Erste Schritte in TUSCRIPT
Importieren, Exportieren, XML

22. ITUG-Jahrestagung
7. Oktober 2015
Goethe- und Schiller-Archiv Weimar

Skript

Matthias Schneider
⟨schneiderm@uni-trier.de⟩

Kompetenzzentrum für elektronische
Erschließungs- und Publikationsverfahren
in den Geisteswissenschaften
Universität Trier
www.kompetenzzentrum.uni-trier.de
www.m-schneider.eu

Gliederung

Einführung

TUSTEP

- Tübinger System von Textverarbeitungs-Programmen (TUSTEP)
- entwickelt an der Universität Tübingen, Programmierung: Prof. Dr. Wilhelm Ott, Kuno Schälkle
- verschiedene, modulare Programmteile: Editor, Satz, Skripting, Register/Indexerstellung, Vergleich/Kollationierung u.v.m.

Designprinzipien nach Ott 2000, S. 97f.:
Modularität → Set von spezialisierten, kombinierbaren Einzelprogrammen
Professionalität → Anforderungen an Benutzer, Leistungsorientierung
Portabilität/Plattformunabhängigkeit → minimalistische graphische Oberfläche und syntaxbasiertes Arbeiten
Integration → für alle Phasen geisteswissenschaftlicher/ textbasierter Arbeiten zu nutzen
Von der Datenerfassung über die Kollationierung und die Datenmodifikation bis hin zur Publikation in Print-, HTML-, Datenbankform oder als eBook muss die Plattform nicht verlassen werden.

- typische Anwendungsbereiche: Paläographie, Editionen, Wörterbücher, Retrodigitalisierung, Satz, Indizes, quantitative Linguistik, automatisches und teilautomatisches Markup, XML-Processing
- Die per default nicht barrierefreie Darstellung der Benutzeroberfläche kann mit wenig Aufwand angenehmer und barrierefreier gestaltet werden.[1]

TUSCRIPT

> TUSCRIPT ist die eigene, hochentwickelte, extrem flexible und effiziente Skriptsprache von TUSTEP, speziell für die Verarbeitung von Textdaten, besonders auch SGML/XML.[2]

Exemplarische TUSCRIPT-Snippets (im September 2015: 146 Beispiele) von Thomas Kollatz können im Portal `Rosetta-Code` recherchiert und mit diversen anderen Programmier- und Skriptsprachen verglichen werden.[3]
TUSCRIPT geht auf die ursprünglich sog. *Kommandomakros* zurück. Daher werden die Begriffe *TUSCRIPT*, *Makros* und seltener *Kommandomakros* oft synonym verwendet. Im TUSTEP-Handbuch ist die Beschreibung zu TUSCRIPT im Kapitel »Makros« zu finden.

---

[1] www.tustep.wikispaces.com/Erleichterungen+im+Umgang+mit+TUSTEP.
[2] http://tustep.wikispaces.com/TUSCRIPT.
[3] http://rosettacode.org/wiki/Category:TUSCRIPT.

TUSCRIPT vs. parametergesteuerte Programme (#KOPIERE, #NUMMERIERE...): Stärken/Schwächen

- relativ eingängige TUSCRIPT-Syntax und sprechende -Nomenklatur
- i.d.R. einfacher zu warten als z.B. #KO-Sprungtabellen, allerdings auch im Vergleich zu parametergesteuerten Programmen mehr Notationsaufwand
- Ähnlichkeit mit gängigen Skriptsprachen
- größere Funktionsvielfalt von TUSCRIPT im Vergleich zu parametergesteuerten Programmen:
  - Interaktion mit dem Web
  - Interaktion mit Betriebssystemebene (z.B. via Zwischenablage, Abfrage von Variablen, Ausgabe von Meldungen über Windows-Fenster)
  - Verarbeitung von Fremddaten (z.B. XML) ohne vorheriges #UMWANDLE[4]
  - Interaktion mit dem Benutzer während der Laufzeit (z.B. über graphische Benutzeroberflächen)
  - Aufruf von externen Programmen
  - Eingabe von UTF–8-Daten via Webbrowser[5]
- TUSCRIPT und parametergesteuerte Programme können zusammen verwendet werden
- Möglichkeit zur Strukturierung von TUSCRIPT-Routinen durch Verwendung von SECTION, SUBMACRO

Grundlagen der Benutzung

Viele Grundlagen der TUSTEP-Benutzung gelten äquivalent bei der Verwendung von TUSCRIPT, etwa die Regeln zur Benennung von Dateinamen oder die Reglen des Pattern Matching. Die TUSCRIPT-Syntax folgt jedoch anderen Regeln als die der parametergesteuerten Programme. Die Positionierung der Anweisung (z.B. durch Einrückungen) ist bis auf singuläre Fälle (z.B. ACCESS-Anweisung) frei wählbar. Häufig finden sich für die aus den parametergesteuerten Programmen gewohnten Operationen (z.B. XX-Parameter in #KO zum Austauschen von Textteilen) Entsprechungen in TUSCRIPT (EXCHANGE-Funktion). Die TUSCRIPT-Funktionen können unterschiedliche Komplexitätsgrade haben und je nach Anwendungszweck passend ausgewählt werden, z.B.:

---

[4] #UMWANDLE ist notwendig, weil die parametergesteuerten Programme nur Dateien im TUSTEP-eigenen Speicherformat verarbeiten können.

