Hilfe zur Makrosprache der T-Online Software

Sie erhalten ...

... eine Einführung in die Makrosprache.

... eine Alphabetische Liste der Makro-Befehle.

... eine nach Themen geordnete Liste der Makro-Befehle.

... die Hilfe zur Bedienung des Makro-Editors.

Durch Drücken der Schaltfläche **[Inhalt]** in der Menüleiste dieser Hilfe kommen Sie jederzeit zu dieser Übersichtsseite zurück.

Für Hilfe zur Hilfe bitte Funktionstaste [F1] drücken.

Einführung

Allgemeine Informationen

Die T-Online Software verfügt über eine Makrosprache, mit deren Hilfe Sie häufig benötigte Vorgänge automatisieren können. Alle Tätigkeiten, die Sie innerhalb des T-Online Systems durchführen, wie zum Beispiel das Anwählen einer Seite oder das Herunterladen von Telesoftware, läßt sich mit Hilfe eines Makros automatisieren.

Ein Makro ist in diesem Zusammenhang nichts anderes als eine einfache Befehlsfolge, die beim Aufruf des Makros ausgeführt wird. Die Ausführung des Makros hat den gleichen Effekt, als würden Sie die einzelnen Schritte über die Tastatur oder mit der Maus durchführen. Allerdings stehen innerhalb des Makros zusätzliche Möglichkeiten, wie zum Beispiel das Arbeiten mit variablen Parametern, zur Verfügung.

Wie werden Makros erstellt und ausgeführt?

Alle T-Online Makros werden mit Hilfe des in der T-Online Software integrierten Editors erstellt und in Form einer Textdatei abgespeichert. Sie können Makros daher auch außerhalb der T-Online Software erstellen und bearbeiten. Durch die direkte Anbindung des Editors an die T-Online Software können aus dem Editor heraus Makros gestartet werden.

Die T-Online Software enthält in seinem T-Online Menü einen Eintrag mit dem Namen **Makro**, nach dessen Auswahl ein Untermenü mit folgenden Einträgen erscheint:

- **Makro ausführen**
- **Makro abbrechen**
- **Makro neu erstellen**
- **Makro editieren**
- **Makro aufzeichnen**
- **Makro aufzeichnen beenden**
- **Makrohilfe**

Die Makrosprache der T-Online Software ist relativ einfach strukturiert. Sie ist mit der Stapelsprache von MS-DOS vergleichbar, wenngleich sie aber eine Reihe zusätzlicher Befehle beinhaltet.

Hinweis:

Jeder Makrobefehl wird anhand eines kommentierten Beispiels veranschaulicht. Fast alle Beispiele sind weiterhin als fertige Makros (<Makroname>.mkr) im Makroverzeichnis verfügbar und können direkt ausgeführt werden.

Makro ausführen

Die Ausführung des Befehls **Makro ausführen** startet den Inhalt einer Makrodatei (Erweiterung .mkr).

Makro abbrechen

Mit dem Befehl **Makro abbrechen** stoppen Sie die Ausführung des aktiven Makros.

Makro neu erstellen

Mit dem Befehl **Makro neu erstellen** können Sie den Makroeditor starten, um ein neues Makro zu erstellen.

Makro editieren

Mit dem Befehl **Makro editieren** starten Sie den Makroeditor und öffnen ein bereits vorhandenes Makro zur Ansicht bzw. zur weiteren Bearbeitung.

Makro aufzeichnen

Das Kommando **Makro aufzeichnen** ist besonders interessant, denn es erlaubt Ihnen eine beliebige Aktion innerhalb der T-Online Software in ein Makro umzuwandeln.

Nach Auswahl dieses Kommandos (was allerdings nur Online möglich ist) müssen Sie zunächst den Namen des neuen Makros eingeben (vergessen Sie die Erweiterung .**mkr** nicht). Anschließend verfolgt die T-Online Software alle Ihre Schritte und Mausklicke und wandelt diese in Makrobefehle um (das Aufzeichnen eines Makros wird durch einen entsprechenden Hinweis in der Statuszeile angezeigt)

Sind Sie mit einer Tätigkeit fertig, wird die Aufzeichnung des Makros über das Kommando **Makro aufzeichnen beenden** wieder beendet.

Makro aufzeichnen beenden

Das Kommando **Makro aufzeichnen beenden** stoppt die Aufzeichnung des Makros.

Makrohilfe

Über diesen Menüpunkt erhalten Sie diese Hilfe.

Variablen und Konstanten

Die Makro-Sprache arbeitet mit String-Variablen (Zeichenketten) und -Konstanten.

- Numerische Werte werden dezimal im Klartext im String abgelegt und können ein negatives Vorzeichen ("-") haben.
- Variablennamen können beliebig lang sein und dürfen aus Buchstaben, Ziffern und dem Unterstrich ("_") bestehen.
- Konstanten werden in Anführungszeichen eingeschlossen.
- Sonderzeichen innerhalb von Konstanten werden mit einem Backslash ("\") eingeleitet. Es sind folgende Sonderzeichen definiert:

\b	Sonderzeichen <backspace>
\c	T-Online-Steuerzeichen "HOME"
\d	T-Online-Steuerzeichen "DCT"
\f	Sonderzeichen <form feed>
\h	T-Online-Steuerzeichen "HOME"
\n	Sonderzeichen <new line>
\r	Sonderzeichen <carriage return>
\t	Sonderzeichen <tab>
\v	Sonderzeichen <vertical tab>
\\	Sonderzeichen <backslash>
num	Zeichen mit dem Zeichencode *num* (oktal, 1- bis 3-stellig)
\d*num*	Zeichen mit dem Zeichencode *num* (dezimal, 1- bis 3-stellig)
\x*num*	Zeichen mit dem Zeichencode *num* (hexadezimal, 2-stellig)
\a	'ä'
\A	'Ä'
\o	'ö'
\O	'Ö'
\u	'ü'
\U	'Ü'
\s	'ß'
\#	T-Online-Steuerzeichen "TER"
*	T-Online-Steuerzeichen "INI"
\.	T-Online-Steuerzeichen "DCT"

Reservierte Variablen

ARGC
: Diese Variable enthält die Anzahl der Aufrufargumente.

ARGV_0, ARGV_1, ARGV_2, ...
: In diesen Variablen werden die Aufrufargumente abgelegt. ARGV_0 enthält den Makronamen, ARGV_1 das erste Aufrufargument. Eine flexiblere Auswertung der Aufrufargumente ermöglicht der Befehl getargv.

CALLVAL
: Enthält den Rückgabewert des letzten Makros, das über Befehl call aufgerufen wurde.

DEBUG
: Der Wert "on" für diese Variable schaltet den Debug-Modus ein. Im Debug-Modus wird jeder ausgeführte Makro-Befehl im Protokollfenster angezeigt. Der Wert "off" schaltet den Debug-Modus aus.

ERRNO
: In der Variablen ERRNO steht der Fehlercode des zuletzt aufgetretenen Fehlers.

RETVAL
: Ein Makro, das über den Befehl call aufgerufen wurde, liefert den Wert dieser Variable an das aufrufende Makro zurück.

SDEBUG
: Der Wert "on" veranlaßt, daß alle Ausgaben, die ins Protokollfenster gehen, zusätzlich in der Statuszeile angezeigt werden. Um diesen Modus zu beenden, ist der Wert wieder auf "off" zu setzen.

FILETEXTMODE
: Die Variable FILETEXTMODE gibt an, ob Dateien im Text- oder Binärmode geöffnet werden. Im Textmode erfolgt eine automatische Umsetzung von CR/LF in LF. Default ist (aus Kompatibilitätsgründen) Binärmode.

  Syntaxbeispiel:
  set FILETEXTMODE = "on"
  set FILETEXTMODE = "off"

Befehls-Argumente

Die Befehle der Makrosprache haben als Argumente String-Konstanten oder -Variablen . Es gibt folgende Typen von Argumenten:

var - String-Variable

Für diesen Argumenttyp sind nur Variablen zugelassen. Er wird immer dann verwendet, wenn der Inhalt des Arguments durch den Befehl verändert wird, z.B. bei einer Zuweisung mit dem Befehl set.

str - Variable oder Konstante

Das Argument ist entweder eine Variable oder eine Konstante.

list - variable Argumentliste

Argument-Liste mit variabler Anzahl der Komponenten. Die Komponenten sind durch Kommata getrennt und dürfen jeweils eine Variable oder ein Konstante sein.

label - Sprungmarke

Das Argument bezeichnet ein Ziel für eine Verzweigung durch goto oder gosub, eine sogenannte Sprungmarke. Die Sprungmarke selbst wird definiert, indem die entsprechende Stelle mit Ihrem Namen und einem nachgestellten Doppelpunkt gekennzeichnet wird.

Sprungmarken dürfen nur an einer Stelle im Makro definiert werden, können aber von beliebig vielen Stellen aus angesprungen werden.

var - String-Variable

Für diesen Argumenttyp sind nur Variablen zugelassen. Er wird im dort verwendet, wo der Inhalt des Arguments durch den Befehl verändert wird, z.B. bei einer Zuweisung mit dem Befehl set.

Siehe auch:

Befehls-Argumente

str - Variable oder Konstante

Das Argument ist entweder eine Variable oder eine Konstante.

Siehe auch:

Befehls-Argumente

list - variable Argumentliste

Argument-Liste mit variabler Anzahl der Komponenten. Die Komponenten sind durch Kommata getrennt und dürfen jeweils eine Variable oder ein Konstante sein.

Siehe auch:

Befehls-Argumente

label - Sprungmarke

Das Argument bezeichnet ein Ziel für eine Verzweigung durch goto oder gosub, eine sogenannte Sprungmarke. Die Sprungmarke selbst wird definiert, indem die entsprechende Stelle mit Ihrem Namen und einem nachgestellten Doppelpunkt gekennzeichnet wird.

Sprungmarken dürfen nur an einer Stelle im Makro definiert werden, können aber von beliebig vielen Stellen aus angesprungen werden.

Siehe auch:

Befehls-Argumente

Alphabetische Liste der Makro-Befehle

Thematische Liste der Makro-Befehle

Ablaufsteuerung

- call
- end
- gosub
- goto
- if
- ifnot
- ifless
- ifeqless
- ifgreater
- ifeqgreater
- return

Allgemeine Befehle

- set
- getargv

Automatischen Fehlerbehandlung

- onerror
- resume
- resumeat

CEPT-Ausgabe

ceptprint
ceptwrite

Clipboard

- clipaddtext
- clipdelete
- clipgettext
- clipputtext

Decoder-Befehle

Dialogbefehle des Benutzers

Ein- und Ausgabe

- append
- iniread
- iniwrite
- read
- readline
- write

Numerische Operationen

- add
- and
- dec
- div
- inc
- mult
- mod
- or
- subtract
- xor

String-Befehle

Systembefehle

- sleep
- system

Timergesteuerte Decoderbefehle

waitdcttime

T-Online-Teilnehmerdaten

getbtxuser
setbtxuser

add <var A>, <str B>

Zu einer Variablen <A>, die einen numerischen Wert enthält, wird der ebenfalls numerische Wert des Arguments <B> addiert .

