datei lautet `message2action.ini`. Diese Datei liegt im gleichen Verzeichnis wie das ausführbare Programm.
In der ini-Datei können Werte zu Parametern angegeben werden, in welchem Verzeichnis zum Beispiel verarbeitete Dateien abgelegt werden sollen. Jeder Buchstabe hinter dem Zeichen ‚#' (ohne Anführungszeichen) wird als Kommentar gewertet. In der ausgelieferten ini-Datei ist zu sehen, wie Kommentare aussehen.
Direkt hinter einem Parameter muss das ‚=' (ohne Anführungszeichen) stehen. Direkt nach dem ‚=' muss der Wert angegeben werden. Siehe dazu die folgenden Beispiele:
CMD03='/home/pi/demo' ist richtig. Aber
CMD03 ='/home/pi/demo' ist falsch und
CMD03= '/home/pi/demo' ist ebenso falsch.

5.1) Generelle Konfiguration des Programms

`capitalize_values`, Default: true; mögliche Werte: true, false.
Dieser Parameter legt fest, ob alle eingehenden Kommandos nach Großbuchstaben gewandelt werden sollen (true) oder nicht (false). Dieser Parameter ist hilfreich, um beim Tippen von Kommandos Fehler wie „Stop", „stoP" anstelle von „stop" zu vermeiden.

`file_logfile`, Default: /home/pi/message2action/message2action.log; mögliche Werte: vollständiger Pfad zu einer Log-Datei (inklusive Name der Datei).
In dieser Datei werden Log-Informationen mitgeschrieben, während das Programm läuft. Die Informationen werden mit einem Zeitstempel und Tags wie „Info", „Warning" und „Error" versehen.
"Info": Es handelt sich lediglich um eine Informatin wie zum Beispiel das gefundene Kommando in einer SMS oder auch der Status des Programms.
"Warning": Irgendetwas läuft nicht einwandfrei und der Nutzer sollte dazu informiert werden. Zum Beispiel wurde eine unbekannte Telefonnummer als Absender in einer SMS erkannt.
"Error": Es ist ein Fehler aufgetreten, der die Nutzung des Programms einschränken kann. Zum Beispiel konnte die Log-Datei nicht zum Scrhreiben geöffnet werden oder das Verschieben der verarbeiteten Datei scheitert an den nötigen Rechten.

`max_size_logfile`, Default: 500; mögliche Werte: jede ganze Zahl.
Dieser Wert legt die maximale Größe der Log-Datei fest und zwar in kBytes (1 kByte = 1024 Bytes). Wenn diese Größe erreicht ist, wird die Log-Datei in 2 Hälften geteilt und der ältere Teil wird gelöscht. So bleibt stets der jüngste Teil des Inhaltes übrig.
Die Praxis hat gezeigt, dass die Logdatei mit 500 kByte etwa xyz Zeilen enthält.

`garbage_collection_days`, Default: 30; mögliche Werte: jede ganze Zahl.
Mit diesem Parameter wird das maximale Alter ausgedrückt in ganzen Tagen für verarbeitete Dateien festgelegt. Das Programm durchsucht die Verzeichnisse, in denen verarbeitete Dateien gespeichert sind. Anschließend werden alle Dateien gelöscht, die älter als die eingestellte Anzahl von Tagen ist.
Diese Funktion schützt davor, dass der Platz in den Filesystemen erschöpft wird, weil beliebig alte Daten dort immer noch liegen.

5.2) Versand und Empfang von SMS

`receivingfrom_cell_phone00` bis `receivingfrom_cell_phone09`, Default: 4999; mögliche Werte: Jede internationale Telefonnummer ohne führendes ‚+'-Zeichen (ohne Anführungszeichen).
Mit den 10 Telefonnummern wird festgelegt, von wem SMS verarbeitet werden dürfen und an wen Antworten (zum Beispiel der Status des Programms) zurückgesendet werden dürfen. Vielleicht sollen Familienmitgliedern oder Mitgliedern eines Teams erlaubt werden, dass das Programm SMS von diesen verarbeiten darf. Jede Telefonnummer hat das gleiche Recht, es gibt kein Rollenmodell zu den Telefonnummern. Diese Funktion schützt davor, dass Unberechtigte Kommandos ausführen können.

Falls eine Telefonnummer erkannt wurde, die nicht in der ini-Datei gelistet ist, so wird das Kommando ignoriert und es wird an alle Telefonnummern eine Warnung versendet. Diese Warnung sieht wie folgt aus:
"Warning: invalid sender-phonenumber in incoming SMS: %s!". Die Variable %s enthält die erkannte Telefonnummer von der die SMS gekommen ist.

`path_incoming_SMS`, Default: /var/spool/sms/incoming/; mögliche Werte: vollständiger Pfad, in dem eingehende SMS zu finden sind.
Dieser Pfad gibt an, wo eingehende SMS als Textdatei zu finden sind. In diesem Pfad sucht das Programm nach Textdateien.

`path_outgoing_SMS`, Default: /var/spool/sms/outgoing/; mögliche Werte: vollständiger Pfad, in dem ausgehende SMS abgelegt werden sollen.
Das ist der Pfad, in dem das Programm Textdateien ablegt, die per SMS zu versenden sind.

`path_processed_SMS`, Default: /var/local/message2action/processedsms/; mögliche Werte: vollständiger Pfad, in dem verarbeitete SMS zu finden sind.
In dieses Verzeichnis verschiebt das Programm die eingegangenen Dateien, nach dem das Programm den Inhalt verarbeitet hat (egal mit welchem Status). Damit wird das Eingangsverzeichnis stets geleert. So kann geprüft werden, ob das Programm die eingehenden SMS verarbeitet.

`wait_time_next_sms`, Default: 2; mögliche Werte: alle ganzen Zahlen größer 0.
Der Wert gibt die Anzahl Sekunden an, die das Programm warten soll, nachdem es das Eingangsverzeichnis nach SMS-Nachrichten im Textformat durchsucht hat und die letzte Datei verarbeitet hat. So kann also eine Wartezeit bis zum nächsten Suchen eingestellt werden.
Beachte: Das Programm zum Erzeugen der SMS als Text-Datei hat auch eine Wartezeit und diese beiden Wartezeiten sollten aufeinander abgestimmt sein.

`inbound_sms_file_flag`, Default: true; mögliche Werte: true, false.
Dieses Flag legt fest, ob eingehende SMS verarbeitet werden sollen (true) oder nicht (false).

5.3) Versand und Empfang von Mails

`mail_address_00` bis `mail_address_09`, Default: dummyuser@pathfinder-outlander.lit; mögliche Werte: jede Mailadresse.
Es können maximal 10 Mailadressen angegeben werden, von denen Text-Nachrichten zum Verarbeiten akzeptiert werden. Dieses Vorgehen ist das gleiche wie bei den Telefonnummern beim Verarbeiten von SMS. Wenn eine Mailadresse als Absender erkannt wurde, die hier nicht angegeben ist, so wird an alle angegebenen Mailadresse eine Warnung versendet. Diese Warnung sieht wie folgt aus:
"Warning: invalid mail address in incoming mail: %s!". Die Variable %s enthält die erkannte Mailadresse. Mit dieser Funktion wird erreicht, dass nur bekannte Mailabsender Kommandos ausführen können.
Tipp: Ich empfehle auch die Mailadresse anzugeben, von der aus Ihr Mailprovider Mails mit „unbekannte Mailadresse“ versendet. Zum Beispiel lautet diese Adresse für T-Online der Deutschen Telekom `mailer-daemon@t-online.de`.