[5] http://tustep.wikispaces.com/TUSCRIPT+–+Dateneingabe+im+Browser.

Definitionen:[6]

```
EXCHANGE (var, xtab, apos, epos, xnum, xanz)
EXCHANGE (var, xtab, apos, epos, xnum, xanz, num, anz, xtabx)
```

exemplarischer Aufruf:

```
neu = EXCHANGE (alt, tausch)
neu = EXCHANGE (alt, t, a, e, 2, 1, 1, 0)
```

TUSCRIPT ist nicht (!) case-sensitive, d.h. es macht keinen Unterschied, ob eine Variable als »neu«, »Neu«, »NEU«, »NeU« [...] deklariert wird. Es wird zwischen *normalen Variablen* beliebigen Inhalts[7] und *Sternvariablen*,[8] die mehrere Zeilen enthalten, unterschieden. Bsp.:

```
wort = "Once"
satz = *
DATA Once more unto the breach,
DATA dear friends,
DATA once more
```

TUSCRIPT-Programme: Aussehen und Aufruf von TUSCRIPTs

Organisatorischer Hinweis:
Der Einfachheit halber werden zusätzlich zum Skript die längeren Beispielprogramme in einer Segmentdatei mitgeliefert,[9] die als Makrodatei definiert werden sollte, um die Beispielprogramme möglichst komfortabel aufrufen zu können:[10]

```
#DE,3:skripte.m
```

Ein vollständiges TUSCRIPT, das in Form eines Kommandos aufgerufen werden kann, benötigt die folgende Zeile zur Deklaration von optionalen Spezifikationen:[11]

```
$$!
```

in der zugleich Variablen initiiert und belegt werden können, z.B.:

```
$$! quelle=textfile.tf, ziel=textausg.xml
```

Zur Festlegung des Interpretationsmodus ist folgende Zeile einzufügen:

```
$$ MODE TUSCRIPT, {}
```

---

[6] TUSTEP-Handbuch, Beschreibung der EXCHANGE-Funktion.

[7] TUSCRIPT erwartet keine Deklaration, ob eine zu initialisierende Variable mit dem Datentyp Character, Integer, String o.ä. belegt werden soll. Automatisch unterschieden wird jedoch bei der Variablenbelegung zwischen numerischen Belegungen einerseits und gemischten bzw. nichtnumerischen Belegungen andererseits. Numerische Variablendefinitionen benötigen keine Anführungszeichen und eignen sich für Berechnungen.

[8] Hinweis insb. für Testläufe: Sternvariablen können nicht mit PRINT auf den Bildschirm ausgeben werden. Um dies zu umgehen, kann die Sternvariable mit der Funktion JOIN (s.u. »Datensätze zusammenfassen«) zunächst zusammengefasst und anschließend ausgegeben werden.

[9] Zum Umgang mit Segmentdateien s. http://tustep.wikispaces.com/Grundlagen+Segmentdatei.

[10] http://tustep.wikispaces.com/Grundlagen+Makrodatei.

[11] Die Deklarationszeile muss am Anfang eines TUSCRIPT stehen. Davor sind nur Kommentarzeilen erlaubt.

Lässt man die geschweiften Klammern weg, wird aus Kompatibilitätsgründen nach wie vor der Interpretationsmodus der <>-Parameter angewandt.[12]

Beispiel für ein vollständiges TUSCRIPT:

```
$$!
$$ MODE TUSCRIPT, {}
PRINT "Das ist ein TUSCRIPT-Beispiel."
```

Befehlssyntax

TUSCRIPT unterscheidet *Anweisungen* und *Funktionen*. Funktionen werden grundsätzlich auf Variablen angewandt, während Anweisungen unabhängig hiervon genutzt werden können.

```
PRINT "Hallo" ~ Anweisung
LOOP...END LOOP ~ Anweisung
var_neu = EXCHANGE (var_alt, "|a|ä|") ~ Funktion
```

Zur Belegung von Variablen kann grundsätzlich die Anweisung `SET` vorangestellt werden, die jedoch ohne Weiteres auch weggelassen werden kann. Die beiden nachfolgenden Zeilen sind daher funktional identisch:

```
txt = "Once more unto the breach, dear friends, once more"
SET txt = "Once more unto the breach, dear friends, once more"
```

»Hello World« (1)

- auf Kommandoebene: Anlegen einer permanenten Datei »hello.m«
- Eintragen der folgenden Zeilen:

```
$$!
$$ MODE TUSCRIPT, {}
PRINT "Hello World"
```

- Ausführen auf Kommandoebene: `#MA,hello.m`

»Hello World« (2)

- Anlegen einer permanenten Datei »hello2.m«
- Aufgabe: Modifikation des vorherigen Beispiels durch Verwendung einer Variablen für den Text:

```
$$!
$$ MODE TUSCRIPT, {}
txt = "Hello World"
PRINT txt
```

- Ausführen auf Kommandoebene: `#MA,hello2.m`

---

[12] S. hierzu im Handbuch die Kapitel *{}-Parameter* bzw. *<>-Parameter*.