Ist der Wert von <A> oder der Wert von <B> kein numerischer Wert, wird eine Fehlermeldung ausgegeben.

Beispiel (add.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "12"
set B = "26"
set C = A

#     Addition durchfuehren
add C, B

#     Ergebnis ausgeben
debug A, " + ", B, " = ", C, "\n"
```

Siehe auch:

inc
dec
div
mod
mult
subtract

and <var A> = <str B>, <str C>

Führt eine bitweise UND-Operation (<B> AND <C>) durch.

Beispiel (And.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Flags = "129"
set Mask  = "1"

#     AND-Operation durchfuehren
and Result = Flags, Mask

#     Ergebnis ausgeben
debug Flags, " AND ", Mask, " = ", Result, "\n"
```

Siehe auch:

or
xor

append <str A> to <str B>, <str L>

Der Inhalt des Arguments <A> wird an eine existierende Datei mit dem Namen <B> angehängt. Wenn die Datei nicht existiert, wird sie erzeugt.

Oft ist es erwünscht, daß in der Datei Zeilenumbrüche erscheinen. Diese müssen dann in <A> als Sonderzeichen enthalten sein oder anschließend mit einem weiteren **append**-Befehl geschrieben werden.

Ist zusätzlich der Parameter <L> angegeben, beschränkt sich der Schreibvorgang auf die in dieser Variablen angegebene Anzahl von Zeichen.

Beispiel (append.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Dateimodus auf Text setzen
set FILETEXTMODE = "on"

#     Dateiname setzen
set Filename = "FILETEST.TXT"

#     Zeile im Textmodus anhaengen
#     Datei wird erzeugt, wenn sie noch nicht exisitert
#     "\n" wird nach "\r\n" konvertiert
append "Erste Textzeile\n" to Filename

#     Dateimodus auf Binaer umschalten
set FILETEXTMODE = "off"

#     Jetzt muss man "\r\n" schreiben (wegen Binaermodus)
append "Zweite Textzeile\r\n" to Filename

#     Hinweis ausgeben
debug "Datei ", Filename, " wurde geschrieben\n"
```

Siehe auch:

read
readline
write

call <list A>

Ein anderes Makro wird als Unterprogramm ausgeführt. Die Komponenten von <A> sind die Aufrufargumente des auszuführenden Makros, dabei ist die erste Komponente der Name des auszuführenden Makros.

Das aktuelle Makro wartet bis das andere Makro beendet ist und bekommt dessen Rückgabewert in der Variablen CALLVAL zurückgeliefert.

Beispiel (call.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"

#     Makroname holen
set Makroname = argv_0

#     CALL-Ebene holen, bei Aufruf ohne Parameter mit "0" vorbelegen
set Ebene = "1"

#     wenn argc kleiner als "2", dann wurde das Makro nicht von sich selbst aufgerufen
ifless argc than "2" goto NoPara

#     Makro wurde von sich selbst aufgerufen => Ebene holen und hochzaehlen
set Ebene = argv_1
inc Ebene
goto Start

#     Diesen Teil nur beim ersten Aufruf ausfuehren
NoPara:
debug "Makroname: ", argv_0, "\n"

#     Start-Hinweis ausgeben
Start:
debug "[", Ebene, "] Makro gestartet\n"

#     Irgendwann auch mal aufhoeren mit sich selbst aufrufen ...
ifgreater Ebene than "9" goto Fertig

#     Makro ruft sich selbst auf
call Makroname, Ebene

#     call-Befehl kehrt erst zurueck, wenn das aufgerufene Makro beendet ist
#     dann geht es hier weiter
debug "[", Ebene, "] Erhaltener Rueckgabewert: ", CALLVAL, "\n"

#     Makro beenden
Fertig:
set RETVAL = "0", Ebene, "0"
debug "[", Ebene, "] Setze Rueckgabewert: ", RETVAL, "\n"
debug "[", Ebene, "] Makro beendet\n"
```

Siehe auch:

end

RETVAL

ceptprint <str A>

Die aktuelle CEPT-Seite wird ausgedruckt. Der Parameter <A> gibt den Typ an. "G" steht für Grafikformat und "T" für Textformat. Hierbei muß beachtet werden, daß die Zeichen "G" und "T" in Anführungszeichen gesetzt werden.

Beispiel (Ceptprin.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     und ausdrucken
debug "CEPT-Seite wird im Grafikmodus gedruckt ...\n"
ceptprint "G"

debug "CEPT-Seite wird im Textmodus gedruckt ...\n"
ceptprint "T"
```

Siehe auch:

ceptwrite

ceptwrite <str A>, <str B>

Speichert die aktuelle CEPT-Seite als Text oder Grafik in die Datei <A>. Der Parameter <B> gibt den Typ an. "G" steht für Grafikformat und "T" für Textformat. Hierbei muß beachtet werden, daß die Zeichen G und T in Anführungszeichen gesetzt werden.

Bei Wahl des Textformats "T" wird die Datei innerhalb des T-Online Software-Verzeichnisses unter "Seiten" abgelegt, sonst unter "Makro".

Wenn kein Dateiname angeben wird, dann kann über die Dateiauswahlbox ein Dateiname gewählt werden. In diesem Fall steht der Pfad auf dem Windows-Verzeichnis.

Beispiel (Ceptwrit.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     und in Datei schreiben
debug "CEPT-Seite wird im Grafikmodus gespeichert ...\n"
ceptwrite "NULLSEIT.BMP", "G"

#     wird im Verzeichnis SEITEN abgelegt
debug "CEPT-Seite wird im Textmodus gespeichert ...\n"
ceptwrite "NULLSEIT.TXT", "T"

#     Dateiname ueber Dialog waehlen
debug "CEPT-Seite wird im Grafikmodus gespeichert ...\n"
ceptwrite "", "T"
```

Siehe auch:

ceptprint

charcode <var A> = <str B>

Der Zeichencode des ersten Zeichens im Argument <B> wird als numerischer Wert in der Variablen <A> abgelegt.

Ist <B> ein Leerstring, so wird der Wert "0" in <A> abgelegt.

Beispiel (Charcode.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Name = "Hugo Habicht"

#     Name in Puffer kopieren
set Buffer = Name
debug "Text: ", Name, "\nCodes: "

#     Schleife: bei jedem Durchlauf den Code des ersten Zeichen ausgeben
#     und dieses Zeichen dann loeschen
Loop:
charcode Code = Buffer
debug Code, " "
#     erstes Zeichen loeschen
strdelete Buffer, "0", "1"
ifnot Buffer == "" goto Loop

debug "\n"
```

Siehe auch:

hexstr
makechar

clipaddtext <list A>

Hängt den Inhalt von <A> an den bisherigen Inhalt der Zwischenablage an.

Ist die Zwischenablage leer oder enthält andere Elemente wie Text, so verhält sich **clipaddtext** so wie clipputtext .

Beispiel (Clipaddt.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

debug "Addresse von Hugo wird in die Zwischenablage kopiert...\n"

#     Zwischenablage loeschen
clipdelete

#     Adresse in Zwischenablage legen
clipaddtext "Hugo Habicht\n"
clipaddtext "Mohnweg 2\n"
clipaddtext "77777 Entenhausen\n", "Tel.: 0777777777\n"
```

Siehe auch:

clipdelete
clipgettext
clipputtext

clipdelete

Leert die Zwischenablage.

Beispiel (Clipdele.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Adresse in Zwischenablage legen
debug "Addresse von Hugo wird in die Zwischenablage kopiert...\n"
clipputtext "Hugo Habicht\nMohnweg 2\n77777 Entenhausen\nTel.: 0777777777\n"

#     Zwischenablage loeschen
debug "Zwischenablage wird geloescht...\n"
clipdelete

#     Zwischenablage lesen
debug "Zwischenablage wird gelesen (erzeugt Fehler, da leer) ...\n"
clipgettext Text
debug "Gelesen: ", Text, "\n"
```

Siehe auch:

clipaddtext
clipgettext
clipputtext

clipgettext <var A>

Kopiert den Inhalt der Zwischenablage in <A>.

Beispiel (Clipgett.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Adresse in Zwischenablage legen
debug "Addresse von Hugo wird in die Zwischenablage kopiert...\n"
clipputtext "Hugo Habicht\nMohnweg 2\n77777 Entenhausen\nTel.: 0777777777\n"

#     Zwischenablage lesen
debug "Zwischenablage wird gelesen ...\n"
clipgettext Text
debug "Gelesen:\n", Text, "\n"
```

Siehe auch.

clipaddtext
clipdelete
clipputtext

clipputtext <list A>

Schreibt den Inhalt von <A> in die Zwischenablage. Der bisherige Inhalt der Zwischenablage wird dabei überschrieben.

Beispiel (Clipputt.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Zwischenablage loeschen
clipdelete

#     Erste Adresse in Zwischenablage legen
debug "Addresse von Egon wird in die Zwischenablage kopiert...\n"
clipputtext "Egon Krautwickel\nSonnenstrasse 4\n88888 Ganshausen\nTel.: 0888888888\n"

#     Zweite Adresse in Zwischenablage legen, dadurch wird die erste ueberschrieben
debug "Addresse von Hugo wird in die Zwischenablage kopiert...\n"
clipputtext "Hugo Habicht\nMohnweg 2\n77777 Entenhausen\nTel.: 0777777777\n"

#     Zwischenablage lesen (enthaelt nur die zweite Adresse)
debug "Zwischenablage wird gelesen...\n"
clipgettext Text
debug "Gelesen:\n", Text, "\n"
```

Siehe auch:

clipaddtext
clipdelete
clipgettext

connect

Die T-Online Software baut die Verbindung zu T-Online auf und meldet den Benutzer an. Dabei werden die in der T-Online Software eingestellten Zugangsdaten verwendet. Erst nach Abschluß der Zugangsprozedur oder Feststellen eines Fehlers durch die T-Online Software läuft das Makro weiter.

Anschließend kann mit dem Befehl getstate festgestellt werden, ob die T-Online-Anmeldung erfolgreich war, um entsprechend zu reagieren.

Wenn die T-Online Software bereits online ist, hat connect keine Wirkung.

Beispiel (Connect.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Status anzeigen
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"

#     T-Online-Verbindung herstellen
connect

#     Status anzeigen
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"
```

Siehe auch:

connectas
disconnect
getstate

connectas <str A>

Nach Aufruf dieses Befehls wird mit den in <A> übergebenen Parametern eine Anwahl durchgeführt. Die zu diesem Zeitpunkt bereits in der T-Online Software eingetragenen Zugangsdaten werden nach dem Aufruf wieder auf die vorherige Einstellung zurückgesetzt.

