UCall

COLLABORATORS			
	TITLE : UCall		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 UCall **1**

1.1 main 1

1.2 Einleitung 1

1.3 Systemvoraussetzungen 2

1.4 Installation 3

1.5 Das Programm 3

1.6 Konfigurationsfile 3

1.7 Loginscript 5

1.8 Bugs & anderes 13

Chapter 1

UCall

1.1 main

```
                          UCALL - universaler Mailer
              V 1.9 vom 24.8.1995
        Freeware von Lunqual%mab@wsb.freinet.de


               Einleitung

               Systemvoraussetzungen

              Installation

              Das Programm

               Konfigurationsfile

               Loginscript

               Bugs & anderes
```

1.2 Einleitung

UCall ist ein universaler Mailer, d.h. ein Programm, was selbständig eine Mailbox anruft, Pointpuffer abholt und verschickt. Im Gegensatz zu gängigen Produkten verwendet UCall keine festgelegte Loginprozedur, sondern wird von einem Loginscript gesteuert.

Nun, wieso gibt es das Programm überhaupt ? Gibts nicht genug andere MAiler ?? Naja, doch, schon... aber diese Mailer fallen in 2 Kategorien:

erstens die Teile, die zwar im allgemeinen funktionieren, aber nicht im besonderen, weil meine Leib- und Magenbox halt vor das login noch ein Linefeed gesetzt haben will, was den dämlichen Mailer ganz aus dem Tritt bringt, weil er halt ohne zu fragen gleich den Usernamen + linefeed schickt !

zweitens die Programme, die zwar im allgemeinen wie auch im besonderen funktionieren, aber mir mit der hartnäckigen Behauptung dass ich irgendeinem Raubritter 20 Magg schulden würde stark auf die Nerven fallen.

Deswegen zählt UCall zur dritten Kategorie: Lunqualitätsware ! Die neigt zwar im allgemeinen auch zum Versagen, aber funktioniert dafür im besonderen ganz gut. ;)

Aber trotzdem:

```
*****************************************************************

  UCall ist FREEWARE. Das Programm kann ohne Einschränkung
  kopiert und/oder weitergegeben werden, wenn das Archiv
  vollständig bleibt.

  Folgende Dateien sind in diesem Archiv:

     Bin (dir)
          catalogs (dir)
               english (dir)
                 ucall.catalog
       UCall                          UCall.info
     Source (dir)
       english.ct                     english.ct.info
       SCOPTIONS                      SCoptions.info
       ucall.c                        ucall.c.info
       ucall.cd                       ucall.cd.info
       ucall.h                        ucall.h.info
       UCall.lnk
     Dok (dir)
       UCall.guide                    UCall.guide.info
       ucall.history                  ucall.history.info
     Script (dir)
       ucall.config                   ucall.config.info
       znetz.login                    znetz.login.info
  Bin.info                       Dok.info
  Script.info                    Source.info

  Der Autor übernimmt keinerlei Haftung für Schäden oder
  Fehlfunktionen, die aus der Benutzung des Programms
  entstehen können.

*****************************************************************
```

1.3 Systemvoraussetzungen

Systemvoraussetzungen von UCall:

1. Kickstart/Workbench 2.0 oder besser
   Wer das noch nicht hat, dem is nimmer zu helfen ! ;)

2. metaxpr.library V2 oder besser.

3. Optional Locale.library (Keine Bedingung)

1.4 Installation

Da gibts nicht viel zu installieren.

Kopieren Sie die metaxpr.library ins LIBS:-Verzeichnis, sofern sich dort noch keine befindet und das UCall Programm irgendwohin.

Die eingebaute Sprache ist Deutsch. Wenn sie (ab WB 2.1+) die englische Version benutzen wollen, sollten sie die Datei bin/catalogs/english/ucall.catalog nach LOCALE:catalogs/english/ kopieren

1.5 Das Programm

UCall kann nur aus einer Shell heraus verwendet werden.

Programmaufruf: 'UCall <
konfigurationsfile
> <
loginscript
>'

Beide Files MÜSSEN vorhanden sein. Im Konfigurationsfile stehen alle Angaben über Modem/Schnittstelle Übertragunsprotokolle usw., während das Loginscript Telefonnummern u.ä. enthält.

Das Programm kann jederzeit durch ein herzhaftes control-c abgebrochen werden.

1.6 Konfigurationsfile

Das Konfigurationsfile enthält die Befehle für die verwendeten SChnittstelle,Protokolle usw.