»Hello World« (3)

- Anlegen einer permanenten Datei »hello3.m«
- Aufgabe: Modifikation des vorherigen Beispiels durch Verwendung von zwei Variablen für einen zusammenzusetzenden Text:

```
$$!
$$ MODE TUSCRIPT, {}
txt = "Hello World"
user = "Max Meier"
PRINT txt, ", ", user
```

- Ausführen auf Kommandoebene: `#MA,hello3.m`

Importieren/Exportieren

Inhalt einer **TUSTEP-Datei** einlesen, vearbeiten und als XML ausgeben

Aufgabenstellung: Der Inhalt einer **TUSTEP-Datei** wird eingelesen, (exemplarisch) verarbeitet und der veränderte Inhalt in eine neue TUSTEP-Datei geschrieben.

```
$$!
$$ MODE TUSCRIPT, {}
- Bsp.skript "TUSTEPFILE"
- TUSTEP-Datei einlesen, Inhalt verarbeiten, Ergebnis ausgeben
quelle = "frosch_reg.tf"
ziel = "frosch_reg1.tf"
- Datei zum Lesen anmelden, andernfalls Fehlermeldung
ERROR/STOP OPEN (quelle, READ, -STD-)
- Datei anlegen bzw. anmelden,  andernfalls Fehlermeldung
ERROR/STOP CREATE (ziel, SEQ-O, -STD-)
- Quelldatei in Variable einlesen
txt = FILE (quelle)
- Text modifizieren
BUILD X_TABLE html = *
DATA |<p>|<p style="font-size: 20px;">|
txt = EXCHANGE (txt, html)
- modifizierten Text in Zieldatei schreiben
FILE/ERASE $ziel = txt
```

Inhalt einer **XML-Datei** einlesen, verarbeiten und wiederum als XML ausgeben (1)

Im ersten Beispiel wird die XML-Datei mit dem Kommando `#UMWANDLE` nach TUSTEP konvertiert, bevor die eigentliche Verarbeitung beginnt und anschließend wieder mit `#UMWANDLE` nach XML konvertiert. Im zweiten Fall wird die XML-Datei direkt verarbeitet, was mit parametergesteuerten Programmen nicht möglich ist (vgl. Unterscheidung oben).

```
$$!
quelle = "frosch-1.xml"
ziel = "frosch-1be.xml"
$$ MODE TUSCRIPT, {}
```

```
  – Bsp.skript "XMLPROC1"
  – Funktionalität:
  – XML-Datei umwandeln, einlesen, Inhalt verarbeiten,
  – Ergebnis als XML ausgeben
  – Quelldatei zum Lesen anmelden, andernfalls Fehlermeldung
  ERROR/STOP OPEN (quelle, READ, -STD-)
  – Zieldatei anlegen bzw. anmelden,  andernfalls Fehlermeldung
  ERROR/STOP CREATE (ziel, FDF-O, -STD-)
  – temporäre Zwischendatei einrichten, ggf. löschen (Erase)
  s1 = "s1"
  ERROR/STOP CREATE (s1, SEQ-E, -STD-)
  – XML nach TUSTEP importieren mit parametergesteuerten
  – Programmen
  EXECUTE #UMW,QU=frosch–1.xml,ZI=s1,CO=utf8,LO=+
  – Text in TUSCRIPT-Variable einlesen
  txt = FILE (s1)
  – Text modifizieren
  BUILD X_TABLE html = *
  DATA |<p>|<p style="font-size: 20px;">|
  txt = EXCHANGE (txt, html)
  s2 = "s2"
  ERROR/STOP CREATE (s2, SEQ-E, -STD-)
  – modifizierten Text aus Variable txt in Zwischendatei schreiben
  FILE/ERASE $s2 = txt
  – Zwischendatei in XML-Datei exportieren
  EXECUTE #UMW,QU=s2,ZI=frosch–1be.xml,CO=utf8,LO=+
```