Ist die T-Online Software bereits online, hat dieser Befehl keine Auswirkung.

Die Syntax lautet:

connectas "<Anschlußkennung> <Teilnehmernummer> <Mitbenutzerzusatz> <Kennwort>"

Hinweise:

Dieser Befehl wirkt sich nur auf den Classic-Login aus, der IP-Login erfolgt mit dem aktuellen Zugangsdatenprofil.

Es ist eine partielle Übernahme der Einstellungen der T-Online Software möglich, z.B. in folgender Form: connectas "* * 100 Kennwort" oder connectas "* * * Kennwort"

Beispiel (Connecta.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Status anzeigen
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"

#     Teilnehmerdaten fuer Zugang setzen
debug "ACHTUNG!!! Dummy-Daten fuer T-Online-Zugang\n"
set Kennung    = "000000000000"
set TlnNr      = "01111122222"
set MBZ         = "1"
set Kennwort = "12345678"
set UserData = Kennung, " ", TlnNr, " ", MBZ, " ", Kennwort

#     T-Online-Verbindung herstellen
connectas UserData

#     Status anzeigen
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"
```

Siehe auch:

connect
disconnect
getstate

debug <list A>

Ist die Variable DEBUG auf "on" gesetzt, werden die Komponenten der Argumentliste <A> in der angegebenen Reihenfolge zu einer Meldung zusammengesetzt und im Protokollfenster ausgegeben. Abhängig vom Zustand der Variablen SDEBUG wird die Meldung, unabhängig vom Wert der Variablen DEBUG, zusätzlich in der Statuszeile angegeben.

Beispiel (Debug.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

# Mehrere Zeilen ausgeben
debug "Zeile1\nZeile2\nZeile3\n"
debug "Zei"
debug "le4"
debug "\n"
debug "Zeile5", "\n", "Zeile6", "\n"
```

Siehe auch:

info

dec <var A>

Der numerische Wert in <A> wird um den Wert 1 erniedrigt.

Ist <A> kein numerischer Wert, wird eine Fehlermeldung ausgegeben. Dieser Befehl wird häufig bei der Programmierung von Schleifen eingesetzt.

Beispiel (Dec.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "12"
debug "Alter Wert: A = ", A, "\n"

#     Operation durchfuehren, Wert von A wird um eins kleiner
debug "Befehl dec A wird ausgefuehrt\n"
dec A

#     Ergebnis ausgeben
debug "Neuer Wert: A = ", A, "\n"
```

Siehe auch:

add
div
inc
mod
mult
substract

delblanks <var A>

In der Variablen <A> werden alle Leerzeichen am Anfang sowie am Ende entfernt. Leerzeichen in der Mitte, d.h. zwischen anderen Zeichen, werden nicht entfernt.

Beispiel (Delblank.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text1 = " Hugo Habicht       "
set Text2 = "    Hugo Habicht     "
set Text3 = "     Hugo Habicht    "
set Text4 = "      Hugo Habicht "

#     Texte ausgeben
debug "Text1 = <", Text1, ">\n"
debug "Text2 = <", Text2, ">\n"
debug "Text3 = <", Text3, ">\n"
debug "Text4 = <", Text4, ">\n"

#     Leerzeichen am Anfang und am Ende der Texte entfernen
debug "Leerzeichen am Anfang und Ende werden entfernt ...\n"
delblanks Text1
delblanks Text2
delblanks Text3
delblanks Text4

#     Neue Texte ausgeben
debug "Text1 = <", Text1, ">\n"
debug "Text2 = <", Text2, ">\n"
debug "Text3 = <", Text3, ">\n"
debug "Text4 = <", Text4, ">\n"
```

Siehe auch:

strdelete

disconnect

Die Verbindung zu T-Online wird beendet. Besteht zur Zeit des Aufrufs keine T-Online-Verbindung, ist der Befehl wirkungslos.

Beispiel (Disconne.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Status anzeigen
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"

#     Wenn T-Online-Verbindung besteht, trennen
if Status == "DISCONNECTED" goto Fertig
disconnect
getstate Status
debug "Status der T-Online-Software: " , Status, "\n"

Fertig:
```

Siehe auch:

connect
connectas
getstate

div <var A> = <str B>, <str C>

Dividiert den numerischen Wert des Arguments <B> durch den ebenfalls numerischen Wert <C> und speichert ihn in <A> ab.

Ist der Wert von <B> oder der Wert von <C> kein numerischer Wert, wird eine Fehlermeldung ausgegeben.

Beispiel (Div.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "12"
set B = "4"

#     Division durchfuehren
div C = A, B

#     Ergebnis ausgeben
debug A, " / ", B, " = ", C, "\n"
```

Siehe auch:

add
dec
Inc
mult
substract

end

Die Ausführung des aktuellen Makros wird beendet.

Wurde das Makro von einem anderen Makro aus aufgerufen, kann durch Setzen der Variable RETVAL ein Wert an dieses zurückgegeben werden, um z.B. Erfolg oder Mißerfolg der im aktuellen Makro durchgeführten Aktion zu melden.

Ein Makro endet auch ohne **end**, wenn das Makroende erreicht ist.

Beispiel (End.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Makro beenden
debug "Makro wird beendet\n"
end

#     Die folgenden Befehle werden nicht mehr ausgefuehrt
debug "Wenn diese Ausgabe erscheint, hat der end-Befehl versagt ...\n"
end
```

Siehe auch:

call

get <var A> = <str B>, <str C>, <str D>, <str E>

Von der momentan angezeigten T-Online-Seite wird ein rechteckiger Bereich ausgelesen und in der Variablen <A> abgelegt.

Position und Größe des auszulesenden Bereichs werden durch die Argumente <B> bis <E> in folgender Weise bestimmt:

<B> = X-Position obere linke Ecke (1..40)
<C> = Y-Position obere linke Ecke (1..24)
<D> = X-Position untere rechte Ecke (1..40)
<E> = Y-Position untere rechte Ecke (1..24)

Beispiel (Get.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Nullseite aufrufen
send "\*0\#"
waitdct

#    Eine Zeile vom CEPT-Bild lesen
#    Puffer enthaelt KEIN LF-Zeichen am Ende
debug "Zeile 6 wird gelesen:\n<"
get Buffer = "1", "6", "40", "6"
debug Buffer, ">\n"

#    Mehrere Zeilen holen, ACHTUNG!!!
#    Zeilenvorschubzeichen (\n) auch von der letzten Zeile
#    sind jetzt im Puffer enthalten
debug "Zeilen 7-9 werden gelesen:\n<"
get Buffer = "1", "7", "40", "9"
debug Buffer, ">\n"

#    Ausschnitt holen, ACHTUNG!!!
#    Zeilenvorschubzeichen (\n) auch von der letzten Zeile
#    sind jetzt im Puffer enthalten
debug "Zeilen 7-9, Spalten 4-20 werden gelesen:\n<"
get Buffer = "4", "7", "20", "9"
debug Buffer, ">\n"
```

Siehe auch:

Decoder-Befehle

getargv <var A> = <str B>

Der Wert eines Aufrufarguments (argv_0, argv_1, ...) wird in die Variable <A> kopiert. Das Argument <B> gibt die Nummer des Aufrufarguments an. Ein Aufruf mit dem Wert "0" für <B> liefert argv_0.

Mit diesem Befehl ist eine flexiblere Auswertung der Aufrufargumente als über fixe Variablennamen möglich, z.B. in einer Schleife. Ein nicht-numerischer Wert für <B> verursacht einen Fehler.

Beispiel (Getargv.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"

#     Makroname holen
set Makroname = argv_0

#     wenn argc kleiner als "2", dann wurde das Makro nicht von
#     sich selbst aufgerufen
ifless argc than "2" goto NoPara

#     Makro wurde von sich selbst aufgerufen => Ebene holen und hochzaehlen
#     Alle Argumente ausgeben
set Count = "0"

#     Schleife, bei jedem Durchlauf ein Argument ausgeben
Loop:
getargv Arg = Count
debug "argv_", Count, " = <", Arg, ">\n"
inc Count
ifless Count than argc goto Loop
end

#     Diesen Teil nur beim ersten Aufruf ausfuehren
NoPara:
debug "Makroname: ", argv_0, "\n"

#     Makro ruft sich selbst auf
debug "Makro ruft sich selber auf ...\n"
call Makroname, "Hallo", "Name", "0777788889999"
end
```

Siehe auch:

Reservierte Variablen

getbkz <var A>

Weist der Variablen <A> die aktuelle Bereichskennzahl zu. Wenn die BKZ unbekannt ist, erhält <A> den Wert "0".

Hinweis: Ohne BKZ-Wechsel in den eigenen Bereich, ist direkt nach der Anwahl die BKZ unbekannt.

Beispiel (Getbkz.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     T-Online-Verbindung trennen
disconnect
#     Nullseite aufrufen
send "\*0\#"
waitdct

#     BKZ holen, ist ggf. noch nicht bekannt (dann 0)
getbkz BKZ
debug "Aktuelle Bereichskennzahl: ", BKZ, "\n"

#     BKZ-Wechsel in eigenen Bereich durchfuehren
send "\*78\#"
waitdct
send "\#"
waitdct
send "19"
waitdct
send "\#"
waitdct

#     BKZ holen, ist jetzt in jedem Fall bekannt
getbkz BKZ
debug "Aktuelle Bereichskennzahl: ", BKZ, "\n"
```

Siehe auch:

Decoder-Befehle

getbtxuser <var A>

Der Variablen <A> werden die in der T-Online Software eingestellter Zugangsdaten bzw. der Wert OFFLINE zugewiesen. Ein manuelles Umloggen mittels *91# hat keine Auswirkung auf die Rückgabewerte.