Die Konfigurationsdaten können in einem beliebigen Textfile (bspw. ums-konfiguration) stehen.

Die Daten beginnen mit der Zeile "UCall-config" und enden entweder am Ende des Files oder bei einer weiteren Zeile mit "UCall-config"

Das format einer Zeile ist

    befehl=<wert>

wobei der Wert DIREKT auf das Gleichheitszeichen folgen muss.

Befehle sind :

```
DEVICE=<device>       Das verwendete Device, z.B.
                      serial.device, nullmodem.device etc
                      Voreinstellung = serial.device

UNIT=<unit>           Die verwendete Unit, z.B. 0
                      Voreinstellung = 0

BAUD=<baud>           Die Schnittstellengeschwindigkeit, z.B.
                      19200, 38400 usw.
                      Voreinstellung = 19200

BUSY     =<busy>
NOCARRIER=<nocarrier>
DELAYED=<delayed>
ERROR    =<error>
CONNECT  =<connect>   Hier werden die
                      Zeichenketten eingetragen,
                      die das Modem im entsprechenden Fall
                      schickt.

XPRLIB=<xprlib>       Hier ist das verwendete
                      Übertragungsprotokoll
                      einzutragen, z.B. xprzmodem.library usw.
                      Voreinstellung = xprzmodem.library

XPROPTS=<opts>        Die Optionen des Protokolls.
                      Keine Voreinstellung

DATABITS     =<7/8>   Anzahl der Datenbits
                      Voreinstellung 8

PARITY       =<NONE/EVEN/ODD> Parität
                      Voreinstellung None

STOPBITS     =<0/1/2> Anzahl der Stopbits
                      Voreinstellung 1

RTS/CTS               RTS/CTS-Handshake
                      Voreinstellung RTS/CTS

XON/XOFF              XON/XOFF-Handshake


Beispiel-Konfiguration:
----------------------------schnipp----------------------

UCall-config
DEVICE       =nullmodem.device
UNIT         =0
BAUD         =19200
BUSY         =BUSY
NOCARRIER    =NO CARRIER
NODIALTONE   =NO DIALTONE
DELAYED      =DELAYED
ERROR        =ERROR
CONNECT      =CONNECT
XPRLIB       =xprzmodem.library
```

```
XPROPTS      =TN,OR,B256,F0,E20,AN,DN,KY,SN,RN,P
DATABITS     =8
PARITY       =NONE
STOPBITS     =1
RTC/CTS


----------------------------schnapp----------------------
```

1.7 Loginscript

Hier wird der Loginvorgang "programmiert"

Das geht so, dass Zeichenketten ans Modem gesendet werden, zum initialisieren der SChnittstelle oder zum Wählen und im Gegenzug auf gewisse Schlüsselworte wie "Username:" gewartet wird um entsprechend zu reagieren.

Die vorhandenen Befehle im einzelnen:

1. Allgemeine Befehle

end	Programmende
delay <sekunden>	Pause von x sekunden. Wenn kein Wert angegeben ist, wird die Länge der Anwahlpause (siehe pause-befehl) benutzt.
getenv <var> =<ev-var>	Der variablen <var> wird der Wert der Standard-AmigaDos-Environment-Variablen <ev-var> zugewiesen. <ev-var> MUSS unmittelbar nach dem Gleichheitszeichen kommen. Beispiel : getenv ks =Kickstart im Script wird dann an jeder Stelle, an der $ks steht, der Inhalt der Env-Variablen 'Kickstart' eingesetzt.
linebuffer <wert>	Die maximale Länge einer Scriptzeile Voreinstellung ist 1024, mehr ist jederzeit möglich, Mindestwert ist 512 Byte
msg <Message>	Eine Message wird angezeigt. Der Text kann auch \n und \r sowie \x<HEXZAHL> Statements enthalten.
pause <wert>	Die Anwahlpause in Sekunden, wenn die Nummer besetzt ist, oder
protocol <datei>	öffnet die angegebene Protokolldatei
protmsg <message>	schreibt die <message> in die Protokolldatei

showlogin <file>
Der gesamte Loginvorgang wird in das File geschrieben. Wenn kein File angegeben ist, dann wird "con:0/0/640/400/UCall-Login" verwendet.

set <var> =<wert>
Die Variable <var> wird auf den Wert <wert> gesetzt. Der Wert MUSS unmittelbar nach dem Gleichheitszeichen kommen.
Beispiel :

```
set username    =LUNQUAL\r\n
```

im Script wird dann an jeder Stelle, an der $username steht, LUNQUAL\r\n eingesetzt.