Inhalt einer **XML-Datei** einlesen, verarbeiten und wiederum als XML ausgeben (2)

```
  $$!
  quelle = "frosch–1.xml"
  ziel = "frosch–1be1.xml"
  $$ MODE TUSCRIPT, {}
  – Bsp.skript "XMLPROC2"
  – Funktionalität:
  – XML-Datei direkt einlesen, Inhalt verarbeiten,
  – Ergebnis als XML ausgeben
  – Quelldatei zum Lesen anmelden, andernfalls Fehlermeldung
  ERROR/STOP OPEN (quelle, READ, -STD-)
  – Zieldatei anlegen bzw. anmelden,  andernfalls Fehlermeldung
  ERROR/STOP CREATE (ziel, FDF-O, -STD-)
  – Text modifiizeren
  BUILD X_TABLE html = *
  DATA |<p>|<p style="font-size: 20px;">|
  – Kompilieren kann u.U. Geschwindigkeitsvorteile bringen
  COMPILE
  – Zugriffe auf Quelle und Ziel definieren
  –  datensatzweise Verarbeitung, UTF8 als Zusatz für Verarbei-
tung der Fremddatei
  ACCESS q: READ/RECORDS/UTF8        $quelle s, txt
  ACCESS z: WRITE/ERASE/RECORDS/UTF8 $ziel   s, txt
  – die eigentliche Verarbeitung läuft in einer Schleife ab:
  – max. 999999 Durchgänge, Quelle datensatzweise einlesen,
  – Variable txt modifizieren, Datensatz nach Ziel schreiben
 LOOP/999999
```

```
  READ/NEXT/EXIT q
    txt = EXCHANGE (txt, html)
  WRITE/NEXT z
 ENDLOOP
  - Zugriffe beenden und Meldung ausgeben
  ENDACCESS/PRINT q
  ENDACCESS/PRINT z
  ENDCOMPILE
```

Nützliches

Austauschen innerhalb von Variableninhalten

Für die Modifikation von Variableninhalten kann u.a. die Funktion EXCHANGE genutzt werden, für die sich in aller Regel die Definition einer Exchange-Table (X_TABLE)[13] anbietet, z.B.

```
var = "Es war einmal eine Königstochter, die ging hinaus in den Wald"
BUILD X_TABLE x1 = *
DATA |Königstochter|Waidfrau|
DATA |ging hinaus in den|jagte im|

var = EXCHANGE (var, x1)
```

Regelmäßig soll eine Austauscheanweisung nur in einem bestimmten Bereich ausgeführt werden, z.B. nur innerhalb von Fußnoten, im Apparattext oder in Metadatenabschnitten. Hierzu kann eine erweiterte Version der EXCHANGE-Funktion eingesetzt werden. Exemplarische Aufgabenstellung: In einem Text (Inhalt der Variable txt) sollen innerhalb des Fußnotentextes (<fn>) und innerhalb von Einschaltungen (<e>) alle Sperrungen in Kursivierungen ausgetauscht werden.

```
BUILD X_TABLE x1 = *
DATA |<s>|<i>|
DATA |</s>|</i>

txt = EXCHANGE (txt, x1, ":<fn>:<e>:", ":</fn>:</e>:")
```

Alternativer Ansatz mit einer Search-Table (S_TABLE)

```
BUILD S_TABLE anf = "|<fn>|<e>|"
BUILD S_TABLE end = "|</fn>|</e>|"
BUILD X_TABLE x1 = *
DATA |<s>|<i>|
DATA |</s>|</i>

txt = EXCHANGE (txt, x1, anf, end)
```

[13] Austausche- oder Suchtabellen dienen zur Definition von Austausche- bzw. Suchzeichenfolgen, auf die in anderen Makroanweisungen Bezug genommen werden kann.

Bedingte Ausführung von Aktionen

Wie in anderen Programmiersprachen üblich, kennt auch TUSCRIPT die bedingte Ausführung von Aktionen mittels der Konstruktion

```
IF bedingung THEN
aktion
END IF
```

sowie erweitert

```
If bedingung_1 THEN
aktion_1
ELSE IF bedingung_2 THEN
aktion_2
ELSE IF bedingung_n THEN
aktion_n
ELSE aktion_m
END IF
```

Beispielhaft sei folgende Anwendung skizziert:

```
$$!
$$ MODE TUSCRIPT, {}
- Bsp.skript "IFTHENBSP"
- Beispiel für die Anwendung einer IF-THEN-ELSE-Abfrage
- Ausgabe der KHM-Auflage auf den Bildschirm
quelle = "frosch_reg.tf"
ERROR/STOP OPEN (quelle, READ, -STD-)
txt = FILE (quelle)
- der Einfachheit halber hier Zusammenfassung
- der Datensätze in einen einzigen String
txt = JOIN (txt)
IF (txt .CT. ":KHM, 1. Aufl.:") THEN
PRINT "Textausgabe: Kinder- und Hausmärchen, 1. Auflage"
ELSE IF (txt .CT. ":KHM, 2. Aufl.:") THEN
PRINT "Textausgabe: Kinder- und Hausmärchen, 2. Auflage"
ELSE IF (txt .CT. ":KHM, 3. Aufl.:") THEN
PRINT "Textausgabe: Kinder- und Hausmärchen, 3. Auflage"
ELSE
PRINT/ERROR "Es scheint sich um keine der ersten drei Auflagen"
PRINT/ERROR "der Kinder- und Hausmärchen zu handeln."
END IF
```