`path_incoming_mail`, Default:/home/pi/mail/; mögliche Werte: vollständiger Pfad, in dem **ein**gehende Mails als Textdateien zu finden sind.
Das ist das Eingangsverzeichnis, in dem das Programm nach Mails sucht.

`path_outgoing_mail`, Default:/home/pi/mail/outgoing/; mögliche Werte: vollständiger Pfad, in dem **aus**gehende Mails als Textdateien zu finden sind.

`//fullpath_mailfile_PID`, default: /home/pi/message2action/mailpid
//Same meaning as for SMS.

`wait_time_next_mail`, Default: 30; mögliche Werte: alle ganzen Zahlen
Der Wert gibt die Anzahl Sekunden an, die das Programm warten soll, nachdem es das Eingangsverzeichnis nach Mail-Nachrichten im Textformat durchsucht hat und die letzte Datei verarbeitet hat. So kann also eine Wartezeit bis zum nächsten Suchen eingestellt werden.
Beachte: Das Programm zum Erzeugen der Mail als Text-Datei hat auch eine Wartezeit und diese beiden Wartezeiten sollten aufeinander abgestimmt sein.

`inbound_mail_file_flag`, Default: true; mögliche Werte: true oder false.
Dieses Flag legt fest, ob eingehende Mails verarbeitet werden sollen (true) oder nicht (false).

5.4) Versand und Empfang von Telegram-Nachrichten

`telegram_bot_token`, Default: my_telegram_bot_token; mögliche Werte: der Token für Telegram.
Dieser Token bezieht sich auf den Bot in Telegram. Aktuell ist nur 1 Bot für das Programm konfigurierbar. Aber es können bis zu 10 Chats angegeben werden. Der Token wird wird mit der Chat-ID für das Abholen und Versenden von Chat-Inhalten benötigt.

`telegram_chat_id_00` bis `telegram_chat_id_09`, Default: 999999999; mögliche Werte: jede zulässige Chat-ID.
Das sind maximal 10 Chat-IDs, zu denen der Bot eingeladen werden kann. Sofern beim Abholen von Chat-Nachrichten eine Chat-ID erkannt wurde, die nicht in der ini-Datei angegeben wurde, so wird das Kommando ignoriert und es wird eine Warnung an alle Chats versendet. Die Warnung sieht genauso aus wie für SMS und Mails.
Mit dieser Feature wird verhindert, dass jemand Fremdes Kommandos auf dem Pi ausführt.

`path_incoming_telegram`, Default:/home/pi/telegram/; mögliche Werte: vollständiger Pfad, in dem **ein**gehende Telegram-Nachrichten als Textdateien zu finden sind.

`path_outgoing_telegram`, Default:/home/pi/telegram/outgoing_to_bot/; mögliche Werte: vollständiger Pfad, in dem **aus**gehende Telegram-Nachrichten als Textdateien zu finden sind.

`path_processed_telegram`, Default:/home/pi/message2action/telegramprocessed/; mögliche Werte: vollständiger Pfad, in dem verarbeitete Telegram-Nachrichten als Textdateien zu finden sind.

`wait_time_next_telegram`, Default: 2; mögliche Werte: alle ganzen Zahlen größer 0
Der Wert gibt die Anzahl Sekunden an, die das Programm warten soll, nachdem es das Eingangsverzeichnis nach Telegram-Nachrichten im Textformat durchsucht hat und die letzte Datei verarbeitet hat. So kann also eine Wartezeit bis zum nächsten Suchen eingestellt werden.
Beachte: Das Programm zum Erzeugen der Telegram-Nachricht als Text-Datei hat auch eine Wartezeit und diese beiden Wartezeiten sollten aufeinander abgestimmt sein.

`inbound_telegram_file_flag`, Default: true; mögliche Werte: true oder false.
Dieses Flag legt fest, ob eingehende Telegram-Nachrichten verarbeitet werden sollen (true) oder nicht (false).

5.5) Kommandos in Nachrichten an den Pi

Zunächst einmal wird zwischen sogenannten „eingebauten“ und „externen“ Kommandos unterschieden, die an den Pi gesendet werden können.
Eingebaute Kommandos sind solche, die in das Programm eingebaut sind. Beispiele dafür sind `stop`, `shutdown` und `WOL` (Wakeup on LAN). In der ini-Datei können Schlüsselwörter festgelegt werden, die für das Ausführen der eingebauten Kommandos verwendet werden sollen.
Externe Kommandos sind solche, die Programme oder auch Skripte auf dem Pi ausführen lassen. Der vollständige Pfad zu einem solchen Programm ist ein Wert zu dem externen Kommando.

5.6) "Build-in" commands

Als mögliche Werte für die eingebauten Kommandos gilt: 1 Wort beliebiger Länge ist erlaubt.

`message2action_stop_command`, Default: stop
Wenn man das Kommando „stop“ (ohne Anführungszeichen) an den Pi sendet, so beendet sich das Programm. Sofern einem das Wort nicht gefällt, könnte man es auch auf „anhalten“ setzen.
Sofern man irgendwelche Sicherheitsbedenken festgestellt hat, ist es nützlich, die Verarbeitung von weiteren Nachrichten anhalten zu lassen.

`shutdown_pi_command`, Default: shutdown
Wenn man das Kommando "shutdown" (ohne Anführungszeichen) an den Pi sendet, so wird der gesamte Pi heruntergefahren.
Der Zweck für dieses Program ist der gleiche für das `message2action_stop_command`. Man kann bei festgestellten Bedenken den gesamten Pi herunterfahren.

`reboot_pi_command`, Default: reboot
Dieses Kommando beendet das Programm und startet den gesamten Pi neu.

`status_pi_command`, Default: status
Mit diesem Kommando kann der Status des Programms abgefragt werden. Der Status enthält eine Meldung zu jedem Kanal die Anzahl der verarbeiteten Nachrichten, der Warnungen und Fehler seit dem Start des Programms.

`message2action_help_command`, Default: help
Mit diesem Kommando wird eine kleiner Hilfstext zurückgesendet, der die möglichen Kommandos auflistet.

`message2action_questionmark_command`, Default: ?
Mit dem Zeichen für das Fragezeichen wird das gleiche ausgelöst wie mit dem Kommando für die Hilfe. Vielleicht kann man sich das Fragezeichen leichter merken und es ist auch kürzer.

`message2action_config_command`, Default: config
Mit diesem Kommando wird die aktuelle Konfiguration des Programms ausgegeben. Das Ergebnis ist das gleiche, als würde man auf der Kommandozeile „config“ (ohne Anführungszeichen) als Parameter eingeben.

