

## MAKROS

[Einführung](#)

[Variablen und Konstanten](#)

[Reservierte Variablen](#)

[Alphabetische Liste der Makro-Befehle](#)

[Makro-Befehle nach Themen](#)

[Inhalt Decoder](#)

## Einführung

Der T-Online-Decoder verfügt über eine Makrosprache, mit deren Hilfe Sie häufig benötigte

Vorgänge automatisieren können. Alle Tätigkeiten, die Sie innerhalb des Btx-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 Btx-Makros werden mit Hilfe eines einfachen Editors, wie zum Beispiel dem Windows-Editor (Notepad), erstellt und in Form einer Textdatei abgespeichert. Sie können Makros daher auch außerhalb des T-Online-Decoders erstellen und bearbeiten. Der T-Online-Decoder 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 Ausführung des Befehls MAKRO AUSFÜHREN startet den Inhalt einer Makrodatei (Erweiterung .MKR). Mit dem Befehl MAKRO NEU ERSTELLEN können Sie einen Texteditor starten, um ein neues Makro zu erstellen. Der Befehl MAKRO BEARBEITEN startet den Notepad-Editor von

Windows mit einem bereits vorhandenen Makro. Eine Hilfestellung zu den in Frage kommenden Makrobefehlen erhalten Sie allerdings nur über die Makrohilfe.

Besonders interessant ist natürlich das Kommando MAKRO AUFZEICHNEN, denn es erlaubt Ihnen, eine beliebige Aktion innerhalb des T-Online-Decoders 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 der T-Online-Decoder 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. Das Ergebnis ist eine Makrodatei, die Sie sich zum Beispiel über das Kommando MAKRO EDITIEREN anschauen, und die Sie jederzeit über das Kommando MAKRO AUSFÜHREN zur Ausführung bringen können.

Die Makrosprache des T-Online-Decoders ist relativ einfach strukturiert. Sie ist mit der

Stapelsprache von MS-DOS vergleichbar, wenngleich sie aber eine Reihe zusätzlicher Befehle beinhaltet, was durch ein [Beispiel](#) schnell verdeutlicht wird.

## Beispiel

Die Anwendung der Makrosprache wird am besten an einem Beispiel deutlich. Das folgende Beispiel wählt das Internet-Mail-Angebot an, das von der Firma Fun zur Verfügung gestellt wird. Es enthält folgende Makrobefehle:

```
connect
send "\c*66000\#"
waitdct
send "\c1"
waitdct
send "\c20"
waitdct
send "\c12"
waitdct
send "\c10"
waitdct
send "19"
debug "E-Mail im Internet-Gateway aufgerufen\n"
end
```

Die einzelnen Befehle sind relativ selbsterklärend. So stellt der Connect-Befehl die Anbindung zum Btx-System unter der aktuellen Einstellung her. Der folgende Send-Befehl sendet die angegebene Zeichenfolge, zum Beispiel zur Auswahl einer Seite, an das System. Damit der folgende Befehl nicht eher ausgeführt wird bis das Btx-System den vorhergehenden Befehl ausgeführt hat, wartet der waitdct-Befehl auf eine Bestätigung. Der Debug-Befehl ist lediglich dazu da, einen Kontrolltext auszugeben, damit Sie erkennen können, daß die Ausführung des Makros an dieser Stelle angelangt ist. Beendet wird ein Makro durch einen End-Befehl.

## Variablen und Konstanten

Die Makro-Sprache des Decoders 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 <i>num</i> (oktal, 1- bis 3-stellig)
\dnum	Zeichen mit dem Zeichencode <i>num</i> (dezimal, 1- bis 3-stellig)
\xnum	Zeichen mit dem Zeichencode <i>num</i> (hexadezimal, 2-stellig)
\a	ä
\A	Ä
\o	ö
\O	Ö
\u	ü
\U	Ü
\s	ß
\#	BTX-Steuerzeichen "TER"
\*	BTX-Steuerzeichen "INI"
\.	BTX-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.