Datensätze zusammenfassen

Mit der Funktion `JOIN` können Datensätze zusammengefasst werden. Dabei wird per default ein Hochkomma (') an die Stelle gesetzt, an der zwei Dateisätze zusammengefügt wurden. Alternativ können andere Trennzeichen angegeben werden.

```
gedicht=*
DATA Königstochter, jüngste,
DATA mach mir auf,
DATA weißt du nicht was gestern
DATA du zu mir gesagt

gedicht_neu1 = JOIN (gedicht)
gedicht_neu2 = JOIN (gedicht, "$")
```

Datensätze aufspalten

Mit der Funktion `SPLIT` können Datensätze aufgeteilt werden. Dabei wird per default bei Hochkommata (') die Unterteilung vorgenommen, was u.U. – wie unten beschrieben – zu unerwünschten Folgen für die Datencodierung führen kann. Es empfiehlt sich daher eine explizite Definition, an welcher Stelle/an welchen Stellen die Datensätze unterteilt werden sollen.

```
text = *
DATA Absatzende. $ Hier startet ein neuer Absatz.

text = SPLIT (text, |":$:")
ERROR/STOP CREATE ("ziel_split", SEQ-O, -STD-)
FILE/ERASE "ziel_split" = text
```

Variante: Wenn man den senkrechten Strich vor der Definition des Zeichens, an welchem die Datensätze gesplittet werden sollen, weglässt, wird dieses Zeichen beim Aufteilen entfernt (=default-Einstellung).

Levenshtein-Distanz berechnen[14]

Die Levenshtein-Distanz ist ein Maß zur Angabe der Unterschiede zwischen zwei Strings beliebiger Länge. Hierbei wird gezählt, wie viele Operationen (Einfügung, Löschung oder Austausch eines Zeichens) notwendig sind, um aus einem String den Vergleichsstring herstellen zu können. Bsp.:
LD(baum, haus) = 2 (b → h, m → s)

Für die Berechnung der Levenshtein-Distanz zweier Strings können die TUSCRIPT-Funktionen `DISTANCE` und `DISTANCE EXACT` genutzt werden. Letztere unterscheidet im Gegensatz zur ersteren auch Unterschied in der Groß- und Kleinschreibung.

```
string1 = "Baum"
string2 = "Haus"
lev1 = DISTANCE (string1, string2)
string1 = "Baum"
string2 = "haus"
lev2 = DISTANCE_EXACT (string1, string2)
```

[14] http://tustep.wikispaces.com/Levenshtein-Distanz.

Inhalte aus Variablen extrahieren

Eine sehr nützliche TUSCRIPT-Funktion ist die Möglichkeit, aus Variablen bestimmte Inhalte auszuschneiden und anderweitig weiterzuverarbeiten. Dabei kann mithilfe der Anfangs- und Endposition sowie weiterer Auswahlkritierien, etwa der Angabe, ab dem wievielten Bereich ausgewählt werden soll, die Extraktion gesteuert werden. Wie bei den o.g. Funktionen kann bei der Verwendung des hierzu vorgesehenen `EXTRACT` gesteuert werden, ob die Anfangs- und Endposition mit ausgeschnitten werden soll oder nicht.

```
  text = "bald darauf klopfte es auch an die Thüre und rief: <q>Königs-
tochter, jüngste, mach mir auf!</q> Sie lief hin"
  BUILD S_TABLE anf = *
 DATA |<q>|
  BUILD S_TABLE end = *
 DATA |</q>|
  zitat = EXTRACT (text, |anf, end|)
  PRINT zitat
```

Steuerung des Programmablaufs/häufige Fehlerquellen

Bei der Nutzung von TUSCRIPT werden sehr häufig Tabellen definiert, die an einer späteren Stelle genutzt werden sollen. Ein hochfrequenter Fehler ist die Definition von Tabellen, die anschließend nicht aufgerufen werden und damit wirkungslos bleiben (z.B. Definition einer `X_TABLE`, die nachher nicht in einer `EXCHANGE-Funktion` eingesetzt wird), was zu Irritationen beim Anwender führen kann, wenn eine erwartete Aktion offenkundig nicht ausgeführt wird.
Ein weiterer, häufig auftretender Benutzerfehler ist der Versuch, eine bereits existierende Tabelle erneut zu belegen. Dies ist problemlos möglich, allerdings erst dann, wenn die betreffende Tabelle zuvor mit einer `RELEASE-Anweisung` freigegeben wurde.
Beim Einsatz der Makro-Funktionen ist ggf. auf die default-Einstellung zu achten. So kann der Einsatz der Funktion `SPLIT` dazu führen, dass Datensätze an unerwünschten Stellen, z.B. innerhalb eines Wortes, das einen Apostroph enthält, unterteilt werden.