`WOL00` bis `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

Die Werte hinter den Parametern `WOL00` bis `WOL09` sind MAC-Adressen von Netzwerkadaptern. Die MAC-Adresse wird verwendet, um per UDP-Broadcast das sogenannte Magic-Paket (Wake on LAN) an ein System zu senden. Der Empfang des Magic-Paket löst bei dem empfangenden Device zum Beispiel ein Aufwachen aus dem Ruhemodus oder auch einen Kaltstart aus. Es können maximal 10 MAC-Adressen eingestellt werden.
Hinter der MAC-Adresse kann durch ein ‚;‘ (ohne Anführungszeichen) getrennt entweder ein TCP-IP-Hostname oder eine IPV4-IP-Adresse vergeben werden.
Ein Hostname wird angegeben durch:
host=myhostname
Eine IP-Adresse wird angegeben durch:
ip=123.123.123.123
Der Hostname oder die IP-Adresse kann verwendet werden, um nach dem Senden des WOL-Kommandos den Online-Status zu prüfen. Dies geschieht durch den Aufruf von ping.

`online_status_command_suffix`, Default: stat

Dieses Suffix kann an das WOL-Kommando (`WOL00` bis `WOL09`) angehängt werden, um den Online-Status eines Devices abzufragen.
Beispiel: Man möchte wissen, ob das Device, das zu der Adresse von WOL04 gehört, online ist. Dann sendet man `WOL04stat` an das Programm. Das Programm prüft zu dem Device den Online-Status per ping-Befehl und sendet die Antwort zurück.

`check_online_status_waittime`, Default: 30
Die Zahl gibt die Wartezeit in Sekunden an, die zwischen dem Senden eines WOL-Kommandos und der Abfrage es Online-Staus liegen soll. Manche Devices wie zum Beispiel ein PC brauchen mehrere Sekunden nach dem Aufwachen oder Kaltstart, bis der Netzwerk-Stack zur Verfügung steht und reagieren kann.
Mit dem Wert 0 wird diese Funktion ausgeschaltet und man bekommt keine Antwort.

`CMD00` bis `CMD09`, Default für CMD00: date > /tmp/test.log.
Mit diesen Parametern kann man bis zu 10 frei definierte Kommandos, Programme oder Skripte ausführen lassen, in dem man `CMD00` bis `CMD09` an den Pi sendet. Das kann nützlich sein, um zum Beispiel andere Programme wie einen Web-Server oder eine Datenbank neu zu starten. Man kann also auf diesem Wege die Funktion des Programms erweitern.

6) Kommandos in Nachrichten

Die Art und Weise, wie man Kommandos an den Pi sendet, hängt von dem Typ der Nachrichten ab. Im folgenden werden die Typen vorgestellt.

6.1) SMS

Wenn man eine SMS an den Pi sendet, schreibt man einfach nur das Kommando (zum Beispiel „status“ {ohne Anführungszeichen} in die SMS.

6.2) Mail

Das Kommando kommt nur in den Betreff und nicht in den Haupttext der Mail. Der Haupttext kann komplett leer gelassen werden, da dieser nicht ausgewertet wird.

6.3) Telegram

Das Kommando wird in den Chat mit dem Bot eingegeben. Hier sieht man ein Beispiel für 2 Kommandos:

Zuerst wurde `status` eingegeben und hat als Antwort den Status des Programms erhalten. Anschließend wurde das Wake on LAN für das Device 00 eingegben und man erhält den Online-Status zu dem Device zurück.

7) Beispieldateien

In dem Verzeichnis `samples` habe ich Beispieldateien für eingehende Nachrichten abgelegt, die man zum Testen nutzen kann. Mit diesen Beispieldateien muss man kein SMS-Gerät installiert haben, keine Mailprogramme installiert haben und auch nicht Telegram mit Chat-Bot konfiguriert haben. Stattdessen kann man diese Beispieldateien einfach in das Eingangsverzeichnis kopieren und sehen, was passiert.

7.1) SMS

Alle Beispieldateien für SMS heißen `GSM1.*`.

Beispiel: Man will ein Programm starten, das auf dem Pi unter `/home/pi/testdirectory/testprogram.sh` liegt und mit den Parametern `invoice tell` aufgerufen werden soll. Das Programm soll mit dem Kommando `CMD03` ausgelöst werden. Dies ist in der ini-Datei anzupassen:
`CMD03=/home/pi/testdirectory/testprogram.sh invoice tell`
Danach passt man die Beispieldatei an, so dass das Kommando `CMD03` dort zu finden ist. Dazu kann man die Datei GSM1.CMD03 nutzen. In der ersten Zeile muss eine Telefonnummer angegeben sein, die in der ini-Datei gepflegt ist. In die letzte Zeile wird das Kommando `CMD03` eingetragen. Als letzten Schritt wird diese Datei in das Verzeichnis für eingehende SMS kopiert (Default: `/var/spool/sms/incoming`).
Beachte: Zu den Werden der Kommandos CMD00 bis CMD09 finden keine Plausibibilitätsprüfungen statt.

7.2) Mail

Alle Beispieldateien für Mails heißen `mail.*`.
Beispiel: Will man das gleiche Kommando wie aus dem Beispiel für SMS ausführen lassen, so muss man eine Mail an die Mailadresse senden, die von dem Abholprogramm für Mails (Default: fetchmail) ausgelesen wird. Wenn das Abholprogramm die Mail abgeholt hat, bekommt man eine Textdatei. In dem Kapitel zu fetchmail ist beschrieben, wie man das Programm einrichten kann.
In dem Verzeichnis `samples` sind mehrere Beispieldateien zu finden, die per fetchmail empfangen worden sind. Diese Dateien kann man anpassen und dann in das Verzeichnis kopieren, in dem eingehende Mails für das Programm zu finden sind (Default: `mails` im Home-Verzeichnis).

7.3) Telegram

Alle Beispieldateien heißen `telegram.*`.
Das Vorgehen ist das gleiche wie für SMS und Mails. Die angepasste Datei wird in das Verzeichnis für eingehende Telegram-Nachrichten kopiert (Default: `telegram/outgoing_to_bot` im Home-Verzeichnis).

8) Installation und Konfiguration auf dem Pi

Auf meinem Pi habe ich das Programm in meinem Home-Verzeichnis unter `message2action` installiert. Alle folgenden Anweisungen und Befehle beziehen sich auf dieses Verzeichnis.
Im eigenen Homeverzeichnis führt man dieses Kommando aus, um das Repository herunterzuladen:
`git clone https://github.com/Sunblogger/message2action`

Nun erscheinen im Verzeichnis `message2action` mehrere Dateien:
`message2action`: das ausführbare Programm
`message2action.ini`: ini-Datei, die die Konfiguration des Programms enthält. Diese Datei muss individuell angepasst werden.
`message2action_de.pdf, message2action_en.pdf`: deutsche und englische Dokumentation.
`sendmails.sh`: Dieses Skript wird vom Systemd-Service `sendmails.service` aufgerufen, um Mails zu versenden, die von message2action erzeugt worden sind.

Diese Verzeichnisse werden angelegt:
`fetchmail`: Konfiguration von fetchmail, die vom Benutzer vorgenommen werden muss.
`msmtp`: Konfiguration von msmtp, die vom Benutzer vorgenommen werden muss.
`telegram`: Verzeichnis zum Speichern von eingehenden Telegram-Nachrichten.
`mailprocessed`: Verzeichnis zum Speichern von Mails, die von message2action verarbeitet wurden.
`smsprocessed`: Verzeichnis zum Speichern von Mails, die von message2action verarbeitet wurden.
`telegramprocessed`: Verzeichnis zum Speichern von Telegram-Nachrichten, die von message2action verarbeitet wurden.
`samples`: Beispieldateien für SMS, Mails und Telegram-Nachrichten. Siehe Kapitel Beispieldateien. Diese Dateien können zum Testen verwendet werden.
`systemd`: Dateien zum Einrichten von Systemd-Services unter anderem für den automatischen Start und Stop von message2action und anderen Hilfsprogrammen. Für Detail siehe Kapitel systemd.
In dem Verzeichnis liegt eine Installationsdatei namens installservicetimer.sh. Mit diesem Skript können die Systemd-Services installiert werden.