setenv <var> =<wert>
Die (Standard-AmigaDos-Environment-)Variable <var> wird auf den Wert <wert> gesetzt. Der Wert MUSS unmittelbar nach dem Gleichheitszeichen kommen.
Beispiel :

```
setenv erfolg    =ok
```

Diese Environment-variablen stehen dann global dem System zur Verfügung und können mit getenv ausgelesen werden.

system
Das angegebene Kommando wird von der Shell ausgeführt

timeout <wert>
Die Zeit, die ein wait-Befehl höchstens brauchen darf. Wird diese Zeit überschritten wird die Programmausführung am timeout_label fortgesetzt oder abgebrochen wenn dieses nicht vorhanden ist.

timeout_label
Das Script wird nach einem Timeout an dieser Stelle fortgesetzt

workdir
Das neue Arbeitsverzeichnis wird gesetzt. Muss vor dem Download gesetzt werden, da empfangene Files ins momentane Verzeichnis gehen

2. Senden / Empfangen

connect <Nummer>
Das Programm versucht einen Connect aufzubauen.
Wenn die angegebenen Nummer besetzt ist, wird vor dem nächsten Anwahlversuch <pause> sekunden gewartet

getfile <name>
eine Datei wird mit dem voreingestelleten xpr-protokoll empfangen. Hier ist noch ein Bug, die Datei landet

im aktuellen Verzeichnis ;) (Im Zweifelsfall mit
workdir setzen)
Wenn erfolgreich wird das Flag 1 gesetzt

putfile <name> Die angegebene Datei wird mit dem voreingestellten
xpr-protokoll gesendet.
Wenn erfolgreich, wird das Flag 1 gesetzt.

send <Zeichenkette> Die Angegebene Zeichenkette wird
an die SChnittstelle ausgegeben.

Beispiel:

send Das ist ein Test\r\n

sendfile <file> Der Inhalt des angegebenen Files wird UNVERÄNDERT
als Text an die Schnittstelle geschickt. Nicht
zu verwechseln mit 'putfile' ! Da sich das
aktuelle Verzeichnis im Programmverlauf ändern kann,
wird empfohlen, das File mit komplettem Pfad
anzugeben.

wait <Muster>,<Muster>,. Es wird auf den die angegebenen Texte von der
SChnittstelle gewartet. Die Texte dürfen auch
Standard-Amigados Muster enthalten, z.b. "#?wort:",
und werden durch Kommata voneinander getrennt.
Es sind maximal 16 Muster zu je 64 Zeichen zulässig.
Wer mehr braucht, muss in UCall.h das #define ↩
MAX_BREAKCHARS
ändern und den ganzen Unfug neu übersetzen.
Wenn diese Funktion ein Muster gefunden hat, wird
dieses gespeichert und steht einer folgenden
if- Anweisung zur Verfügung.

WICHTIG: Das Programm vergleicht jeweils mit dem LETZTEN
Zeichen des Textes, das deswegen KEIN muster SEIN DARF !.

Beispiel:

wait NO#?CARRIER,ERROR,OK

HINWEIS: \n oder \r darf NUR AM ENDE der Zeile
vorkommen !

Die Anweisung :

wait NO CARRIER\n,ERROR

funktioniert NICHT !!!

3. Kontrollstrukturen

gosub <label> springt zu dem angegebenen label
und setzt das Programm nach einer

return-Anweisung an der Zeile nach
gosub... fort

Beispiel:

```
gosub Modeminit
.
.
.
end

label Modeminit
send ATZ\r
wait ERROR,OK
if #?ERROR end
return
```

goto <label> setzt das Script an dem entsprechenden Label fort.

Beispiel:

```
------------------------
send ATZ\r
wait ERROR,OK
if #?ERROR#? goto init_error
connect ATDP1234\r
..
..
..
end

label init error
msg Das Modeminit ist fehlgeschlagen\n
end
------------------------
```

if <Muster> <Anweisung>,<Anweisung>..
vergleicht ein Muster mit der Zeile, die von der
letzten Wait- Anweisung geliefert wurde.
Wenn das Muster mit dieser Zeile übereinstimmt,
so wird die abhängige Anweisung ausgeführt, wenn
nicht, so wird zur nächsten Zeile gegangen.
Es sind mehrere abhängige Anweisungen möglich, die
durch Kommata getrennt werden. Jede neue Anweisung
MUSS DIREKT nach dem Komma stehen. \r oder \n mitten
in einer Zeile sind NICHT erlaubt ! Zur Not geht
auch \x0a oder \x0d als Ersatz