Kommentare

Kommentarzeilen können innerhalb von TUSCRIPT mit einem Minus (»–«) eingeleitet werden (Voraussetzung: Einstellung `MODE TUSCRIPT`). Zeilenrestkommentare beginnen mit einer Tilde (~) Bsp.:

```
– so kann ein Kommentar aussehen
var_1 = "Text"
var_2 = var_1 ~ und hier der Zeilenrestkommentar
```

Exportieren von XML-Daten als RTF-File

Das nachfolgende Skript konvertiert eine XML-Datei mit bestimmten Codierungen in die vom Standardmakro *SATZ erwarteten Auszeichnungen und exportiert das Ergebnis als RTF-File. Zum Aufruf des Skripts:

```
$satz2rtf,frosch-1.xml
```

```
$$! quelle
$$ MODE TUSCRIPT, {}
- Bsp.skript "SATZ2RTF"
- TUSCRIPT zur Ausgabe von FROSCH-1.XML als RTF mittels #*SATZ im
- Modus Export. Die Zieldatei wird automatisch erstellt.
- Quelldatei zur Bearbeitung anmelden
ERROR/STOP OPEN (quelle, READ, -STD-)
- Zwischendatei anlegen (hier werden die Umcodierungen gespeichert)
fr_umcod = "fr_umcod.tf"
ERROR/STOP CREATE (fr_umcod, SEQ-O, -STD-)
EXECUTE #UM,{quelle},{fr_umcod},CO=UTF8,LO=+
- Zieldatei für den Text anlegen
fr_satz = "fr_satz.tf"
ERROR/STOP CREATE (fr_satz, SEQ-O, -STD-)
- Quelldatei in Variable einlesen
txt = FILE (fr_umcod)
- Quelldatei von überflüssigem TEI-Markup bereinigen
- die aufgezählten Elemente werden in nichts ausge-
tauscht "||", d.h. gelöscht
BUILD X_TABLE del = *
DATA |<?xml*>||
DATA |<!DOCTYPE*>||
DATA |<{0}/TEI*>||
DATA |<{0}/fileDesc>||
DATA |<{0}/title*>||
DATA |<{0}/publicationStmt>||
DATA |<{0}/sourceDesc>||
DATA |<p/>||
DATA |<{0}/text*>||
DATA |#\[0009]||
DATA |#\[2014]|-|
DATA |<{0}/body>||
DATA |#.,|"|#.'|"|
txt = EXCHANGE (txt, del)
- Umbenennung von TEI-Elementen in Satz-Nomenklatur
BUILD X_TABLE um_1 = *
DATA |<{0}/head>|<{+2=}h2>|
DATA |<p>{|}KHM, 1. Aufl. (1812/15) I: 1-5.{|}</p>|<h2>{=2=}</h2>|
DATA |<lg>|<p left="30" size="9">|
DATA |</lg>|</p>|
DATA |<l>*</l>|{+4=}<br/>|
txt = EXCHANGE (txt, um_1)
FILE/ERASE/PRINT $fr_satz = txt
DEFINE/TUSTEP fr_satz = fr_satz
- Ausgabe als RTF-Datei und öffnen
DATA #*SATZ,<fr_satz>, MO=export
```

TUSCRIPT und parametergesteuerte Programme gemeinsam nutzen

Neben der oben gezeigten Möglichkeit, parametergesteuerte Programme (z.B. #UMWANDLE) innerhalb von TUSCRIPT-Routinen zu verwenden, kann umgekehrt TUSCRIPT in Kommando-Prozeduren integriert werden, z.B.

```
#MAKRO
$$ MODE TUSCRIPT, {}
url = "http://www.tustep.uni-tuebingen.de"
– HTML-Seite abrufen und anschließend umcodieren
inh = REQUEST (url)
inh = DECODE (inh, ISO)
zieltest = "zieltest"
ERROR/STOP CREATE (zieltest, SEQ-O, -STD-)
FILE/ERASE $zieltest = inh
*EOF
#DA,s1,FR=–
#KOPIERE,QU=zieltest,ZI=s1,LO=+,PA=*
 * nur die Hyperlinks isolieren
ZF+       |<a href|
XX        |*{|}<a href*</a>{|}*|<li>{=2=}</li>|
 * Zieladresen ergänzen für TUSTEP-HP-Links:
ZF-   2   |href="http://|
XX    2   |href="|{=0=}http://www.tustep.uni-tuebingen.de/|
Z         |<!DOCTYPE html>|
Z
 * HTML-Rahmen ergänzen
Z         |<!DOCTYPE html>|
Z         |<html lang="de">|
Z         | <head>|
Z         |    <meta charset="iso-8859-1">|
Z         |    <title>Links von der TUSTEP-HP</title>|
Z         | </head>|
Z         |  <body>|
Z         |   <h1>Liste mit allen Links der TUSTEP-Startseite</h1>|
Z         |   <ol>|
ZZZ       |   </ol>|
ZZZ       |  </body>|
ZZZ       |</html>|
SPN      0 3
*EOF
#DA,links.html,FDF-AP
#UMW,QU=s1,ZI=links.html,CO=ISO,LO=+
#– Ziel-HTML im Standardbrowser aufrufen
#MAKRO
$$ MODE TUSCRIPT, {}
BROWSE "links.html"
*EOF
```