Folgende Werte werden zurückgegeben:

Offline	"OFFLINE"
Offline Gastzugang	"GAST"
Online Containerzugang	"CONTAINER"
Online Registrierter Teilnehmer	"<Teilnehmernummer>-<Mitbenutzerzusatz>"

Beispiel (Getbtxus.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     T-Online-Nutzer im Offline-Zustand holen
disconnect
getbtxuser User
debug "Aktueller T-Online-Nutzer: ", User, "\n"

#     Nullseite aufrufen (geht online)
send "\*0\#"
waitdct

#     T-Online-Nutzer holen
getbtxuser User
debug "Aktueller T-Online-Nutzer: ", User, "\n"
end
```

Siehe auch:

Decoder-Befehle

getcurpos <var A>, <var B>

Weist der Variablen <A> die aktuelle Spalte (1..40 bzw. 1..80) und <B> die aktuelle Zeile (1..24) des Cursors zu.

Beispiel (Getcurpo.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     Eine Zeile vom CEPT-Bild lesen
#     Puffer enthaelt KEIN LF-Zeichen am Ende
getcurpos X, Y
debug "Cursorposition:\n"
debug "X-Position = ", X, "\nY-Position = ", Y, "\n"
```

Siehe auch:

Decoder-Befehle

getpagenr <var A>

Durch diesen Befehl wird die aktuelle Seitennummer ausgelesen. Es wird nur die reine Seitennummer übergeben, also bei "19104670a" nur die Ziffernfolge "19104670".

Beispiel (Getpagen.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     "Neu in T-Online" aufrufen
send "70"
waitdct

#     Seitennummer holen
getpagenr Seitennr
debug "Aktuelle Seitennummer: ", Seitennr, "\n"
```

Siehe auch:

Decoder-Befehle

getsh <var A>

Liest die SH-Meldung aus.

Beispiel (Getsh.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     BKZ-Wechsel in eigenen Bereich durchfuehren
debug "BKZ-Wechsel in eigenen Bereich wird durchgefuehrt\n"
send "\*78\#"
waitdct

#     Eigenen Bereich bestaetigen
send "\#"
waitdct

#     Pruefung, ob Abfrage fuer BKZ-Wechsel da
getsh Code
ifnot Code == "1A349" goto Fehler
send "19"
waitdct
send "\#"
waitdct
debug "BKZ-Wechsel durchgefuehrt\n"
end

Fehler:
debug "Fehler im Ablauf"
end
```

Siehe auch:

Decoder-Befehle

getstate <var A>

Holt den Status der T-Online Software in die Variable <A>.

Besteht eine T-Online-Verbindung, ist dieser Wert "CONNECTED", im anderen Fall "DISCONNECTED".

Beispiel (Getstate.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     T-Online-Verbindung ggf. trennen
disconnect

#     Online-Status holen
getstate Status
debug "Online-Status: ", Status, "\n"

#     Nullseite aufrufen
debug "Aufruf der Nullseite wird durchgefuehrt ...\n"
send "\*0\#"
waitdct

#     Online-Status holen
getstate Status
debug "Online-Status: ", Status, "\n"
```

Siehe auch:

Decoder-Befehle

getstateext <var A>

Durch diesen Befehl wird der erweiterte Decoderstatus ausgelesen. Der Inhalt der Variaben <A> ist als Bitfeld zu interpretieren. Bitte beachten Sie, daß einige Bits nur in Abhängigkeit von anderen eine sinnvolle Bedeutung haben.

Dieser Befehl ist eignet sich in erster Linie für die Einbindung in den T-Online Application Access. T-Online Application Access ist eine einfach zu bedienende Nachrichtenschnittstelle, die anderen Programmen Zugriffe auf T-Online ermöglicht. T-Online Application Access ist wie die T-Online Makrosprache ein Bestandteil der T-Online Software.

Hinweis: Zur Auswertung des Rüchgabewertes kann der Befehl "and" (bitweise UND-Operation) eingesetzt werden.

Bit	Bedeutung wenn gesetzt	Bedeutung wenn nicht gesetzt
1	Online	Offline
2	VT100	CEPT
4	KIT aktiv	kein KIT
8	KIT suspended	kein KIT
16	KIT native	KIT pagebased
32	Externer Rechner	Telekom-Rechner
64	ER mit X.29	ER mit EHKP
128	Internet aktiv	kein Internet
256	Interner eMail-Client aktiv	kein eMail
512	Makro aktiv	kein Makro

Beispiel (Getstatx.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"

debug "Makroname: ", argv_0, "\n"

#     Erweiterten Status holen und einige Zustandsflags anzeigen
getstateext Status
and OnlineFlag = Status, "1"
and VT100Flag     = Status, "2"
and KITFlag = Status, "4"
and ERFlag = Status, "32"
and MakroFlag = Status, "512"

debug "OnlineFlag = ", OnlineFlag, "\n"
debug "VT100Flag = ", VT100Flag, "\n"
debug "KITFlag = ", KITFlag, "\n"
debug "ERFlag = ", ERFlag, "\n"
debug "MakroFlag =", MakroFlag, "\n"

set JA = "JA"
set NEIN = "NEIN"

set Info = "Decoderinformationen:\n\nOnline: "
set Flag = NEIN
if OnlineFlag == "0" goto Weiter1
set Flag = JA
```

```
Weiter1:
strcat Info, Flag, "\nVT100-Modus:"
set Flag = NEIN
if VT100Flag == "0" goto Weiter2
set Flag = JA

Weiter2:
strcat Info, Flag, "\nKIT-Modus:"
set Flag = NEIN
if KITFlag == "0" goto Weiter3
set Flag = JA

Weiter3:
strcat Info, Flag, "\nER-Verbindung:"
set Flag = NEIN
if ERFlag == "0" goto Weiter4
set Flag = JA

Weiter4:
strcat Info, Flag, "\nMakro aktiv:"
set Flag = NEIN
if MakroFlag == "0" goto Weiter2
set Flag = JA

Weiter5:
strcat Info, Flag, "\n"

messagebox Antwort: "Hinweis", Info, "OK"
```

Siehe auch:

Decoder-Befehle

gettrans <var A>

Liest den zuletzt empfangenen transparenten Datenblock aus und schreibt ihn in die Variable <A>.

Zuvor muß durch einen Aufruf entsprechender Seiten der Empfang veranlaßt worden sein.

Beispiel (Gettrans.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Filename = "TSW.BIN"

#     Letzten empfangenen Block mit transparenten Daten holen
gettrans Data
strlen Len of Data
debug Len, " Bytes gelesen\n"

#     und in Datei schreiben
write Data to Filename
```

gettype <var A>

Diese Funktion liefert den Typ des Decoders in <A> zurück.

Beispiel (Gettype.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Decoder-Typ holen
gettype Type
debug "Decodertyp: ", Type, "\n"
```

Siehe auch:

Decoder-Befehle

getversion <var A>

Übergibt die Versionsnummer der T-Online Software an <A> (z.B. bei einer Versionsnummer 2.0 ist der Wert von <A> "20").

Beispiel (Getversi.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Decoder-Version holen
getversion Version
debug "Decoder-Version: ", Version, "\n"
```

Siehe auch:

Decoder-Befehle

gosub <label A>

Ruft eine Unterfunktion im gleichen Makro auf, die durch die Sprungmarke <A> gekennzeichnet ist.

Eine Unterfunktion endet mit dem Befehl <return>. Das Makro wird danach mit dem auf die <gosub>-Anweisung folgenden Befehl fortgesetzt.

Beispiel (Gosub.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     CALL-Ebene holen, bei Aufruf ohne Parameter mit "0" vorbelegen
set Ebene = "0"
gosub SubFunc
end

#     ========================================
#     Unterfunktion, wird mit gosub aufgerufen
SubFunc:
inc Ebene
debug "[", Ebene, "] Unterfunktion \"SubFunc\" aufgerufen\n"

#     Irgendwann auch mal aufhoerhren mit Selbstaufruf
ifgreater Ebene than "9" goto Fertig

#     Unterunfktion ruft sich selber auf
gosub SubFunc

Fertig:
debug "[", Ebene, "] Unterfunktion \"SubFunc\" beendet\n"
dec Ebene
return
```

Siehe auch:

goto
return

goto <label A>

Setzt die Makro-Ausführung an der durch die Sprungmarke <A> bezeichneten Stelle fort.

Beispiel (Goto.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

goto Beginn

Lauf1:
goto Mitte

Lauf2:
goto Schluss

#     Schluss
Schluss:
debug "Schlussteil\n"
goto Ende

#     Mitte
Mitte:
debug "Mittelteil\n"
goto Lauf2

#     Beginn
Beginn:
debug "Beginn\n"
goto Lauf1


Ende:
debug "Makro beendet\n"
```

Siehe auch:

gosub
return

hexstr <var A> = <str B>

Der Hexadezimalcode des numerischen Wertes <B> wird in der Variablen <A> abgelegt.

Beispiel (Hexstr.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Name = "Hugo Habicht"

#     Name in Puffer kopieren
set Buffer = Name
debug "Text: ", Name, "\nCodes: "

#     Schleife: bei jedem Durchlauf den Code des ersten Zeichen ausgeben
#     und dieses Zeichen dann loeschen
Loop:
charcode Code = Buffer
hexstr HexCode = Code
debug HexCode, " "
#     erstes Zeichen loeschen
strdelete Buffer, "0", "1"
ifnot Buffer == "" goto Loop

debug "\n"
```

Siehe auch:

charcode
makechar

if - Bedingte Verzweigungen

if <str A> == <str B> goto <label C>

ifnot <str A> == <str B> goto <label C>

ifless <str A> than <str B> goto <label C>

ifeqless <str A> than <str B> goto <label C>

ifgreater <str A> than <str B> goto <label C>

ifeqgreater <str A> than <str B> goto <label C>

Die Argumente <A> und <B> werden verglichen. In Abhängigkeit vom Ergebnis des Vergleichs wird die Programmausführung an der durch die Sprungmarke <C> bezeichneten Stelle fortgesetzt. Im einzelnen führen die Befehle die Verzweigung unter folgender Bedingung aus:

if	Die Werte von <A> und <B> sind identisch.
ifnot	Die Werte von <A> und <B> sind nicht identisch.
ifless	Der Wert von <A> ist (numerisch) kleiner als der Wert von <B>.
ifeqless	Der Wert von <A> ist (numerisch) kleiner oder gleich dem Wert von <B>.
ifgreater	Der Wert von <A> ist (numerisch) größer als der Wert von <B>.
ifeqgreater	Der Wert von <A> ist (numerisch) größer oder gleich dem Wert von <B>.