RETVAL

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

SDEBUG

Ein Wert von "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.

## **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 im dort verwendet, wo 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 eine 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.

## Alphabetische Liste der Makro-Befehle

### Nach Themen

add  
append  
call  
ceptprint  
ceptwrite  
clipaddtext  
clipdelete  
clipgettext  
clippgettext  
charcode  
connect  
dec  
debug  
delblanks  
disconnect  
end  
get  
getargv  
getpagenr  
getsh  
getstate  
getstateext  
gosub  
goto  
if  
ifnot  
ifless  
ifeqless  
ifgreater  
ifeqgreater  
inc  
info  
iniread  
iniwrite  
input  
makechar  
messagebox  
offwin  
onwin  
read  
return  
send  
sendfield  
set

sethead  
setpart  
settail  
sleep  
statusmsg  
strcat  
strchange  
strdelete  
strinsert  
strlen  
strpos  
subtract  
system  
waitdct  
whatdct  
write



## **Makro-Befehle thematisch**

[Ablaufsteuerung](#)

[Allgemeine Befehle](#)

[CEPT-Ausgabe](#)

[Clipboard](#)

[Decoder-Befehle](#)

[Dialogbefehle des Benutzers](#)

[Ein- und Ausgabe](#)

[Numerische Operationen](#)

[String-Befehle](#)

[Systembefehle](#)

[Alphabetische Liste](#)

## **Ablaufsteuerung**

call

end

gosub

goto

if

ifnot

ifless

ifeqless

ifgreater

ifeqgreater

return

## Ein- und Ausgabe

[append](#)

[iniread](#)

[iniwrite](#)

[read](#)

[write](#)

## Decoder-Befehle

[connect](#)

[disconnect](#)

[get](#)

[getpagenr](#)

[gettype](#)

[getstate](#)

[getstateext](#)

[getsh](#)

[getversion](#)

[offwin](#)

[onwin](#)

[send](#)

[sendfield](#)

[waitdct](#)

[whatdct](#)

## Numerische Operationen

add

dec

inc

subtract

## String-Befehle

[charcode](#)

[delblanks](#)

[makechar](#)

[sethead](#)

[setpart](#)

[settail](#)

[streat](#)

[strchange](#)

[strdelete](#)

[strinsert](#)

[strlen](#)

[strpos](#)

## **CEPT-Ausgabe**

[ceptprint](#)

[ceptwrite](#)

## Clipboard

[clipaddtext](#)

[clipdelete](#)

[clipgettext](#)

[clipputtext](#)



## Allgemeine Befehle

set

getargv

## Dialogbefehle des Benutzers

debug

info

input

statusmsg

## Systembefehle

sleep

system

**add** <var A>, <str B>

**subtract** <var A>, <str B>

Zu einer Variablen <A>, die einen numerischen Wert enthält, wird der ebenfalls numerische Wert des Arguments <B> addiert (add), bzw. von ihr subtrahiert(subtract).

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

**Siehe auch:** [inc / dec](#)

**Beispiel:**

```
set A = "12"  
add A, "26"  
#   A ist jetzt "38"
```

```
set Zahl = "30"  
subtract Zahl, "52"  
#   Zahl ist jetzt "-22"
```

**append <str A> to <str B>**

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.

**Siehe auch:** [read](#) [write](#)

**Beispiel:**

```
# Fehlermeldung in Protokolldatei schreiben
LogError:
set ErrText = "Fehler aufgetreten in Makro", argv_0, "\n"
append ErrText to "MAKRO.LOG"
```

## **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.

**Siehe auch:** end

## **Beispiel:**

```
#   BTX-Login und Btx plus aufrufen
call "connect.mkr", "\*plus\#"
if CALLVAL == "1" goto Ok
debug "FEHLER !!!"
end
Ok:
#   Ok, weiter geht's
```

**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.

**Siehe auch:** [makechar](#)

**Beispiel:**

```
set count = 1
Anfang:
makechar test = count
charcode cc = test
debug <, test, > = ,
cc, \n
inc count
ifless count than 256
goto Anfang
end
```

## **connect**

Der Decoder baut die Verbindung zu BTX auf und meldet den Benutzer an. Dabei werden die im Decoder eingestellten Zugangsdaten verwendet. Erst nach Abschluß der Zugangsprozedur oder Feststellen eines Fehlers durch den Decoder läuft das Makro weiter.