Anwendungsbsp. 1: Zusammenspiel BBAW-Verarbeitungsroutine

Inhaltliche Aufgabe:

manuell erstellte Kollationstabellen → Verarbeitung mit TUSTEP/TUSCRIPT (Kontraktion) → Ausgabe als CTE-like Textfile oder XML

Technischer Ablauf:

`TUSTEP-Server` stellt auf dem `localhost` HTML-GUI zur Verfügung, über die Quelle, Zieldatei und Ausgabeformat eingestellt werden → via `CGI` Aufruf von `TUSTEP` im `Batch-Modus` → Verarbeitung (Makro steht in einer `INI-Datei`), ggf. Interaktion mit dem Benutzer, z.B. Abfrage über Windows-Fenster, ob Zieldatei überschrieben werden darf u.ä. → Abschlussmeldung

Anwendungsbsp. 2: TUSCRIPT als Teil einer *Toolchain*

Ausgangssituation:

Im Rahmen einer TUSCRIPT-Verarbeitung soll ein externes Tool für eine bestimmte Aufgabe genutzt werden. Zuvor sollen die Daten mit TUSCRIPT vorbereitet, dann an das externe Programm übergeben und anschließend mit TUSCRIPT weiterverarbeitet werden.[15]

```
$$ MODE TUSCRIPT, {}
- ### Vorverarbeitung ###
- ================================
- Python-Skript zur Berechnung des dekadischen Logarithmus
- ausführen. Als Argument wird exemplarisch
- der zu berechnende Bruch "10/3" übergeben.
- ================================
python = "log10_fi.py" ~ Python-Skript zur Berechnung
ERROR/STOP OPEN (python, READ, -STD-)
pfad = FULL_NAME (TUSTEP, "log10_fi.py")
EXECUTE "{pfad} 10/3"
- ================================
- nach Ausführung des Skriptes wird das Ergebnis aus
- der Ergebnisdatei eingelesen
- ================================
- Ergebnis aus Datei einlesen
ERROR/STOP OPEN ("log10out.txt", READ, -STD-)
log_erg = FILE ("log10out.txt")
log_erg = JOIN (log_erg)
PRINT "Das Ergebnis (aus Datei) lautet: ", log_erg
```

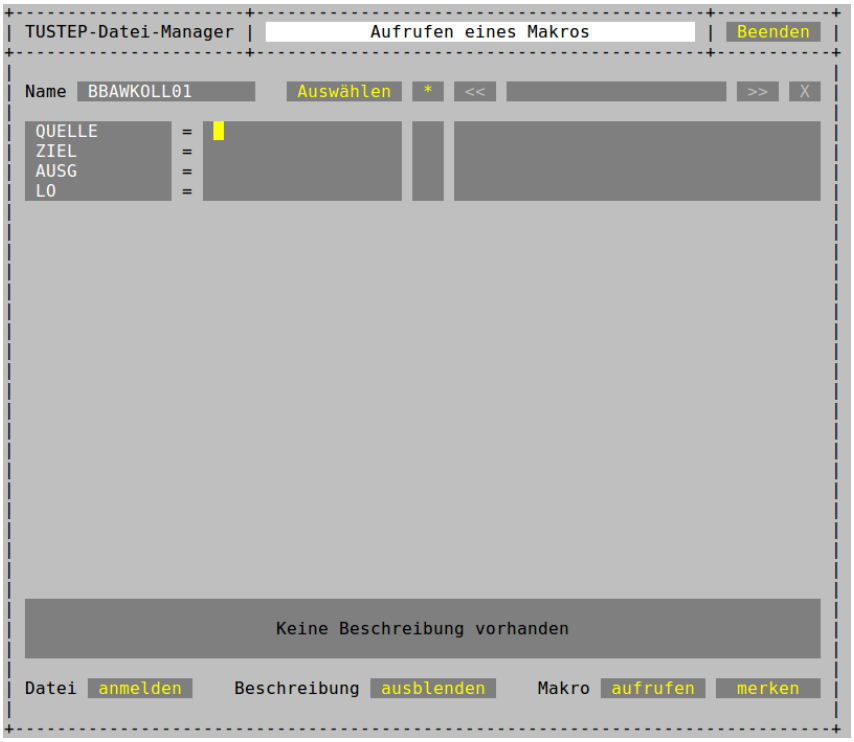
[15] Vgl. http://tustep.wikispaces.com/Externe+Programme+aufrufen.