8.1) Quellcode

`message2action.cpp`: Quellcode in C++ für das Programm

8.2) SMS-Tools

Sofern man die SMS-Tools verwendet, muss die Datei `/etc/smsd.conf` angepasst werden. Das Device zum Senden und Empfangen von SMS ist einzurichten. Eingehende SMS müssen erlaubt werden mit `incoming = yes`.

8.3) Mail, fetchmail

Sofern man fetchmail zum Abholen von Mails nutzt, gilt das folgende:
Im Verzeichnis `fetchmail`:
Die Datei `fetchmailparser.sh` benötigt keine Anpassungen.
`fetchmailrc`: Hier muss der Mailserver, der Account plus Passwort und der ssl-Fingerprint eingetragen werden.

8.4) Mail, msmtp

Sofern man msmtp zum Versenden von Mails nutzt, muss man im Verzeichnis msmtp die Datei `.msmtprc` anpassen: Der Mailaccount plus Passwort und weiteres ist einzugeben.

8.5) Telegram

Im Verzeichnis `telegram`:
`send_telegram_to_bot.sh`: Keine Anpassung erforderlich.
`telegramparser.sh`: Keine Anpassung erforderlich.

Es wird ein Programm namens `telegram_check_updates` ausgeliefert, um aus einem Telegram-Chat Textnachrichten zu extrahieren und als Textdatei zu speichern. Wenn das Programm ohne Parameter aufgerufen wird, erhält man eine kleine Hilfe. Die Datei `telegram_check_updates.txt` enthält weitere Erklärungen.

9) systemd-Services

Im Verzeichnis `systemd` befinden sich Service und Timer-Dateien, die optional ausgeliefert werden und nicht zwingend für den Betrieb benötigt werden. In den folgenden Abschnitten werden die Dateien erklärt. Um die Service-Dateien zu installieren, kann das Skript `installservicetimer.sh` benutzt werden. Das Skript kopiert die Service-Dateien in das Verzeichnis `/lib/systemd/system`. Nach dem Kopieren wird jede Servicedatei mit `systemctl` registriert.

9.1) message2action.service

`message2action.service` startet das Programm beim Start des Betriebssystems und stoppt das Programm beim Herunterfahren des Betriebssystems.
Um die Lauffähigkeit der Servicedatei zu überprüfen, kann dies eingegeben werden:
`sudo systemctl status message2action`
Wenn das Programm als Service gestartet werden soll, kann dies eingegeben werden:
`sudo systemctl start message2action`
Um das Programm als Service anzuhalten, kann diese eingegeben werden:
`sudo systemctl stop message2action`

9.2) getmails.service und getmails.timer

`getmails.service` ruft `fetchmail` auf, um Mails abzurufen und speichert die Mails als Textdateien im Eingangsverzeichnis für Mails. Im Verzeichnis `fetchmail` liegt eine Konfigurationsdatei für `fetchmail` und ein Parser-Skript, das in der Konfigurationsdatei aufgerufen wird. Das Parser-Skript wird benutzt, um zum einen nur den Absender und den Betreff der Mail zu erhalten (der Haupttext der Mail wird verworfen). Außerdem sorgt es dafür, dass die Mails unter einem eindeutigen Dateinamen abgespeichert werden.
`getmails.timer` wird benutzt, um `getmails.service` regelmäßig aufzurufen. Per Default geschieht das alle 5 Minuten. Beachte: Der Mailprovider reagiert unter Umständen unerfreut, wenn zu oft Mails abruft.

9.3) sendmails.service

`sendmails.service` ruft ein Skript namens `sendmails.sh` mit dem Parameter `start` auf um in einer Schleife vorliegende Mails zu versenden. `sendmails.service` kann dies beenden, in dem das Skript `sendmails.sh` mit dem Parameter `stop` aufgerufen wird (das geht auch manuell). `sendmails.sh` wird ebenso wie die Servicedateien als Beispiel ausgeliefert, wie man Mails mit dem Programm `msmtp` versenden kann. `sendmails.sh` läuft in einer

Schleife so lange, bis es eine Datei namens `/tmp/sendmails.stop` (eine sogenannte stop-Datei) gefunden hat. Man kann diese stop-Datei erzeugen, in dem man `sendmails.sh` mit dem Parameter `stop` aufruft.

9.4) send_telegram_to_bot.service

Dieser Service wird eingesetzt, um Telegram-Nachrichten (die message2action erzeugt hat) an die Chat-ID des Bot zu senden. In der Servicedatei wird das Skript `send_telegram_to_bot.sh` aufgerufen, das nach Textdateien im Ausgansgsverzeichnis sucht, die dann mit `curl` an den Chat gesendet werden.

9.5) telegram_check_updates.service

Der Service `telegram_check_updates.service` muss angepasst werden, dass der eigene Token in der Zeile beginnend mit "`ExecStart=`" eingetragen wird.

10) Kompilieren und Linken des Quellcodes

Der Quellcode kann nach eigenem Bedürfnissen angepasst werden. Zum Erstellen der ausführbaren Datei kann das Makefile namens `makefile` im Verzeichnis `message2action` benutzt werden.
Sofern Sie Anpassungen vorgenommen haben, freue ich mich darüber, wenn ich dazu informiert werde. Vielleicht übernehme ich ja diese Anpassungen.
Mit dem Define namens `DEVSTAGE` im Quellcode gibt das Programm mehr Details über die laufenden Vorgänge aus. Das kann beim Debugging hilfreich sein. Das Define ist ganz am Anfang von `message2action.cpp` zu finden.