Anschließend kann mit dem Befehl [getstate](#) festgestellt werden, ob die BTX-Anmeldung erfolgreich war, um entsprechend zu reagieren.

Wenn der Decoder bereits online ist, hat connect keine Wirkung.

**Siehe auch:** [disconnect](#) [getstate](#)

## **Beispiel:**

```
#   BTX-Login
connect
#   erfolgreich ?
getstate STATUS
if STATUS == "DISCONNECTED" goto Fehler
#   BTX-Login OK
```



**debug** <list A>

Verbindet alle Elemente der Liste zu einem String und übergibt diesen der Fehlerausgabe.

**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.

**Siehe auch:** [strdelete](#)

**Beispiel:**

```
set Name = "  Hugo Emil  "  
delblanks Name  
#  Name ist jetzt "Hugo Emil"
```

## **disconnect**

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

**Siehe auch:** [connect](#) [getstate](#)

## **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.

**Siehe auch:** [call](#)

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

Von der momentan angezeigten BTX-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:**

```
# Impressum auslesen und ausgeben
get Line = "1", "1", "40", "1"
sethead Impr = Line
debug "Impressum : ", Impr, "\n"
```

**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.

**Siehe auch:** [Reservierte Variablen](#)

**Beispiel:**

```
# Debug-Modus, wenn ein Aufrufparameter "-d" enthält
set I = "1"
Loop:
ifeqgreater I than argc goto Fertig
getargv ChkArg = I
ifnot ChkArg == "-d" goto Loop
set DEBUG = "on"
goto Loop
Fertig:
```

**getstate** <var A>

Holt den Status des Decoders in die Variable <A>. Besteht eine BTX-Verbindung, ist dieser Wert "CONNECTED", im anderen Fall "DISCONNECTED".

**Siehe auch:** [connect](#) [disconnect](#)

**Beispiel:**

```
# BTX-Login
connect
getstate Status
if Status == "DISCONNECTED" goto Fehler
```

### **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.

**Siehe auch:** [return](#) [goto](#)

### **Beispiel:**

```
#   Holt das Impressum und gibt es aus
gosub GetImpress
debug "Impressum = ", Impr, "\n"
end
GetImpress:
get Line = "1", "1", "40", "1"
sethead Impr = Line
return
```



**goto** <label A>

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

## 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:

```
# Länge von Name soll zwischen 3 und 7 (inkl.) liegen
strlen Len = Name
ifless Len than "3" goto Fehler
ifeqgreater Len than "7" goto Fehler
# Name Ok
```

**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:**

```
# Eingabe des Dateinamens vom Benutzer anfordern
# Defaultname = Nobody
input Name: "Bitte geben Sie Ihren Namen ein", "Nobody"
debug "Hallo, ", Name, " wie geht es Dir ?\n"
```

**inc** <var A>  
**dec** <var A>

Der numerische Wert in <A> wird um den Wert 1 erhöht (inc) bzw. erniedrigt (dec). Ist <A> kein numerischer Wert, wird eine Fehlermeldung ausgegeben. Dieser Befehl wird häufig bei der Programmierung von Schleifen eingesetzt.

**Siehe auch:** [add / subtract](#)

**Beispiel:**

```
# Zahlen 1 bis 10 ausgeben
set I = "1"
Loop:
debug "I = ", I, "\n"
inc I
ifless I than "10" goto Loop
```

**makechar** <var A> = <str B>

Das Argument <B> enthält einen numerischen Wert, der den Zeichencode eines Zeichens repräsentiert, das in der Variablen <A> abgelegt wird.