Beispiel (If.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Name1 = "Hugo Habicht"
set Name2 = "Egon Unkraut"
debug "Name1 = ", Name1, "\n"
debug "Name2 = ", Name2, "\n"

#     Sprung wird nicht ausgefuehrt, da Bedingung nicht erfuellt
if Name1 == Name2 goto Ende
debug "Name1 != Name2\n"

#     Sprung wird nicht ausgefuehrt, da Bedingung nicht erfuellt
ifnot Name1 == "Hugo Habicht" goto Ende
debug "Name1 == Hugo Habicht\n"

set Zahl1 = "5"
set Zahl2 = "7"
debug "Zahl1 = ", Zahl1, "\n"
debug "Zahl2 = ", Zahl2, "\n"

#     Sprung wird nicht ausgefuehrt, da Bedingung nicht erfuellt
ifgreater Zahl1 than Zahl2 goto Ende
debug "Zahl1 <= Zahl2\n"

#     Sprung wird nicht ausgefuehrt, da Bedingung nicht erfuellt
ifless Zahl2 than Zahl1 goto Ende
debug "Zahl2 >= Zahl1\n"
```

```
#     Sprung nicht ausgefuehrt, Bedingung ist erfuellt
ifeqless Zahl1 than Zahl2 goto Ende
debug "Zahl1 > Zahl2\n"

#     Diese Befehle werden gar nicht mehr ausgefuehrt
ifeqgreater Zahl2 than Zahl1 goto Ende
debug "Zahl2 < Zahl1\n"

Ende:
debug "Makro beendet\n"
end
```

inc <var A>

Der numerische Wert in <A> wird um den Wert 1 erhöht.

Ist <A> kein numerischer Wert, wird eine Fehlermeldung ausgegeben. Dieser Befehl wird häufig bei der Programmierung von Schleifen eingesetzt.

Beispiel (Inc.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "12"
debug "Alter Wert: A = ", A, "\n"

#     Operation durchfuehren, Wert von A wird um eins kleiner
debug "Befehl inc A wird ausgefuehrt\n"
inc A

#     Ergebnis ausgeben
debug "Neuer Wert: A = ", A, "\n"
```

Siehe auch:

add
dec
div
mod
mult
substract

info <list A>

Verbindet alle Elemente der Liste <A> zu einem String und übergibt diesen der Benutzerausgabe.

Die Ausgabe des Makrobefehls erfolgt in einem Protokollfenster, welches sich nur öffnet, wenn die Befehlsfolge **set PROTWIN = "on"** vorgeschaltet wird.

Beispiel (Info.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

# Mehrere Zeilen ausgeben
info "Zeile1\nZeile2\nZeile3\n"
info "Zei"
info "le4"
info "\n"
info "Zeile5", "\n", "Zeile6", "\n"
```

Siehe auch:

debug

iniread <var A> = <str B>, <str C>, <str D>, <str E>

Liest aus der INI-Datei <B> im Abschnitt <C> den Wert des Eintrags mit dem Namen <D> und weist diesen <A> zu. Wird kein Eintrag gefunden, erhält <A> den Defaultwert <E>.

INI-Dateien ohne Pfadangabe beziehen sich auf das Unterverzeichnis DATA im T-Online Software-Verzeichnis. Ist der Name der INI-Datei "WIN.INI", wird die WIN.INI im Windows-Verzeichnis verwendet.

Beispiel (Iniread.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Dateiname normalerweise im DATA-Verzeichnis,
#    soll aber im MAKRO-Verzeichnis sein
set Ininame = "..\\MAKRO\\INIREAD.INI"
set Section = "Optionen"
set Entry    = "Name"
set Default = ""

#    Eintrag aus INI-Datei lesen
#    INI-Datei muss im Makroverzeichnis liegen und INIREAD.INI heissen
#    folgender Beispieleintrag wuerde gelesen:
#
#    [Optionen]
#    Name = Hugo Habicht
#
iniread Value = Ininame, Section, Entry, Default
debug "Gelesener Eintrag:\n", Entry, "=", Value, "\n"

ifnot Default == Value goto Ende
debug "Eintrag ist nicht vorhanden\n"
#    ... oder er hat zufaellig den gleichen Wert wie Default ...

Ende:
end
```

Siehe auch:

iniwrite

iniwrite <str A>, <str B>, <str C>, <str D>

Schreibt in die INI-Datei <A> im Abschnitt <B> einen Eintrag <C> mit dem Wert <D>.

Wird <D> weggelassen, wird der Eintrag gelöscht. Wird zusätzlich noch <C> weggelassen, wird der ganze Abschnitt <B> gelöscht.

Beispiel (Iniwrite.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Dateiname normalerweise im DATA-Verzeichnis,
#     soll aber im MAKRO-Verzeichnis sein
set Ininame = "..\\MAKRO\\INIREAD.INI"
set Section = "Optionen"
set Entry = "Name"
set Default = ""
set Name = "Hugo Habicht"

# Eintrag in INI-Datei schreiben
debug "Schreibe Name in INI-Datei ...\n"
iniwrite Ininame, Section, Entry, Name

#     INI-Datei lesen und ausgeben
read Inidata from Ininame
debug Inidata
debug "\n==============================\n"

# Eintrag aus INI-Datei loeschen
debug "Loesche Name aus INI-Datei ...\n"
iniwrite Ininame, Section, Entry

#     INI-Datei lesen und ausgeben
read Inidata from Ininame
debug Inidata
debug "\n==============================\n"

# Section aus INI-Datei loeschen
debug "Loesche Section aus INI-Datei ...\n"
iniwrite Ininame, Section

#     INI-Datei lesen und ausgeben
read Inidata from Ininame

debug Inidata
debug "\n==============================\n"
```

Siehe auch:

iniread

input <var A>: <str B>, <str C>

Der Benutzer wird mit dem Text in <B> in einer Dialogbox zu einer Eingabe aufgefordert. Die Benutzer-Eingabe steht anschließend in <A> zur Verfügung. Das Eingabefeld wird mit dem Text in <C> vorbelegt.

Enthält <C> den Text "*NO ECHO*", so ist die Eingabe nicht sichtbar (z.B. für die Eingabe eines Paßworts). Das Eingabefeld ist in diesem Fall nicht vorbelegt.

Beispiel (Input.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Benutzer nach dem Namen fragen
#     mit Vorbelegung "Hugo"
input Name : "Bitte geben Sie Ihren Namen ein", "Hugo"
debug "Hallo ", Name, ", wie geht's ?\n"
```

Siehe auch:

messagebox

makechar <var A> = <str B>

Der Zeichencode des numerischen Wertes im Argument <B> wird in der Variablen <A> abgelegt.

Beispiel (Makechar.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

debug "Zeichentabelle Codes 32 - 255\n"

set Code = "32"

#     Schleife: Alle Zeichen der Codes 32- 255 ausgeben
#     Pro DUrchlauf ein Zeichen
Loop:
makechar Char = Code
debug Char, "    "
mod Rest = Code, "16"

#     Alle 16 Zeichen eine neue Zeile beginnen
ifnot Rest == "0" goto Weiter
debug "\n"
Weiter:
inc Code
ifless Code than "256" goto Loop

debug "\n"
```

Siehe auch:

charcode
hexstr

messagebox <var A>: <str B>, <str C>, <str D>

Öffnet eine Message-Box und gibt den Text in <C> darin aus. Der Inhalt von <B> wird als Titel der Box ausgegeben.

Es gibt insgesamt fünf Typen der Message-Box, die sich sich in Art und Anzahl der Schaltflächen unterscheiden und durch das Argument <D> ausgewählt werden:

<D>	**Schaltflächen**
"OK"	Ok
"YESNO"	Ja, Nein
""	Ja, Nein, Abbrechen

Die Information, welche Taste gedrückt wurde, wird in <A> abgelegt. Mögliche Werte sind:

<A>	**Betätigte Schaltfläche**
"OK"	"Ok"
"YES"	"Ja" oder "Yes"
"NO"	"Nein" oder "No"
"CANCEL"	"Abbrechen" oder "Cancel"

Zusätzlich stehen zwei nicht-modale Message-Boxen zu Verfügung. Hierbei läuft das Makro ohne Betätigung einer Schaltfläche weiter.

<C>	**Schaltflächen**	
"INFO"	"keine"	Das Schliessen erfolgt durch Aufruf mit einem leeren Text oder Titel bzw. am Makroende.
"INFOOK"	Ok.	Klick auf Ok schließt lediglich die Messagebox. Ist das Fenster bereits geöffnet, wird der Text geändert und ggf. der Button Ok angezeigt oder ausgeblendet.

Beispiel (Messageb.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Hinweisbox waehrend der Anwahl anzeigen
messagebox Antwort: "Hinweis", "T-Online wird angewaehlt", "INFO"
disconnect
send "\*0\#"
waitdct

#    Hinweisbox wieder wegblenden
messagebox Antwort: "", "", "INFO"

#    Frage mit Auswahlmoeglichkeiten Ja / Nein / Abbrechen
#    Wenn nur Ja / Nein angeboten werden soll, muss der letzte
#    Parameter "YESNO" sein statt ""
messagebox Antwort : "Frage", "Fuehlen Sie sich gut heute ?", ""
```

```
debug "Antwort: ", Antwort, "\n"

#     Abbrechen gewaehlt?
if Antwort == "CANCEL" goto Ende

#     Nein gewaehlt?
if Antwort == "NO" goto Schlecht

#     Es wurde offensichtlich Ja gewaehlt
messagebox Antwort : "Meine Meinung", "Freut mich, dass Sie sich gut fuehlen\n", "OK"
goto Ende

Schlecht:
messagebox Antwort : "Meine Meinung", "Das ist aber schade ...\n", "OK"

Ende:
messagebox Antwort : "Hinweis", "Makro wird beendet", "OK"
debug "Antwort: ", Antwort, "\n"
```

Siehe auch:

input

mod <var A> = <str B>, <str C>

Speichert in <A> den Modulowert oder Restwert von <B> dividiert durch <C>.

Ist der Wert von <B> oder der Wert von <C> kein numerischer Wert, wird eine Fehlermeldung ausgegeben.

Beispiel (Mod.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "19"
set B = "7"

#     Operation durchfuehren
mod C = A, B

#     Ergebnis ausgeben
debug A, " modulo ", B, " = ", C, "\n"#
```

Siehe auch:

add
dec
div
inc
mult
substract

mult <var A> = <str B>, <str C>

Muliplziert den numerischen Wert des Arguments <B> mit dem ebenfalls numerischen Wert <C> und speichert ihn in <A> ab.

Ist der Wert von <B> oder der Wert von <C> kein numerischer Wert, wird eine Fehlermeldung ausgegeben.

Beispiel (Mult.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "3"
set B = "4"

#     Division durchfuehren
mult C = A, B

#     Ergebnis ausgeben
debug A, " * ", B, " = ", C, "\n"
```

Siehe auch:

add
dec
div
inc
mod
substract

offwin

Dieser Befehl veranlaßt die T-Online Software, das CEPT-Fenster auszublenden: Die TSW-Downloadanzeige wird ebenfalls unterdrückt. Bei Makroende wird das CEPT-Fenster wieder eingeblendet.