11) Sicherheit

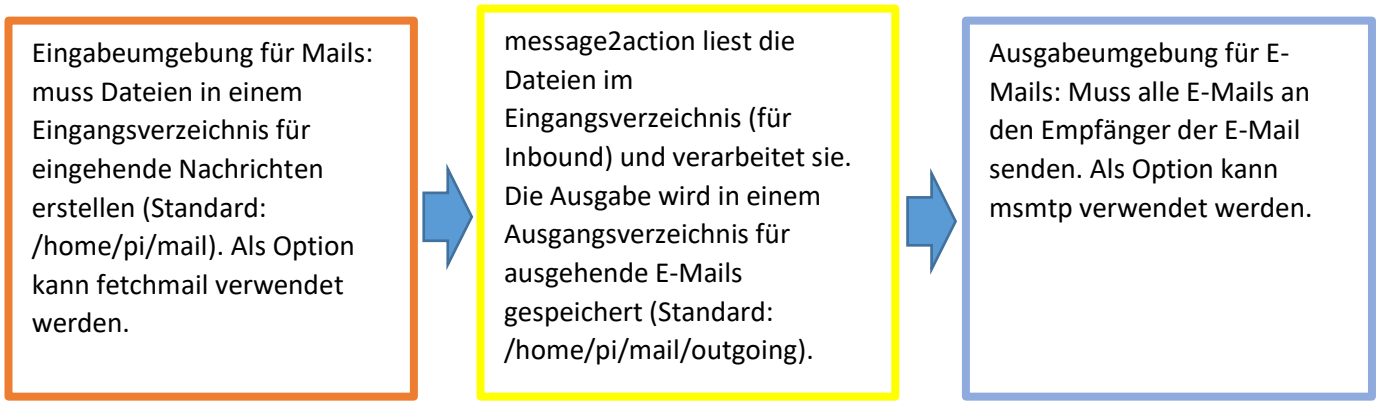
Die „SMS Server Tools 3" schreiben die Handynummer in die SMS-Datei. Diese Handynummer wird von dem Programm auf ihre Gültigkeit überprüft. In der ini-Datei können Sie 10 erlaubte Mobiltelefonnummern angeben. Das heißt, wenn jemand anderes eine SMS an Ihren Pi sendet, funktioniert das Programm damit nicht. Alle gültigen Mobiltelefonnummern erhalten eine Nachricht mit einer Warnung, wenn eine ungültige Telefonnummer erkannt wird. Gleiches gilt auch für E-Mails. In der INI-Datei können Sie 10 gültige Mailadressen angeben. Wenn Ihr Konto eine E-Mail von einem unbekannten Konto erhält, erhalten alle E-Mail-Adressen eine E-Mail mit einer Warnung. Dies kann ein Problem sein, wenn Sie Spam-Mails erhalten, da jedes Mal, wenn eine Spam-Mail empfangen wird, eine E-Mail an alle registrierten E-Mail-Konten in der INI-Datei gesendet wird. Dasselbe gilt auch für Telegram-Nachrichten. In der ini-Datei können Sie 10 gültige Chat-IDs angeben. Wenn Ihr Bot dennoch eine Nachricht von einem Chat erhält, der nicht angegeben ist, erhalten alle Chat-IDs eine Warnmeldung.

12) Zukünftige Features

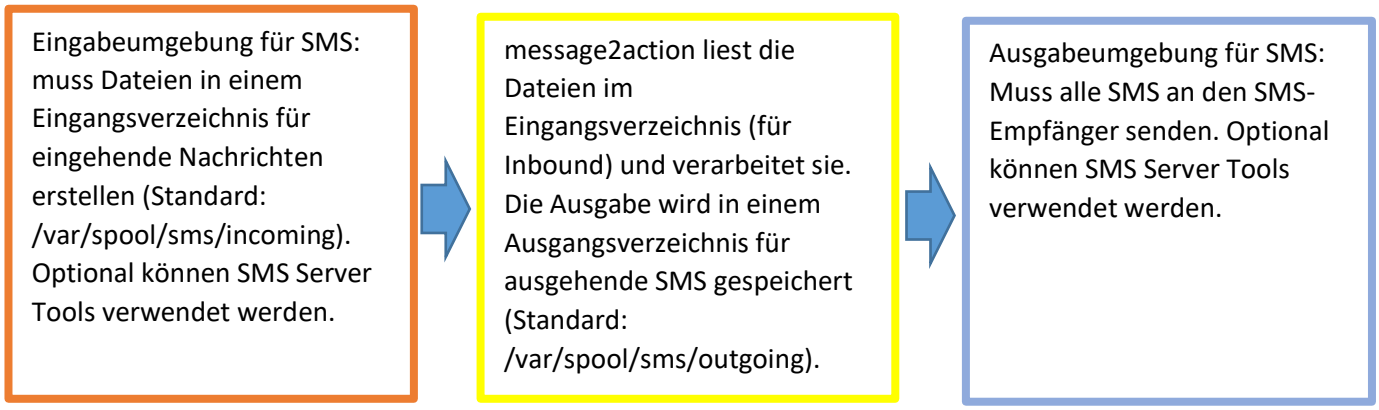
a) Es könnte interessant sein, die GPIOs abzufragen und zum Beispiel Pins per Befehl kurzzuschließen.
b) Aktuell muss man einen Texteditor nutzen, um die ini-Datei anzupassen. Ein Programm mit einer GUI könnte eine Alternative dazu sein.
c) Vielleicht macht es Sinn, die Inhalte der Logdatei (`message2action.log`) in der Datenbank statt in einer Textdatei zu speichern. Dies könnte bei der Analyse von Fehlern hilfreich sein.
d) Es könnte interessant sein, auch die im Raspberry Pi OS eingebauten Mailaccunts zur Kommunikation zu nutzen. So könnte man direkt vom Pi aus eine Nachricht an das Programm senden, ohne über SMS oder das Internet zu gehen.
e) Aktuell werdenn SMS, Mails und Telegram-Nachrichten auf Dateibasis behandelt. In Zukunft könnte eine Behandlung von Mails und Telegram-Nachrichten auch online interessant sein, so dass man keine Dateischnittstelle mehr hat

13) Schnittstelle für Nachrichten

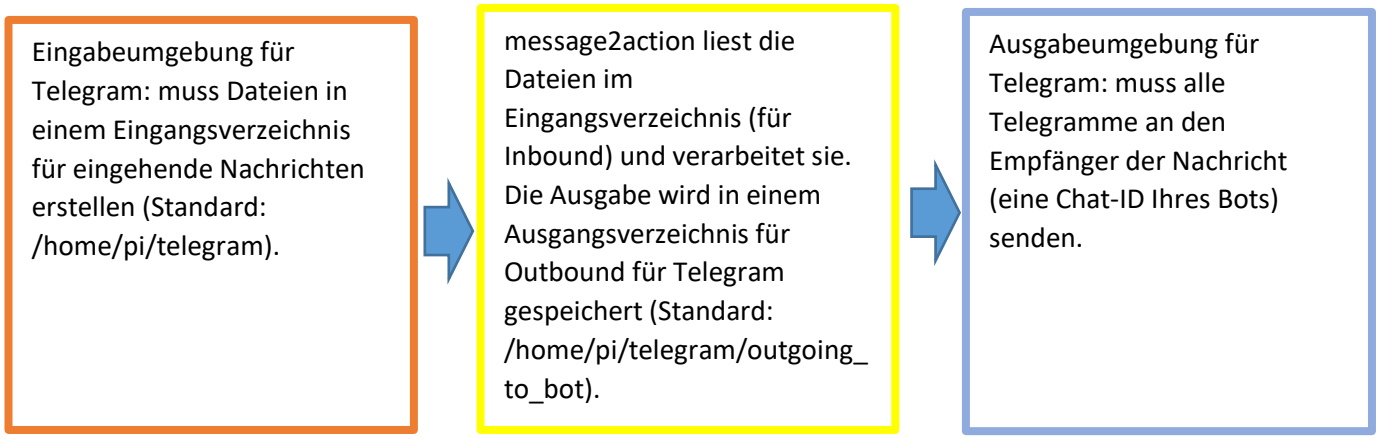
Die Schnittstellen zu message2action sind asynchron und dateibasiert. Das bedeutet, dass alle Nachrichten Dateien sind, die an message2action gesendet werden. Und alle Nachrichten, die von message2action zurückgegeben werden, sind ebenfalls Dateien. Die Konsequenz dieses Designs ist, dass Sie eine Umgebung zum Erstellen der Dateien einrichten müssen, die von message2action verarbeitet werden sollen. Und Ihre Umgebung muss in der Lage sein, die von message2action eingerichteten Dateien aufzunehmen, um sie an den gewünschten Empfänger zu liefern.
Einen allgemeinen Überblick über E-Mails finden Sie im folgenden Diagramm:



Hier ist das entsprechende Diagramm für SMS:



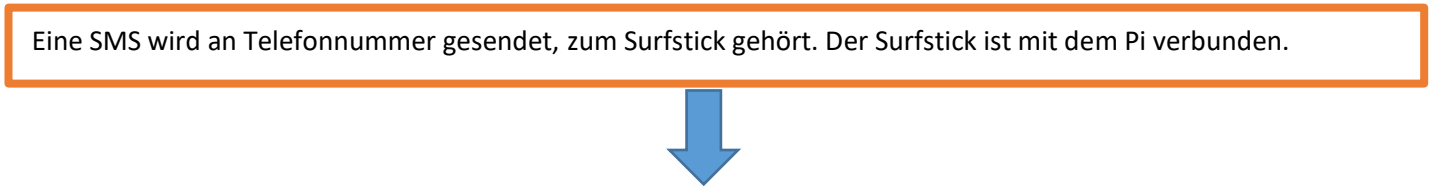
Hier ist das entsprechende Diagramm für Telegram-Nachrichten:

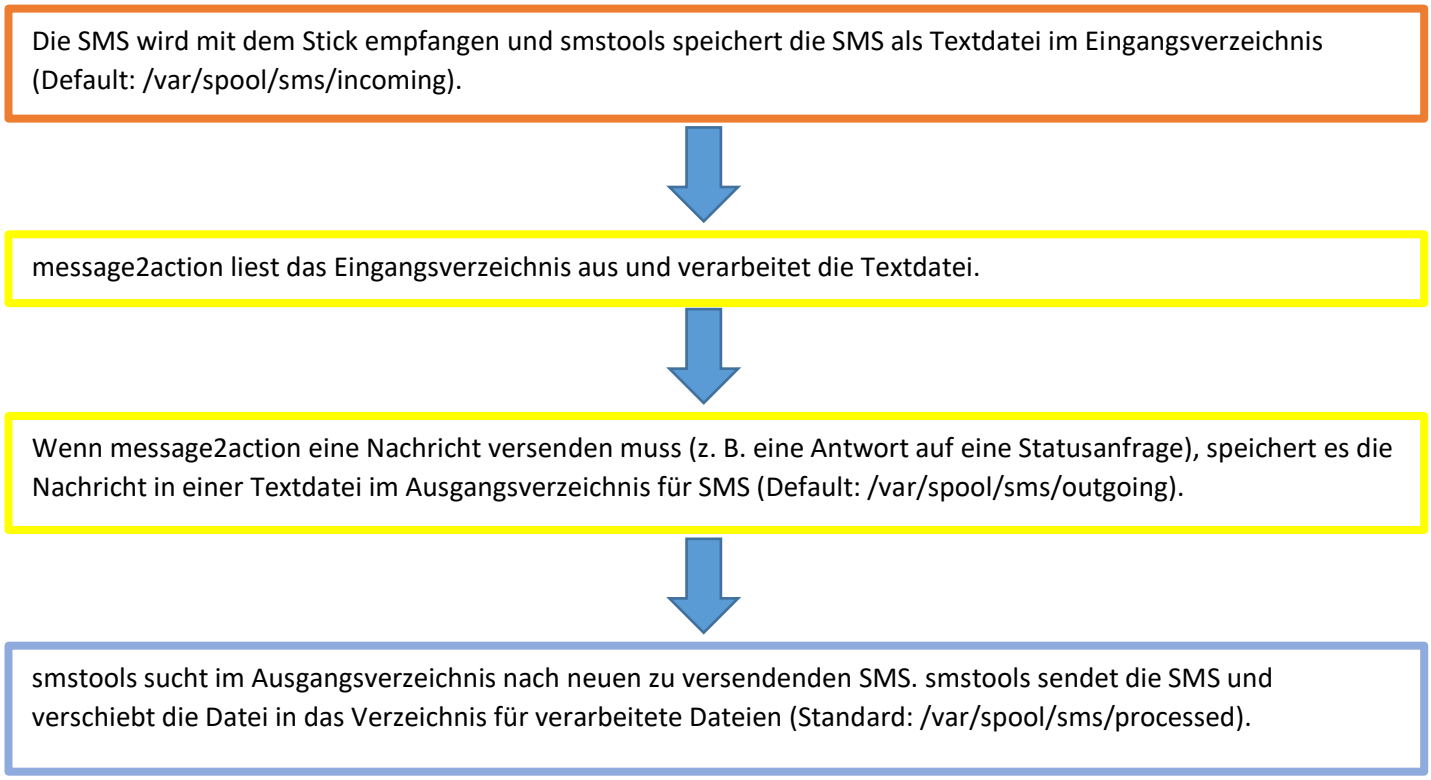


Dieses Design erleichtert die Integration der nächsten Art von Nachrichten, die von messag2action verarbeitet werden sollen. Eine online-Verarbeitung wäre zeitgemäßer und auch für den Nutzer einfacher. Dann müssten aber alle Kanäle in die Software implementiert werden. Stattdessen können einfach vorgefertigte Programme wie smstools, fetchmail, msmtp, curl usw. verwendet werden.

13.1) SMS

Dies ist der Ablauf zum Verarbeiten von SMS auf Ihrem Pi mit message2action. Die Farben der Eingabeumgebung, message2action und der Ausgabeumgebung werden wie oben beibehalten.





13.1.1) Namenkonvention und Aufbau der Textdateien

Hier ist ein Beispiel einer Datei, die von SMStools für eine eingehende SMS erstellt wurde. (Echte Werte werden durch „DUMMY“ ersetzt):

```
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
```

Die erste Zeile enthält nach dem Stichwort From: die Handynummer, von der die SMS stammt. Dieser Wert wird in message2action verwendet, um zu überprüfen, ob eine SMS von einem gültigen Mobiltelefon gekommen ist. Die allerletzte Zeile enthält den Text, der in der SMS gesendet wurde. Dies ist der Befehl, den Sie an Ihren Pi senden und der von message2action gelesen wird. Vor der allerletzten Zeile finden Sie eine Leerzeile.
Die Dateien werden von SMStools mit dieser Namenskonvention gespeichert: GSM1.abxyz, wobei abxyz eine zufällige Zeichenfolge mit 5 Zeichen darstellt. Hier ist ein Beispiel einer Datei, die von message2action versendet wurde:

```
To: 4999

Your text to your cell phone
```

Die Dateien werden von message2action mit dieser Namenskonvention gespeichert:
GSM1.<Datum>_<Uhrzeit>_<Anzahl der CPU-Ticks>. Die <Anzahl der CPU-Ticks> trägt dazu bei, den Namen der Datei eindeutig zu machen.

13.2) Mail

Dies ist der Ablauf zum Verarbeiten von E-Mails auf Ihrem Pi mit message2action unter Verwendung der optionalen Software fetchmail und msmtp:

Sie senden eine E-Mail von Ihrem Standardkonto (Beispiel: alterego@mailmaster.com) an das Konto, das mit Ihrem Pi verknüpft ist (Beispiel: mypi@mydomain.com).