**Siehe auch:** [charcode](#)

**Beispiel:**

```
# Zeichen "@" in Variable A schreiben  
charcode A = "64"
```

**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 drei Typen der Message-Box, die sich in Art und Anzahl der Schaltflächen unterscheiden und durch das Argument <D> ausgewählt werden:

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

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

<A>	Betätigte Schaltfläche
"OK"	Ok
"YES"	Ja
"NO"	Nein
"CANCEL"	Abbrechen

### **Beispiel:**

```
# Sicherheitsabfrage bei schon vorhandener Datei
messagebox Result: "WARNUNG !!!", "Datei überschreiben ?", "YESNO"
ifnot Result == "YES" goto Weiter
write Inhalt to Filename
Weiter:
```

**onwin**

Dieser Befehl veranlaßt den Decoder, das CEPT-Fenster anzuzeigen. Der Befehl ist wirkungslos, wenn das CEPT-Fenster bereits sichtbar ist.

**Siehe auch:** [offwin](#)

**offwin**

Dieser Befehl veranlaßt den Decoder, das CEPT-Fenster auszublenden. Der Befehl ist wirkungslos, wenn das CEPT-Fenster bereits unsichtbar ist.

**Siehe auch:** [onwin](#)



**read** <var A> from <str B>

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.

**Siehe auch:** [append](#) [write](#)

**Beispiel:**

```
# Länge der autoexec.bat feststellen
read Buf from "c:\\autoexec.bat"
strlen Len of Buf
debug "Dateigröße = ", Len, "\n"
```

## **return**

Beendet eine Unterfunktion. Die Programmausführung wird an der Stelle fortgesetzt, an der die Unterfunktion durch den Befehl [gosub](#) aufgerufen wurde.

**Siehe auch:** [gosub](#) [goto](#)

## **Beispiel:**

```
#   Holt das Impressum und gibt es aus
gosub GetImpress
debug "Impressum = ", Impr, "\n"
end
GetImpress:
get Line = "1", "1", "40", "1"
sethead Impr = Line
return
```

**send** <**list** A>

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

Die Maximallänge von Strings ist unbegrenzt.

**Siehe auch:** [sendfield](#)

**Beispiel:**

```
#  Seitenwahl auf Nullseite  
send "\*0\#"
```

**sendfield** <str A>, <str B>

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

**Siehe auch:** [send](#)

**Beispiel:**

```
#   Teilnehmernummer eintragen
set TlnNr, "0788888888"
sendfield TlnNr, "12"
```

**set** <var A> = <list B>

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

**Siehe auch:** [sethead](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

Der Befehl `set DEBUG = on` aktiviert die Ausgabe aller Makrozeilen in ein Protokollfenster. Dieses muß zuvor durch `set PROTWIN = on` geöffnet werden.

**Beispiel:**

```
set Time = "Uhrzeit = ", Std, ":", "Min", ".", Sek, "\n"
debug Time
#   Bsp.-Ausgabe:      Uhrzeit = 12:17.12
```

**sethead** <var A> = <str B>

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

**Siehe auch:** [set](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set Buffer, "This is a token test"
sethead First = Buffer
#   First ist jetzt "This"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set Buffer = "0123456789"
setpart Teil = Buffer, "3", "6"
# Teil ist "3456"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
# Token Test
set Buffer = "Token Test Makro"
sethead First = Buffer
# First ist "Token"
settail Last = Buffer
# Last ist " Test Makro"
sethead First = Last
# First ist "Test"
```



**sleep** <str A>

Unterbricht die Makroausführung für <A> Sekunden.

**Beispiel:**

```
# 3 Sekunden warten  
sleep "3"
```

**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:**

```
#   Uhrzeit in Statuszeile ausgeben
statusmsg "Uhrzeit: ", Std, ":", "Min", ".", Sek
```

**strcat** <var A>, <list B>

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

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set VName = "Hugo"
set NName = "Habicht"
set Name = VName
strcat Name, " ", NName
#   Name ist "Hugo Habicht"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strcat](#) [strdelete](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set Text = "Hugo liebt Eis"
strchange Text, "hasst", "5"
#   Text ist "Hugo hasst Eis"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strinsert](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set Text = "Hugo liebt kein Eis"
strdelete Text, "11", "5"
#    Text ist "Hugo liebt Eis"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strlen](#) [strpos](#)

**Beispiel:**