Hinweise:
Die T-Online Software muß vor dem Befehlsaufruf bereits online sein.
Der Befehl ist wirkungslos, wenn das CEPT-Fenster bereits unsichtbar ist.

Beispiel (Offwin.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

disconnect
#     CEPT-Fenster ausblenden => bringt jetzt nichts, da kein CEPT-Fenster aktiv (offline)
offwin

#     Nullseite aufrufen, das sieht man jetzt
send "\*0\#"
#     Auf Seitenende warten.
waitdct

#     Jetzt geht das Abschalten des CEPT-Bildes
offwin
#     Telekomangebot verdeckt aufrufen und 5 Sekunden warten
send "\*TELEKOM\#"
waitdct

messagebox Antwort: "Hinweis", "CEPT-Bild muesste jetzt ausgeblendet sein\n
5 Sekunden warten ...\n", "INFO"
sleep "5"

#     Bei Makroende wird das CEPT-Fenster automatisch wieder eingeschaltet
```

Siehe auch:

onwin

onerror <label A>

Installiert eine Fehlerbehandlungsroutine. Bei Auftreten eines Fehlers wird das angegebene Label angesprungen. In der Spezialvariablen ERRNO steht der Fehlercode des letzten Fehlers.

Beispiel (Onerror.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Fehlerbehandlung aktivieren
#    ACHTUNG!!! Nach Ausfuerung dieses Befehls wird
#    bei *ALLEN* Fehlern zum Label Fehler verzweigt !!!
#    z.B. bei Verwendung einer nichtinitialisierten Variable
onerror Fehler

goto Loop

#    ========================
#    Fehlerbehandlungsroutine
Fehler:
ifnot ERRNO == "8" goto Alle

#    Fehler mit ERRNO = 8 behandeln (nicht numerisches Argument)
messagebox Antwort: "Fehler in der Eingabe", "Der Wert ist nicht numerisch", "OK"
#    Eingabe wiederholen
resumeat Loop

#    Alle anderen Fehler behandeln, so ein Teil sollte immer vorhanden sein
#    fuer unvorhergesehene Faelle
Alle:
set Text = "Das Makro wird wegen eines Fehlers abgebrochen.\n Fehlercode = ", ERRNO
messagebox Antwort: "Fehler im Makro", Text, "OK"
# Fehlerbehandlungsroutine muss mit end, resume oder resumeat
# abgeschlossen werden, sonst knallt es beim naechsten Fehler
end
#    ========================

#   Eingabe einer Zahl zwischen 1 und 16
Loop:
input Zahl : "Bitte geben Sie eine Zahl zwischen 1 und 16 ein\n
(Versuchen Sie mal die Eingabe eines nichtnumerischen Wertes)", "16"
ifless Zahl than "1" goto Loop
ifgreater Zahl than "16" goto Loop

debug "Eingabe: ", Zahl, "\n"
end
```

Siehe auch:

resume
resumeat

onwin

Dieser Befehl veranlaßt die T-Online Software, das CEPT-Fenster anzuzeigen.

Der Befehl ist wirkungslos, wenn das CEPT-Fenster bereits sichtbar ist.

Beispiel (Onwin.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen, das sieht man jetzt
send "\*0\#"
#     Auf Seitenende warten.
waitdct

#     Schleife: Messagebox einblenden, in der das CEPT-Bild mit JA ein-
#     und mit NEIN ausgeschaltet werden kann. Abbrechen beendet die Schleife
Loop:
messagebox Antwort: "Frage", "Soll der CEPT-Bildschirm sichtbar sein ?\nJA: CEPT-Bild
einblenden\nNEIN: CEPT-Bild ausblenden\nAbbrechen: Makro beenden", ""
if Antwort == "CANCEL" goto Ende
if Antwort == "YES" goto Ja
offwin
goto Loop

Ja:
onwin
goto Loop

Ende:
End
```

Siehe auch:

offwin

or <var A> = <str B>, <str C>

Führt eine bitweise ODER-Operation (<B> OR <C>) durch.

Beispiel (Or.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Flags = "129"
set Mask   = "4"

#     OR-Operation durchfuehren
or Result = Flags, Mask

#     Ergebnis ausgeben
debug Flags, " OR ", Mask, " = ", Result, "\n"
```

Siehe auch:

and
xor

quit

Beendet die T-Online Software ohne Benachrichtigung.

Beispiel (Quit.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Sicherheitsabfrage
messagebox Antwort: "Frage", "Soll der Decoder wirklich beendet werden ?", "YESNO"

if Antwort == "NO" goto Ende

#     Decoder beenden
debug "Decoder wird beendet...\n"
quit

Ende:
end
```

Siehe auch:

disconnect

read <var A> from <str B>, <str L>

Liest den gesamten Inhalt der Datei mit dem Namen <B> unverändert in die Variable <A> ein. Ist die Datei nicht vorhanden, wird eine Fehlermeldung ausgegeben.

Ist zusätzlich der Parameter <L> angegeben, beschränkt sich der Lesevorgang auf die in dieser Varibalen angegebenen Anzahl von Zeichen.

Um anschließend in der Datei weiterlesen zu können, kann ein "*" anstelle des Dateinames angegeben werden. Bei Erreichen des Dateiendes wir ein Leerstring ("") zurückgeliefert.

Hinweis: Über die Variable FILETEXTMODE wird definiert, ob Dateien im Text- oder Binärmode geöffnet werden:

Beispiel (Read.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Erst mal eine Beispieldatei erzeugen
debug "Schreibe Testdatei ...\n"
set Filename = "READBSP.TXT"
write "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" to
Filename

#     Anzahl der Zeichen pro Zeile bei der Ausgabe
set Maxchar = "16"

#     Datei lesen und auf mehrere Zeilen verteilt ausgeben
read Data from Filename, Maxchar

Loop:
debug "Gelesen: <", Data, ">\n"
#     Weiterlesen
read Data from "*", Maxchar
#     Alles gelesen ? Dann Ende
if Data == "" goto Ende
goto Loop

Ende:
end
```

Siehe auch:

append
readline
write

readline <var A> from <str B>

Liest die erste Zeile der Datei mit dem Namen <B> in die Variable <A> ein. Ist die Datei nicht vorhanden, wird eine Fehlermeldung ausgegeben.

Um anschließend in der Datei weiterlesen zu können, kann ein "*" anstelle des Dateinames angegeben werden. Bei Erreichen des Dateiendes wird ein Leerstring ("") zurückgeliefert.

Hinweis: Über die Variable FILETEXTMODE wird definiert, ob Dateien im Text- oder Binärmode geöffnet werden:

Beispiel (Readline.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Erst mal eine Beispieldatei erzeugen
debug "Schreibe Testdatei ...\n"
set FILETEXTMODE = "on"
set Filename = "READLINE.TXT"
write "0123456789\nABCDEFGHIJKLMNOPQRSTUVWXYZ\nabcdefghijklmnopqrstuvwxyz\n" to
Filename

#     Erste Zeile lesen
readline Zeile from Filename

Loop:
debug "Gelesen: <", Zeile, ">\n"
#     Weiterlesen
readline Zeile from "*"
#     Alles gelesen ? Dann Ende
if Zeile == "" goto Ende
goto Loop

Ende:
end
```

Siehe auch:

append
read
write

resume

Setzt die Ausführung des Makros mit der nächsten Anweisung nach der Anweisung fort, die den Fehler auslöste.

Hinweis: Darf nur innerhalb der Fehlerbehandlungsroutine (d.h. nach Auftreten eines Laufzeitfehlers) verwendet werden.

Beispiel (Resume.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Fehlerbehandlung aktivieren
#    ACHTUNG!!! Nach Ausfuerung dieses Befehls wird
#    bei *ALLEN* Fehlern zum Label Fehler verzweigt !!!
#    z.B. bei Verwendung einer nichtinitialisierten Variable
onerror Fehler

#    nichtinitialisierte Variable Hugo
debug "Erzeuge Fehler: 1 Sekunde warten auf DCT\n"
waitdcttime "1"

debug "Makro laeuft weiter ...\n"

goto Ende

#    ========================
#    Fehlerbehandlungsroutine
Fehler:
set Text = "Es ist ein Fehler aufgetreten (Fehlercode = ", ERRNO, ")\nSoll das Makro trotzdem
fortgesetzt werden ?"
messagebox Antwort : "Fehler im Makro", Text, "YESNO"
if Antwort == "NO" goto Ende
resume

Ende:
debug "Makro beendet ...\n"
end
```

Siehe auch:

onerror
resumeat

resumeat <label A>

Setzt die Ausführung des Makros an dem angegebenen Label <A> fort.

Hinweis: Darf nur innerhalb der Fehlerbehandlungsroutine (d.h. nach Auftreten eines Laufzeitfehlers) verwendet werden.

Beispiel (Resumeat.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Fehlerbehandlung aktivieren
#    ACHTUNG!!! Nach Ausfuerung dieses Befehls wird
#    bei *ALLEN* Fehlern zum Label Fehler verzweigt !!!
#    z.B. bei Verwendung einer nichtinitialisierten Variable
onerror Fehler

set Filename = "RESUMEAT.TXT"
Read:
debug "Lese Datei ", Filename, " ...\n"
#    Wenn nicht vorhanden, wird Fehler mit ERRNO = "25" erzezugt
read Inhalt from Filename
debug "Gelesen: <", Inhalt, ">\n"

goto Ende

#    ========================
#    Fehlerbehandlungsroutine
Fehler:

ifnot ERRNO == "25" goto Alle
messagebox Antwort : "Fehler", "Fehler im Makro: Datei konnte nicht gelesen werden\nSoll sie
erzeugt und danach gelesen werden ?", "YESNO"
if Antwort == "NO" goto Ende
#    Datei erzeugen
write "Hugo Habicht" to Filename
#    Nochmal versuchen, zu lesen
resumeat Read

#    alle anderen Fehler
Alle:
debug "Fehler, Code = ", ERRNO, "\n"

Ende:
debug "Makro beendet\n"
end
```

Siehe auch:

onerror
resume

return

Beendet eine Unterfunktion. Die Programmausführung wird an der Stelle fortgesetzt, an der die Unterfunktion durch den Befehl gosub aufgerufen wurde.

Beispiel (Return.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

debug "Rufe Unterfunktion auf ...\n"
gosub SubFunc

#    Hier muesste ein end-Befehl stehen
#    Das Makro laeuft jetzt weiter in die Unterfunktion
#    hinein. Der return-Befehl dort erzeugt dann einen
#    Fehler.
messagebox Antwort: "Hinweis", "Jetzt kommt gleich ein (absichtlicher) Fehler ... siehe Makro-Quelltext\n", "OK"

#    Unterfunktion
SubFunc:
debug "Unterfunktion \"SubFunc\" aufgerufen\n"
debug "Unterfunktion \"SubFunc\" beendet\n"
return
```

Siehe auch:

gosub
goto

send <list A>

Die Komponenten des Arguments <A> werden zu einem String verbunden und anschließend an T-Online geschickt. Die Zeichen erscheinen dort so, als ob der Benutzer sie direkt über die Tastatur eingegeben hätte.