Systemd auf Ihrem Pi ruft `getmails.service` auf (wie in `getmails.timer` konfiguriert). In `getmails.service` wird fetchmail aufgerufen, um die empfangene E-Mail vom Konto mypi@mydomain.com herunterzuladen. fetchmail ruft ein Shell-Skript auf, das die heruntergeladene E-Mail im Eingangsverzeichnis für E-Mails speichert (Default: /home/pi/mail).

message2action liest die Textdatei im Eingangsverzeichnis und verarbeitet sie.

Wenn message2action eine E-Mail versenden muss (zum Beispiel eine Antwort auf eine Statusanfrage), speichert es die Nachricht in einer Textdatei im Ausgangsverzeichnis für E-Mails (Default: /home/pi/mail/outgoing/mail .txt).

`systemd` ruft beim Booten Ihres Pi `sendmails.service` auf. `sendmails.service` ruft das Skript auf, das `msmtp` nutzt (Default: /home/pi/message2action/sendmails.sh). `msmtp` sendet eine E-Mail über mypi@mydomain.com an die E-Mail-Adresse, von der die ursprüngliche E-Mail kam: alterego@mailmaster.com

13.2.1) Namenkonvention und Aufbau der Textdateien

Hier ist ein Beispiel einer Textdatei für eine eingehende E-Mail:

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

Die E-Mail-Adresse wird von message2action verwendet, um zu überprüfen, ob wir eine Nachricht von einem gültigen E-Mail-Konto erhalten haben. Der Text nach `Subject:` enthält den Befehl, den Sie an Ihren Pi senden.

Hier ist ein Beispiel für eine Textdatei, die eine ausgehende E-Mail enthält:

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

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

Die Dateien werden mit dieser Namenskonvention gespeichert: `<Datum>_<Uhrzeit>_<Anzahl der CPU-Ticks>.mail`. Die `<Anzahl der CPU-Ticks>` trägt dazu bei, den Namen der Datei eindeutig zu machen.

13.3) Telegram

Dies ist der Ablauf für die Verarbeitung von Telegram-Nachrichten auf Ihrem Pi mit message2action unter Verwendung der optionalen Software Curl:

Sie senden eine Nachricht in Ihrer Telegram-App oder dem webbasierten Client an Ihren Bot, indem man einen Text in den Chat eingibt.

`systemd` auf Ihrem Pi ruft `telegram_check_updates.service` beim Booten des Pi auf. Im `telegram_check_updates.service` wird ein Programm aufgerufen, um Textnachrichten zu lesen, die Sie in Ihrem Chat mit Ihrem Telegram-Bot eingegeben haben. Die extrahierte Nachricht wird im Eingangsverzeichnis für Telegramme gespeichert (Default: /home/pi/telegram).

message2action liest die Textdatei im Eingangsverzeichnis und verarbeitet sie.

Wenn message2action eine Nachricht an Ihren Bot senden muss (z. B. eine Antwort auf eine Statusanfrage), speichert es die Nachricht in einer Textdatei im Ausgangsverzeichnis für Telegramme (Default: /home/pi/telegram/ outgoing_to_bot).

`systemd` ruft `send_telegram_to_bot.service` beim Booten Ihres Pi auf. `send_telegram_to_bot.service` ruft das Skript `send_telegram_to_bot.sh` auf. Dieses Shell-Skript sucht im Ausgangsverzeichnis für Telegramme nach Textdateien (Default: /home/pi/telegram/outgoing_to_bot). Wenn eine Textdatei gefunden wird, wird diese gelesen und an Ihre Chat-ID gesendet.

13.3.1) Namenkonvention und Aufbau der Textdateien

Hier ist ein Beispiel für eine Textdatei, die eine Nachricht aus einem Chat mit einem Bot enthält:

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

Die erste Zeile ist derzeit immer gleich. Die zweite Zeile enthält die Chat-ID mit Ihrem Bot. Die letzte Zeile enthält die Nachricht, die Sie in Ihrem Chat mit Ihrem Bot eingegeben haben.
Hier ist ein Beispiel für eine ausgehende Nachricht, die mit Ihrem Bot an die Chat-ID gesendet wird:

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

Die erste Zeile speichert den Token Ihres Bots. In der zweiten Zeile wird die Chat-ID mit Ihrem Bot gespeichert. In der dritten Zeile wird die Nachricht gespeichert, die in Ihrem Chat mit dem Bot sichtbar ist.

14) Releases des Programms

Im August 2016 wurde das Release 0.13 des Vorgängers namens „sms2action“ veröffentlicht. Hier sind die Änderungen, die in den Versionen von message2action vorgenommen wurden:

14.1) Release 0.27

- Verarbeitung von Mails wurde aufgenommen.
- Der Parameter `check_online_status_waittime` wurde in die ini-Datei aufgenommen.
- Wenn das Programm läuft, kann man es durch Drücken einer Taste wieder beenden.
- Systemd-Services `getmails.service`, `getmails.timer` und `sendmails.service` wurden bereitgestellt.
- Eine Dokumentation im PDF-Format wurde veröffentlicht.
- Kommandos an den Pi werden in Großbuchstaben gewandelt (Beispiel: “status” wird zu “STATUS”).
- Verschiedene Fehler wurden behoben.

14.2) Release 0.28

- Ein Fehler beim Lesen der INI-Datei wurde behoben.
- Ein Fehler beim Lesen der Zeile in einer E-Mail, die den Befehl enthält, wurde behoben (es wurde angenommen, dass die neue Zeile immer am Ende der Zeile steht).
- Die Initialisierung globaler Variablen erfolgt jetzt mit konstanten Werten, die mit der Anweisung „define“ definiert wurden.
- In systemd-files wurde der Aufruf des Programms geändert.

14.3) Release 0.30

- Der Parameter `fullpath_systemd_shell_mail` wird nicht mehr benötigt, da diese Funktion in systemd-Dateien verschoben wurde.
- Der Parameter `fullpath_outgoing_mail` wird nicht mehr benötigt, da ein Benennungsverfahren für Dateinamen implementiert wurde.
- Einige Fehler behoben.
- Verarbeitung von Telegram-Nachrichten wurde hinzugefügt.
- Die Benennung der Dateien, die zum Versenden von Nachrichten verwendet werden, wurde geändert.

14.4) Release 0.31

- Die Installation der Python-Pathlib-Bibliothek wurde in dieser Dokumentation hinzugefügt.
- Ein kleiner Fehler in message2action wurde behoben.

14.5) Release 0.32

Für SMS wurde die Art der Suche nach dem Befehl geändert. In einigen Fällen gibt es in der Datei keinen Eintrag für „IMEI:“ und ich war mir sicher, dass dies nur bei meinem Stick der Fall ist. Aber das war falsch. Jetzt sucht das Programm einfach in der Textdatei nach einer Leerzeile und nehme diese als Trennzeichen für den Befehl.

14.6) Release 0.33

3 neue Befehle wurden implementiert: `message2action_help_command`, `message2action_questionmark_command` und `message2action_config_command`. Jetzt können Sie Hilfe erhalten, indem Sie „help“ oder „?“ senden (beide Werte sind Default). Wenn Sie die aktuelle Konfiguration erhalten möchten, können Sie „config“ als Standard verwenden. Die Skripte `sendmails.sh` und `send_telegram_to_bot.sh` mussten angepasst werden.