```
set Text = "Hugo liebt Eis"
strinsert Text, "kein ", "5"
#    Text ist "Hugo liebt kein Eis"
```

**strlen** <var A> of <str B>

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

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strpos](#)

**Beispiel:**

```
set Name = "Hugo"  
strlen Len = Name  
# Len ist "3"
```

**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.

**Siehe auch:** [set](#) [sethead](#) [setpart](#) [settail](#) [strcat](#) [strchange](#) [strdelete](#) [strinsert](#) [strlen](#)

**Beispiel:**

```
set Buffer = "Telefonnummer"
strpos Pos = "fon" un Buffer
#   Pos ist "4"
```



## **waitdct**

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

**Siehe auch:** whatdct

## **Beispiel:**

```
# Anbieter in SeitWahl aufrufen
connect
send "\*0\#"
waitdct
send "\*", SeitWahl, "\#"
waitdct
```

**whatdct** <var A>

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

**Siehe auch:** waitdct

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

**Beispiel:**

```
send Seitwahl  
waitdct  
whatdct DCTType
```

**write <str A> to <str B>**

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

**Beispiel:**

```
# Logfile starten  
write "Start des LOGFILES" to "MAKRO.LOG"
```

## **Token**

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

## **clipputtext**

*clipputtext* <list A>

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

**clipaddtext**

*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 clippertext.

**clipgettext**

*clipgettext* <var A>

Kopiert den Inhalt der Zwischenablage in <A>.

**clipdelete**

*clipdelete*

Leert die Zwischenablage.



## **ceptwrite**

*ceptwrite* <Filename> <Type>

Speichert die Seite als Text oder Grafik. Als Type steht G für Grafikformat und T für Textformat. Hierbei muß beachtet werden, dass die Zeichen G und T in Anführungszeichen gesetzt wird. Wenn kein Dateiname angegeben wird, dann kann über die Dateiauswahlbox ein Dateiname gewählt werden.

## **ceptprint**

*ceptprint* <Type>

Hierdurch kann die Seite ausgedruckt werden. Als Type steht G für Grafikformat und T für Textformat. Hierbei muß beachtet werden, dass die Zeichen G und T in Anführungszeichen gesetzt wird.

## **iniread**

*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 Decoderverzeichnis. Ist der Name der INI-Datei WIN.INI, wird die WIN.INI im Windows-Verzeichnis verwendet.

## **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.

Hinweise zum Dateinamen siehe beim Befehle [iniread](#).

## **info**

Verbindet alle Elemente der Liste zu einem String und übergibt diesen der Benutzerausgabe. Die Ausgabe des Makrobefehls erfolgt in einem Protokollfenster, welches sich nur öffnet, wenn die Befehlsfolge *setprotwin = on* vorgeschaltet wird.

## **gettype**

*getstate* <var *A*>

Übergibt den Decoderzustand (CONNECTED oder DISCONNECTED an <A>).

## **getversion**

*getversion* <A>

Übergibt die Versionsnummer des Decoders an <A> (z.B. bei einer Versionsnummer 1.23 ist der Wert von <A> 123).

**system**

*system* <str *A*>

Führt das externe Programm <A> aus.



## **getsh**

*getsh* <variable>

Liest die SH-Meldung aus.

Beispiel:

```
getsh status  
if status == SH315 goto Fehler_SH315
```

## **getpagenr**

*getpagenr* <variable>

Durch diesen Befehl wird die aktuelle Seitennummer ausgelesen. Es wird nur die reine Seitennummer übergeben, also bei \*31876543# nur die Ziffernfolge 31876543.

## **getstateext**

*getstateext* <variable>

Durch diesen Befehl wird der erweiterte Decoderstatus ausgelesen.

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

## **Inhalt Decoder**