Beispiel:

```
----------------------
label anwahl
send ATDP1234\r
wait CONNECT,BUSY
if #?BUSY#? goto anwahl
```

```
                        wait Username:
                        ----------------------

                        Beispiel:

                        if #?urx msg Urx gelesen\x0a,protmsg ($time)Urx gelesen\ ↩
                           x0a


ifvar <variable>,<muster>,<Anweisung>,....

                        vergleicht eine variable mit dem gegebenen Muster.
                        Wenn das die Variable mit diesem Muster,
                        so wird die abhängige Anweisung ausgeführt, wenn
                        nicht, so wird zur nächsten Zeile gegangen.
                        Es sind mehrere abhängige Anweisungen möglich, die
                        durch Kommata getrennt werden. Jede neue Anweisung
                        MUSS DIREKT nach dem Komma stehen. \r oder \n mitten
                        in einer Zeile sind NICHT erlaubt ! Zur Not geht
                        auch \x0a oder \x0d als Ersatz
                        Hinweis: Die Kommata dienen als Trennung der einzelnen
                        Argumente und MÜSSEN vorhanden sein.

                        Beispiel:

                        --------------------------

                        getenv ks  =kickstart
                        ifvar $ks,39#?,msg Wir laufen unter Kickstart 3.0 !\n

                        --------------------------


label <name>            Definiert ein Sprungziel für die goto-Anweisung


                    4. Variablen


Jede Zeile wird auf eventuell vorhandene Variablen untersucht und
diese dann mit ihrem Wert ersetzt. Variablen können mit dem
set-befehl definiert werden

Folgende vordefinierte Variablen stehen zur Verfügung

         $baud
         $buffersize
         $busy
         $connect
         $config           (name des Configfiles)
         $databits
         $delayed
         $device
         $error
         $handshake        (RTS/CTS oder XON/XOFF)
         $nocarrier
         $nodialtone
```

```
$ok
$parity
$script           (name des Scriptfiles)
$stopbits
$unit
$xprlib
$xpropts

Diese entsprechen den Angaben im Konfigurationsfile

$date             (Datum Tag, xx-Monat-xxxx xx:xx:xx)
$filename         enthält unmittelbar nach dem Up/Download
                  den Dateinamen
$filesize         enthält unmittelbar nach dem Up/Download
                  die Grösse der Datei in Bytes
$fileinfo         (enthält unmittelbar nach einem Up/download
                   Name und Grösse des übertragenen Files)
$line             (enthält die letzte empfangene Zeile)
$linebuffer       Die maximale Länge einer Scriptzeile
```

5. Flags

Als kleines Hilfsmittel für was auch immer stehen auch Flags zur Verfügung, die man nach Belieben setzen und Löschen kann Insgesamt stehen 32 Flags zur Verfügung, 8 (0 - 7)sind reserviert und der Rest (8 - 31) ist zur freien Verfügung.

Befehle für Flags sind:

setflag <x> setzt das Flag Nr X.

clearflag <x> löscht das Flag Nr X.

ifflag <x> <befehl> funktioniert wie die if-Anweisung
wenn das Flag X gesetzt ist, dann wird
der Befehl ausgeführt.
Es sind mehrere abhängige Anweisungen möglich, die
durch Kommata getrennt werden. Jede neue Anweisung
MUSS DIREKT nach dem Komma stehen. \r oder \n mitten
in einer Zeile sind NICHT erlaubt ! Zur Not geht
auch \x0a oder \x0d als Ersatz

Die Flags 0,1 und 2 werden momentan vom Programm verwendet.

Flag 0 wird von "putfile" auf 1 gesetzt, wenn ein Upload fehlerhaft war, Flag 1 wird von getfile bei einem fehlerhaften Download auf 1 gesetzt.

Flag 2 wird gesetzt, wenn ein Timeout aufgetreten ist.

Hinweis: Diese Flags werden NICHT wieder zurückgesetzt !