14.7) Release 0.34

- Wenn message2action als systemd-Dienst ausgeführt wird, versucht es, Tastatureingaben abzufangen. Dies ist jedoch nicht sinnvoll und führt zu der Fehlermeldung „Error: Terminalsettings could not be determined!“ und „Error: Terminalsettings could not be set to new values!“. Um mit diesem Modus umzugehen, wurde ein neuer Befehlszeilenparameter namens „daemon“ implementiert. Es kann die Werte y oder n annehmen. Wenn Sie daemon=y angeben, versucht das Programm nicht mehr, Tastatureingaben abzufangen.
- Das Löschen von alten Dateien ist jetzt möglich. In der Ini-Datei können Sie mit dem Parameter `garbage_collection_days` die Anzahl der Tage angeben, wie alt Dateien in den verarbeiteten Verzeichnissen werden dürfen.

14.8) Release 0.35

Wenn message2action Befehle wie „reboot“ oder „shutdown“ empfängt, versucht es nun, vorher eine Bestätigung zurückzusenden, um zu signalisieren, dass der Befehl empfangen wurde und ausgeführt wird. Dies gilt auch für die Befehle CMD00 … CMD09.

14.9) Release 0.36

2 Fehler beim Senden von WOL an das System wurden behoben. Zunächst wurde unterdrückt, nach dem Versenden von WOL eine Bestätigung über den Status an den Absender zurückzusenden. Der zweite Fehler führte dazu, dass das Ergebnis von ping (zur Überprüfung des Online-Status) falsch interpretiert wurde und message2action in einer Endlosschleife lief.

14.10) Release 0.39

Release 0.37 und 0.38 wurden nicht veröffentlicht. Mehrere Fehler bezüglich der Puffernutzung und dem Vergleich von Zeichenfolgen wurden behoben. Das Verfahren zum Herunterfahren und Neustarten des Raspberry wurde geändert: sudo wurde vor dem Befehl hinzugefügt.

14.11) Release 0.40

Für Raspbian OS „Buster“ wird das Programm `ssmtp` nicht mehr gepflegt. Also musste ich ein anderes Programm finden und bin auf `msmtp` umgestiegen. Die Datei `sendmails.sh` musste angepasst werden.

14.12) Release 0.50

Der Quellcode wurde von C auf C++ geändert. Damit wurden einige neue Funktionen aktiviert:

- Es ist nicht mehr erforderlich, mit Befehlszeilenparametern anzugeben, welcher Eingangskanal zum Empfangen von Nachrichten verwendet wird. Dies legen Sie in der Ini-Datei fest. Nun ist eindeutig, dass eine Antwort auf demselben Kanal gesendet wird, auf dem sie empfangen wurde.
- Multithreading wird verwendet, um mehrere Inbound-Kanäle parallel zu lesen. Dies ist hilfreich, wenn ein eingehender Kanal ausfällt. Sie können einen alternativen Kanal verwenden, um beispielsweise den Pi neu zu starten.
- Ich habe die Verwendung von Python zum Empfang von Telegram-Nachrichten durch ein C++-Programm ersetzt. Dieses Programm heißt `telegram_check_updates`.
- Parameter in der Ini-Datei zum Speichern der Prozess-ID einer Instanz werden nicht mehr benötigt.

14.13) Release 1.10

Die erste Version, die auf Github veröffentlicht wurde.

FAQ (Frequently asked questions)

#1: Q: Warum gibt es zwei Möglichkeiten, das Programm message2action zu stoppen?
A: Zuerst wurde die Methode mit dem Parameter „stopfile“ implementiert. Dieser Weg ist etwas langsamer als der Parameter „stopsignal“. Wenn Sie den Parameter stopfile aufrufen, wird eine Nachrichtendatei erstellt und message2action liest diese Datei. Wenn Sie jedoch in der ini-Datei die Anzahl der Sekunden angegeben haben, die im nächsten Zyklus gewartet werden sollen, müssen Sie warten, bis der Zyklus beendet ist. Dies wird durch die Schlaffunktion in C realisiert.
Wenn Sie das Programm mit dem Parameter „stopsignal“ aufrufen, wird die Sleep-Funktion sofort unterbrochen. Dies ist für die Implementierung von message2action als Dienst auf Ihrem Pi erforderlich, da Sie alle Dienste sehr schnell stoppen möchten, wenn Sie Ihren Pi herunterfahren oder neu starten.

#2: Q: Welche Geräte und Software benötige ich, um message2action zu testen?
A: Sie benötigen weder SMS-Tools noch einen Surfstick oder ähnliches. Sie können die Beispieldateien nutzen und modifizieren, um zu prüfen, ob das Programm Ihren Wünschen entspricht. Sie benötigen auf Ihrem Pi auch kein Mailprogramm, da Sie auch hierfür Beispieldateien verwenden können. Gleiches gilt für Telegram.

#3: Q: In welchen Sprachen kann ich den Autor kontaktieren?
A: In deutsch und englisch.

#4: Warum gibt es einen `getmails.timer`, aber keinen `sendmails.timer`?

A: Als ich mit der Entwicklung der Software begann, konnte ich in Bash keine Skripte schreiben. Das hat sich ein wenig geändert. Ich habe mit `getmails.service` begonnen und einen `getmails.timer` hinzugefügt, weil ich es einfach fand, systemd die Steuerung der Planung zu überlassen. Einige Monate später erstellte ich `sendmails.service` und konnte dann ein `sendmails.sh` schreiben, die den Scheduler enthält. Ich bin mir nicht ganz sicher, welcher Weg der bessere ist: Übergabe des Scheduling-Job an systemd oder diesen Aufruf selbst auszuführen. Daher habe ich beide Methoden stehen lassen, um daraus zu lernen.

#5: Gibt es auch eine Unterstützung für den Telegram-Client (telegram-cli)?
A: Nein nicht mehr. Ich fand es sehr schwierig, das Telegram-CLI zu implementieren, und ich fand es einfacher, Telegram-Bots zu verwenden. Ich habe es geschafft, die Telegram-CLI zum Laufen zu bringen, aber für mich war es keine zufriedenstellende Lösung.

#6: Kann ich mehr als einen Nachrichtentyp parallel verarbeiten?
A: Ja. Dazu muss man lediglich in der ini-Datei die Flags für `inbound_sms_file_flag`, `inbound_mail_file_flag` und `inbound_telegram_file_flag` auf `true` setzen.

15) Kontakt zum Autor des Programms

Wenn Sie Fragen oder Anmerkungen haben, erreichen Sie mich per E-Mail an: `freeofcharge@t-online.de`. Bitte geben Sie diese Informationen an, damit wir Sie einfacher unterstützen können:

Art der Information	Quelle
Release von message2action	In einem Terminal wird einfach eingegeben: `./message2action`
Logdatei von message2action	Siehe Pfad, wo die Logdatei liegt (Default: `/home/pi/message2action`): `message2action.log`
Name und Version des Betriebssystems (in den allermeisten Fällen wird das Raspberry OS sein)	In einem Terminal wird einfach eingegeben: `cat /etc/os-release`
Numme des Kernels und Name der CPU	In einem Terminal wird einfach eingegeben: `uname -a`

16) Legal