Beispiel (Send.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen, wenn keine Verbindung zu T-Online
#     besteht, wird automatisch eine Anwahl durchgefuehrt
send "\*0\#"
#     Auf Seitenende warten.
waitdct

#     Zusammengesetzten Text senden geht auch ...
send "\*", "T", "E", "L", "E", "K", "O", "M", "\#"
waitdct
```

Siehe auch:

sendfield
sendtrans
sendtsw

sendfield <str A>, <str B>

Füllt ein Feld der Länge <B> auf einer T-Online-Dialogseite mit dem Inhalt von <A> aus. Füllt <A> das Feld nicht ganz, wird anschließend das T-Online-Steuerzeichen "TER" geschickt. Es werden maximal soviele Zeichen geschickt, wie in <B> als Feldlänge angegeben, auch wenn <A> mehr Zeichen enthält.

Beispiel (Sendfiel.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

input Ort : "Bitte Ort eingeben", ""
input Name : "Bitte Name eingeben", ""
input Vorwahl : "Bitte Vorwahl eingeben", ""
input PLZ : "Bitte Postleitzahl eingeben", ""
input Strasse : "Bitte Strasse eingeben", ""
input Vorname : "Bitte Vorname eingeben", ""

#     Nullseite aufrufen
send "\*0\#"
waitdct

#     Elektronisches Telefonbuch aufrufen
send "\*etb\#"
#     Auf Seitenende warten.
waitdct

#     Unterpunkt "Telefonbuch" waehlen
send "10"
waitdct

#     Feld "Ort" ausfuellen
#     Wenn Ort weniger als 31 Zeichen hat, wird automatisch ein
#     '#'-Zeichen angehaengt, um die Eingabe des Feldes
#     abzuschliessen. Enthaelt Ort mehr als 31 Zeichen, werden
#     ueberzaehlige Zeichen am Schluss abgeschnitten
sendfield Ort, "31"
waitdct

#     andere Felder ausfuellen
sendfield Name, "31"
waitdct

sendfield Vorwahl, "6"
waitdct

sendfield PLZ, "5"
waitdct

sendfield Strasse, "20"
waitdct

sendfield Vorname, "20"
```

```
waitdct

#   Kein Sondersuche
send "n"
waitdct
```

Siehe auch:

Send
sendtrans
sendtsw

sendtrans <list A>

Mit diesem Befehl können sie transparente Daten z.B. über die Mitteilungsseite für transparente Daten (*820#) versenden, wobei die maximale Anzahl der zu übertragenen Zeichen abhängig ist von der Größe des Dialogfeldes. Der Makrostring kann beliebige Zeichen - auch \0-Zeichen - enthalten.

Die Syntax für die nicht darstellbaren Zeichen lautet:

\x##	für hexadezimale Codierung
\o###	für oktale Codierung
\\	für das Zeichen "\

Siehe auch:

send
sendfield
sendtsw

sendtsw <str A>, <str B>, <str C>

Führt ein Telesoftware-Upload durch. Die TSW-Datei <A> wird über das Protokoll <B> mit der Blockgröße <C> gesendet.

Hinweis: Das transparente Eingabefeld muß vor dem Befehlsaufruf bereits anliegen.

Dieser Befehl ist eignet sich in erster Linie für die Einbindung in den T-Online Application Access. T-Online Application Access ist eine einfach zu bedienende Nachrichtenschnittstelle, die anderen Programmen Zugriffe auf T-Online ermöglicht. T-Online Application Access ist wie die T-Online Makrosprache ein Bestandteil der T-Online Software.

Siehe auch:

send
sendfield
sendtrans

set <var A> = <list B>

Die Komponenten der Argumentliste <B> werden aneinandergehängt und der Variablen <A> zugewiesen.

Beispiel (Set.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Name wird "Hugo"
set Name = "Hugo"
debug "Name: ", Name, "\n"

#     Name bekommt neuen Wert "Egon"
set Name = "Egon"
debug "Name: ", Name, "\n"

#     Name aus mehreren Komponenten zusammensetzen
set Name = "Hugo-", Name, " Habicht"
debug "Name: ", Name, "\n"
```

Siehe auch:

sethead
setpart
settail
strcat
strchange
strdelete
strinsert
strlen
strpos

setbtxuser <str A>

Setzt die Zugangsart und die Teilnehmerdaten in der T-Online Software in Abhängigkeit vom Inhalt der Zeichenkette <A>:

Die Syntax lautet:

setbtxuser "<Anschlußkennung> <Teilnehmernummer> <Mitbenutzerzusatz> <Kennwort>"

Hinweis: Es ist eine partielle Übernahme der Einstellungen in der T-Online Software möglich, z.B. in folgender Form: setbtxuser "* * 100 Kennwort" oder setbtxuser "* * * Kennwort".

Beispiel (Setbtxus.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Teilnehmerdaten fuer Zugang setzen
debug "ACHTUNG!!! Dummy-Daten fuer T-Online-Zugang\n"
set Kennung    = "000000000000"
set TlnNr      = "01111122222"
set MBZ         = "1"
set Kennwort = "12345678"
set UserData = Kennung, " ", TlnNr, " ", MBZ, " ", Kennwort

#     Teilnehmerdaten im Decoder setzen
setbtxuser UserData
```

sethead <var A> = <str B>

Das erste Token im Argument <B> wird in die Variable <A> kopiert. Der Wert von <B> bleibt unverändert.

Beispiel (Sethead.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "   Achim Egon Fritz Hugo Karin Thomas Martin    "
debug "Text: <", Text, ">\n"

#     Erstes Wort aus Text holen, fuehrende Leerzeichen werden uebergangen
sethead First = Text
debug "Erstes Wort: <", First, ">\n"
```

Siehe auch:

set
setpart
settail
strcat
strchange
strdelete
strinsert
strlen
strpos

setpart <var A> = <str B>, <str C>, <str D>

Ein Teil des Inhalts von <B> wird in die Variable <A> kopiert. Die Argumente <C> und <D> geben die Position des ersten und letzten Zeichens an, das kopiert werden soll.

Beispiel (Setpart.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
debug "Text: <", Text, ">\n"

#     Bereich aus Text holen und ausgeben
set Start = "10"
set End     = "19"
setpart Part = Text, Start, End
debug "Text von Position ", Start, " bis ", End, ":\n<", Part, ">\n"
```

Siehe auch:

set
sethead
settail
strcat
strchange
strdelete
strinsert
strlen
strpos

settail <var A> = <str B>

Weist <A> den Teil von <B> zu, der nach dem durch sethead erhaltenen String beginnt. Der Wert von <B> wird nicht verändert.

Beispiel (Settail.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "   Achim Egon Fritz Hugo Karin Thomas Martin    "
debug "Text: <", Text, ">\n"

#     Einzelne Namen aus Text holen und ausgeben
Loop:
sethead First = Text
if First == "" goto Ende
debug "<", First, ">\n"
settail New = Text
set Text = New
goto Loop

Ende:
end
```

Siehe auch:

set
sethead
setpart
strcat
strchange
strdelete
strinsert
strlen
strpos

sleep <str A>

Unterbricht die Makroausführung für <A> Sekunden.

Beispiel (Sleep.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Makroausfuehrung fuer 5 Sekunden anhalten, einzelne
#     Sekunden in Hinweisbox zaehlen
set Count = "5"

Loop:
set Text = "Makro schldft noch ", Count, " Sekunden"
messagebox Antwort : "Hinweis", Text, "INFO"
sleep "10"
dec Count
ifgreater Count than "0" goto Loop

messagebox Antwort: "", "", "INFO"
```

Siehe auch:

waitdct
waitdcttime

startapp<str A>

Startet den eingestellten eMail-Client oder WWW-Browser für das Internet. Mögliche Werte für <A> sind "EMAIL" und "INTERNET".

Nach Ausführung des Befehls wird das Makro beendet.

Beispiel (Startapp.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     EMail starten
debug "Starte EMail-Client...\n"
startapp "EMAIL"

#     wird nicht mehr ausgefuehrt, da Makro bei startapp beendet wurde
debug "Starte Internet-Client...\n"
startapp "INTERNET"
```

statusmsg <list A>

Die Komponenten der Argumentliste <A> werden in der angegebenen Reihenfolge zu einer Meldung zusammengesetzt und in der Statuszeile angezeigt.

In der Statuszeile werden außerdem abhängig vom Zustand der Variablen SDEBUG Ausgaben des Befehls **debug** angezeigt.

Beispiel (Statusms.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

# Text in der Statuszeile der T-Online-Software ausgeben
statusmsg "<<< Eigener Text in der Statuszeile >>>"
messagebox Antwort: "Hinweis", "Sehen Sie sich mal die Statuszeile der
T-Online-Software an!!!", "OK"
```

strcat <var A>, <list B>

Die Komponeneten der Argumentliste <B> werden zusammengesetzt und an den Inhalt von <A> angehängt.

Beispiel (Strcat.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Name wird "Andreas"
set Name = "Andreas"
debug "Name: <", Name, ">\n"

#     Name wird verlaengert
strcat Name, " Fritz"
debug "Name: <", Name, ">\n"

#     Name wird verlaengert
strcat Name, Name, " Paula", " Xaver"
debug "Name: <", Name, ">\n"
```

Siehe auch:

set
sethead
setpart
settail
strchange
strdelete
strinsert
strlen
strpos

strchange <var A>, <str B>, <str C>

Der Inhalt des Arguments <B> wird ab der Position <C> in die Variable <A> geschrieben. Die ursprünglichen Zeichen in <A> werden dabei überschrieben.

Beispiel (Strchang.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "Andreas liebt Eis"
debug "Text: <", Text, ">\n"

#    Wort "liebt" gegen "hasst" austauschen
#    funktioniert hier, weil die Worte gleich lang
#    sind
strchange Text, "hasst", "8"
debug "Text: <", Text, ">\n"

#    hier wird der ganze restliche Text
#    nach "Andreas " ueberschrieben
strchange Text, "ist ein Gegner von Rosinen", "8"
debug "Text: <", Text, ">\n"
```

Siehe auch:

set
sethead
setpart
settail
strcat
strdelete
strinsert
strlen
strpos

strdelete <var A>, <str B>, <str C>

Löscht Zeichen aus der Variablen <A>, beginnend bei Position <B>. <C> gibt die Anzahl der zu löschenden Zeichen an. Die folgenden Zeichen werden nach vorne verschoben, um die Lücke zu füllen.