Beispiel-Script:

```
------------------------------------------------------------------------------

; Z-Netz Loginscript
; made by lunqual%mab@wsb.freinet.de

; ein paar Variablen ersma
; Der Username muss bei Z3.8 ZERBERUS heissen,
; bei ZCONNECT dagegen JANUS


; Konfiguration anzeigen
gosub showconfig

; Variablen setzen
set username      =ZERBERUS\r\n
set systemname    =LUNQUAL\r\n
set passwort      =hehehe\r\n
set download_dir  =work2:theanswer/download
set upload_dir    =work:theanswer/upload
set protocolfile  =ucall.protocol

; protokollfile öffnen
gosub startprotokoll

; Anwahlpause 60 Sekunden
pause 60

; Login mitlesen
showlogin con:0/0/640/200/Zcall-Login

; Timeout 60 sekunden
timeout 60
timeout_label

; Abfrage ob bereits ein Timeout aufgetreten ist
ifflag 2 msg ($time) Timeout aufgetreten\n
clearflag 2

; Verzeichnisse leerräumen
gosub zielverzeichnis räumen

gosub modeminit

protmsg ($time) Anwahl 07631 14115\n
connect ATDP 07631 14115\r\n
msg Warte auf Username\n
protmsg ($time) CONNECT\n
wait Username:
msg \x1b[2mSende username\x1b[0m\n
send $username

msg Warte auf Systemname\n
wait Systemname:
msg \x1b[2mSende Systemname\x1b[0m\n
send $systemname

msg Warte auf Passwort\n
```

```
wait #?wort:
msg \x1b[2mSende Passwort\x1b[0m\n
send $passwort

wait running
msg Daten werden gepackt...\n

msg Warte auf Upload\n
wait **b0#?\n
msg \x1b[2mUpload Start\x1b[0m\n
protmsg ($time) UPLOAD START\n
workdir $upload_dir
putfile caller.lha
ifflag 0 gosub uploadfehler
protmsg ($time) UPLOAD ENDE\n
protmsg ($time) INFO: $fileinfo\n
msg ($time) INFO: $fileinfo\n


msg Warte auf Download\n
msg \x1b[2mDownload Start\x1b[0m\n
workdir $download_dir
getfile called.lzh
ifflag 1 gosub downloadfehler
protmsg ($time) DOWNLOAD ENDE\n
protmsg ($time) INFO: $fileinfo\n
msg ($time) INFO: $fileinfo\n

msg ($time) Netcall Ende\n
protmsg ($time) Netcall Ende\n
end


label modeminit
msg Sende Modeminit\n
send ATZ\r\n
wait $error,$ok
if $error goto init_error
msg \x1b[2mModem init ok\x1b[0m\n
return


label init_error
msg Modeminit fehlerhaft !\n
protmsg ($time) Modeminit Fehlerhaft !\n
end


label  zielverzeichnis räumen
workdir $upload_dir
system delete caller.lha
system delete #?.brt
system delete #?.prv
system lha a caller.lha puffer
return

label showconfig
```

```
msg   ----------------------------------------\n
msg   .           Konfiguration\n\n
msg   Programmstart :$date $time\n
msg   ConfigFile   = $config\n
msg   Scriptfile   = $script\n\n
msg   Device       = $device\n
msg   Unit         = $unit\n
msg   Baud         = $baud\n
msg   Serialbuffer = $buffersize\n
msg   Parameter    = $databits Datenbits\n
msg   .              $stopbits Stopbits\n
msg   Parität      = $parity\n
msg   Xpr-Library  = $xprlib\n
msg   Xpr-Optionen = $xpropts\n
msg   ----------------------------------------\n\n
return



label startprotokoll
protocol $protocolfile
protmsg \n
protmsg ----------------------------------\n
protmsg *          NetCall               *\n
protmsg * $date $time (Scriptstart)\n
return

label uploadfehler
msg   Upload FEHLER !\n
protmsg ($time) Upload FEHLER !\n
return


label downloadfehler
msg   Download FEHLER !\n
protmsg ($time) Download FEHLER !\n
return



------------------------------------------------------------------------
```

1.8 Bugs & anderes

```
Programmiert wurde UCall auf einem A4000/40 mit 10MB RAM, 650MB Platte unter
Kick 39.106/WB39.29 mit SAS/C 6.51.

Späschl Thänx gehen an Broken_Systems%mab@wsb.freinet.de
                       Tachy@wsb.freinet.de
                       und Technics%mab@wsb.freinet.de
für die geistig-moralische Unterstützung und Tests ;)

Wenn Sie Fragen oder Anregungen haben wenden Sie sich an
```

lunqual%mab@wsb.freinet.de
lunqual@crazy.freinet.de

Sackpostadresse:

Karlheinz Klingbeil
Elzstrasse 42

D-79261 Gutach
(Deutschland)