Beispiel (Strdelet.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "Heute gibt es keinen Regen"
debug "Text: <", Text, ">\n"

#     Korrektur des Wetterberichts
strdelete Text, "14", "7"
debug "Text: <", Text, ">\n"
```

Siehe auch:

set
sethead
setpart
settail
strcat
strchange
strinsert
strlen
strpos

strinsert <var A>, <str B>, <str C>

Der Inhalt des Arguments <B> wird in die Variable <A> ab der Position <C> eingefügt. <A> wird dadurch um die Anzahl der Zeichen in <B> länger.

Beispiel (Strinser.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = ""
debug "Text: <", Text, ">\n"

#     Neue Namen am Anfang einfuegen
strinsert Text, "Xaver", "0"
debug "Text: <", Text, ">\n"

strinsert Text, "Fritz ", "0"
debug "Text: <", Text, ">\n"

strinsert Text, "Andreas ", "0"
debug "Text: <", Text, ">\n"
```

Siehe auch:

set
sethead
setpart
settail
strcat
strchange
strdelete
strlen
strpos

strlen <var A> of <str B>

Ermittelt die Anzahl der Zeichen im Argument <B> und speichert diesen numerischen Wert in <A> ab.

Beispiel (Strlen.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = ""
debug "Text: <", Text, ">\n"

#     Neue Namen am Ende einfuegen
set Text = "Andreas"
debug "Text: <", Text, ">\n"

#   Laenge ermitteln (= Position zum Einfuegen am Ende)
strlen Pos of Text

strinsert Text, " Fritz", Pos
debug "Text: <", Text, ">\n"

#   Laenge ermitteln (= Position zum Einfuegen am Ende)
strlen Pos of Text

strinsert Text, " Hugo", Pos
debug "Text: <", Text, ">\n"
```

Siehe auch:

set
sethead
setpart
settail
strcat
strchange
strdelete
strinsert
strpos

strpos <var A> = <str B> in <str C>

Sucht in <C> nach dem ersten Vorkommen von <B> und speichert diese Position in <A> ab. Wurde <B> in <C> nicht gefunden, wird <A> ein leerer String ("") zugewiesen.

Beispiel (Strpos.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Text = "Andreas Detlev Fritz Mark"
debug "Text: <", Text, ">\n"

#    Dieser Name soll aus dem Text geloescht werden
set Delname = "Fritz"

#    Position des Namens suchen
strpos Pos = Delname in Text
debug "Pos = ", Pos, "\n"
ifnot Pos == "" goto Gefunden
messagebox Antwort: "Hinweis", "Name nicht gefunden !!!", "OK"
goto Fertig

Gefunden:
#    Laenge des Namens holen
strlen Len of Delname

#    Name loeschen
strdelete Text, Pos, Len

#    Ist an dieser Position jetzt ein Leerzeichen,
#    dann stand das hinter dem Namen und muss auch geloescht werden
setpart Char = Text, Pos, Pos
ifnot Char == " " goto Fertig
strdelete Text, Pos, "1"
debug "Text: <", Text, ">\n"
messagebox Antwort: "Hinweis", "Name wurde geloescht !!!", "OK"

Fertig:
end
```

Siehe auch:

set
sethead
setpart
settail
strcat
strchange
strdelete
strinsert
strlen

subtract <var A>, <str B>

Von einer Variablen <A>, die einen numerischen Wert enthält, wird der ebenfalls numerische Wert des Arguments <B> subtrahiert.

Ist der Wert von <A> oder der Wert von <B> kein numerischer Wert, wird eine Fehlermeldung ausgegeben.

Beispiel (Subtract.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set A = "30"
set B = "52"
set C = A

#     Subtraktion durchfuehren
subtract C, B

#     Ergebnis ausgeben
debug A, " - ", B, " = ", C, "\n"
```

Siehe auch:

add
Inc
dec
div
mod
mult

system <str A>

Führt das externe Programm <A> aus.

Beispiel (System.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

messagebox Antwort: "Frage", "Moechten Sie den Quelltext zu diesem Makro bearbeiten ?",
"YESNO"

if Antwort == "NO" goto Ende
#     Dieses Makro editieren
set Calltext = "NOTEPAD ", argv_0
system Calltext

Ende:
end
```

waitdct

Hält die Bearbeitung des Makros an, bis T-Online meldet, daß die Seite komplett dargestellt ist. Dies geschieht durch das T-Online-Sonderzeichen "DCT". Dieser Befehl steht in der Regel als nächster Befehl nach send oder sendfield

Beim Aufzeichnen eines Makros über die T-Online-Funktion MAKRO AUFZEICHEN wird nach dem ersten Befehl "*0#" automatisch der Befehl **waitdct** von der T-Online Software eingefügt.

Beispiel (Waitdct.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen, wenn keine Verbindung zu T-Online
#     besteht, wird automatisch eine Anwahl durchgefuehrt
send "\*0\#"
#     Auf Seitenende warten.
waitdct

#     EInige Muellzeichen in die Eingabezeile streuen
send "AA"

#     Timeout fuer waitdct-Befehl setzen (10 Sekunden)
messagebox Antwort: "Hinweis", "Wir erzeugen jetzt absichtlich einen Timeout-Fehler\nBitte nach
\"OK\" 10 Sekunden warten", "OK"
set DCTTIME = "10"

#     Wegen der Muellzeichen (s.o.) wird keine Seitenwahl ausgefuehrt
#     also kommt auch kein Seitenende => Timeout schlaegt bei waitdct zu
send "\*telekom\#"
waitdct

end
```

Siehe auch:

waitdcttime
waittsw
whatdct
sleep

waitdcttime <str A>

Wartet maximal <A> Sekunden auf das Eintreffen eines DCT's.

Trifft das DCT nicht innerhalb des angegebenen Zeitintervals ein, wird ein Timeoutfehler generiert. Dieser kann mit den Befehlen onerror, resume und resumeat verarbeitet werden.

Beispiel (Waitdctt.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Nullseite aufrufen, wenn keine Verbindung zu T-Online
#     besteht, wird automatisch eine Anwahl durchgefuehrt
send "\*0\#"
#     Auf Seitenende warten.
waitdct

#     Einige Muellzeichen in die Eingabezeile streuen
send "AA"

#     Wegen der Muellzeichen (s.o.) wird keine Seitenwahl ausgefuehrt
#     also kommt auch kein Seitenende => Timeout schlaegt bei waitdcttime zu
messagebox Antwort: "Hinweis", "Wir erzeugen jetzt absichtlich einen Timeout-Fehler\nBitte nach
\"OK\" 10 Sekunden warten", "OK"
send "\*telekom\#"
waitdcttime "10"

end
```

Siehe auch:

waitdct
waittsw
whatdct

waittsw <var A>, <var B>

Wartet auf den Empfang einer Telesoftwaredatei. Der Dateiname wird inklusive vollständigem Pfad in der Variablen <A>, die Länge in <B> gespeichert. Der Empfang eines Seitenendekennzeichens (DCT) bricht das Warten ebenfalls ab, <A> enthält dann die leere Zeichenkette. Auf ein waittsw sollte ein waitdct-Befehl folgen.

Beispiel (Waittsw.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     Eine Telesoftwaredatei laden
send "\*0\#"
waitdct
send "\*10223\#"
waitdct
send "99"
waitdct
send "99"

#     Auf das Ende des Downloads warten
waittsw Datei, Laenge
waitdct

#     Information über empfangene Datei ausgeben
debug "Dateiname: ", Datei, "\n"
debug "Länge: ", Laenge, " Bytes"
```

Siehe auch:

waitdct
waitdcttime

whatdct <var A>

Gibt die Art des letzten aufgetretenen DCT-Zeichens von T-Online an. Folgende Werte sind möglich:

1 = DCT Vermittlungsstelle
2 = DCT Externe-Rechner-Verbindung
4 = DCT VT100-Verbindung

Beispiel (Whatdct.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#     T-Online-Verbindung trennen
disconnect
whatdct DCT
debug "DCT: ", DCT, "\n"

#     T-Online-Verbindung aufbauen
send "\*0\#"
waitdct
whatdct DCT
debug "DCT: ", DCT, "\n"
```

Siehe auch:

waitdct
waitdcttime

write <var A> to <str B >, < str L >

Schreibt den Inhalt des Argumentes <A> in die Datei mit dem Namen <B>. Der ursprüngliche Dateiinhalt geht dabei verloren.

Ist zusätzlich der Parameter <L> angegeben, beschränkt sich der Schreibvorgang auf die in dieser Varibalen angegebenen Anzahl von Zeichen.

Hinweis: Über die Variable FILETEXTMODE wird definiert, ob Dateien im Text- oder Binärmode geöffnet werden

Beispiel (Write.mkr):

```
#    Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

#    Dateimodus auf Text setzen
set FILETEXTMODE = "on"

#    Dateiname setzen
set Filename = "FILETEST.TXT"

#    Zeile im Textmodus anhaengen
#    Datei wird erzeugt, wenn sie noch nicht exisitert
#    "\n" wird nach "\r\n" konvertiert
write "Erste Textzeile\nZweite Textzeile\n" to Filename

#    Dateimodus auf Binaer umschalten
set FILETEXTMODE = "off"

#    Jetzt muss man "\r\n" schreiben (wegen Binaermodus)
#    Die vorher geschriebenen Zeilen werden ueberschrieben !!!
write "Erste Textzeile\r\nZweite Textzeile\r\n" to Filename

#    Hinweis ausgeben
debug "Datei ", Filename, " wurde geschrieben\n"
```

Siehe auch:

iniwrite
read

xor <var A> = <str B>, <str C>

Führt bitweise eine exklusive ODER-Operation (<B> XOR <C>) durch.

Beispiel (Xor.mkr):

```
#     Protokollfensterausgaben ermoeglichen
set PROTWIN = "on"
debug "Makroname: ", argv_0, "\n"

set Flags = "133"
set Mask  = "255"

#     XOR-Operation durchfuehren
xor Result = Flags, Mask

#     Ergebnis ausgeben
debug Flags, " XOR ", Mask, " = ", Result, "\n"
```

Siehe auch:

and
or

Token

Tokens sind Worte, die hintereinander in einem String stehen und durch Leerzeichen getrennt sind.