Anlage

TUSCRIPT-Anwendungsszenarien

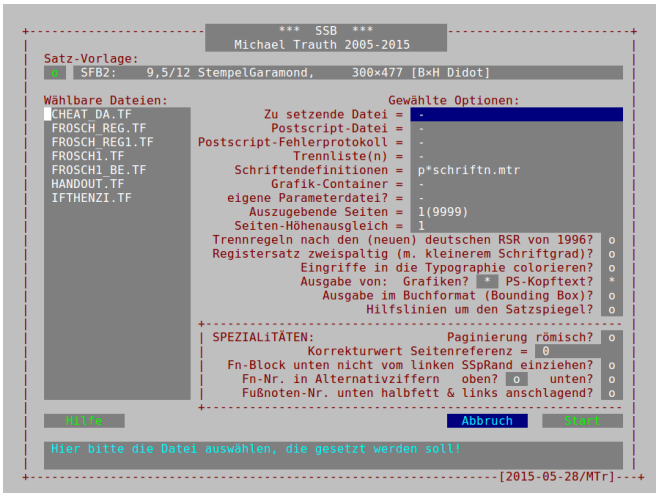
Makrofenster als GUI zur Anwendungssteuerung

Zur Vereinfachung der Bedienung von TUSTEP-/TUSCRIPT-Routinen können mittels TUSCRIPT Makrofenster erstellt werden, die eine graphische Benutzerführung zulassen und den Nutzer von der manuellen Eingabe von Befehlen entbinden. Die einfachste Möglichkeit besteht in der Abfrage von Variablen (z.B. Quelldatei, Zieldatei, Zeichencode usw.) mittels `#*M`.



Ergebnis von `#*M` für eine Routine mit zwei obligatorischen Spezifikationen

Neben dieser automatischen Lösung für eine einfache Eingabemaske können komplexere TUSCRIPT-Benutzeroberflächen mit mehreren Schichten, Benutzerinteraktion, Überprüfung von Eingabefehlern u.ä.m. geschrieben werden.



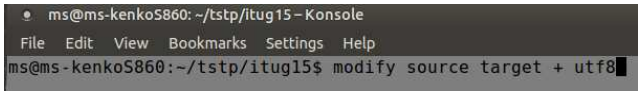
GUI zur Steuerung der Routine SSB.P von Dr. Michael Trauth (Trier)

TUSTEP als System-Kommando ausführen

TUSCRIPT kann auf Systemebene mithilfe der Windows-Eingabeaufforderung (cmd.exe) bzw. mit dem Terminal unter Linux oder MAC OS X gestartet werden:

> TUSTEP kann auch als Programm für ein System-Kommando verwendet werden, bei der keine Interaktion mit dem Benutzer stattfindet. Dabei wird automatisch ein CMD [Command, MS]-Makro aus der INI-Datei [...] ausgeführt und danach TUSTEP wieder beendet.[16]

Nach Einrichtung einer Batch-Datei, die als Konfigurationsdatei fungiert, kann auf der Systemebene der Programmaufruf erfolgen, etwa wie folgt:

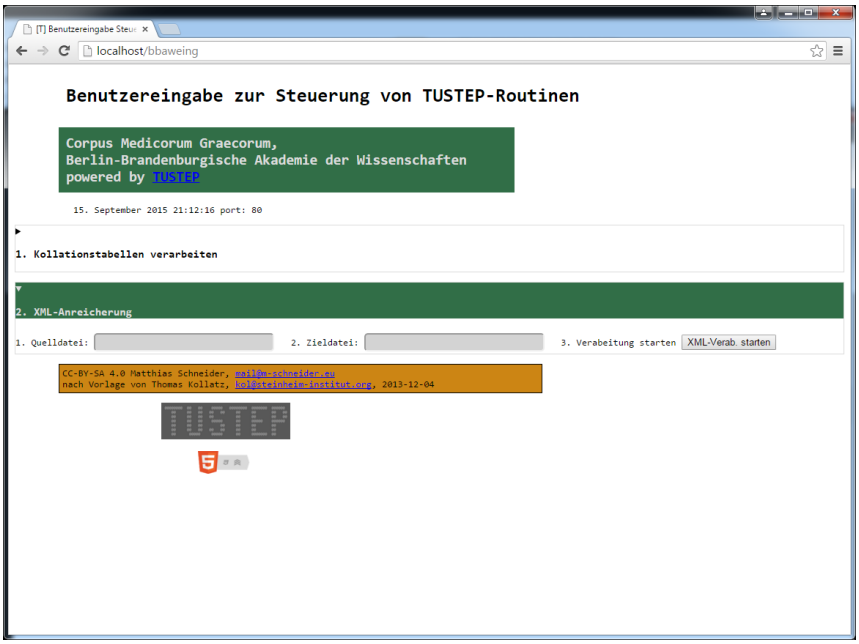


Nach dem Namen des Batch-Skripts werden Quelle und Ziel genannt, für die Spezifikation »Loeschen« der Wert »+« und für »Zeichencode« der Wert »utf8« angegeben. Durch diese Art der Benutzung ist eine Nutzung von TUSTEP ohne (sichtbaren) Aufruf des eigentlichen Programmpakets möglich.

[16] TUSTEP-Handbuch, Kapitel »TUSTEP-Aufruf«.

HTML-GUI zur Eingabe und Steuerung

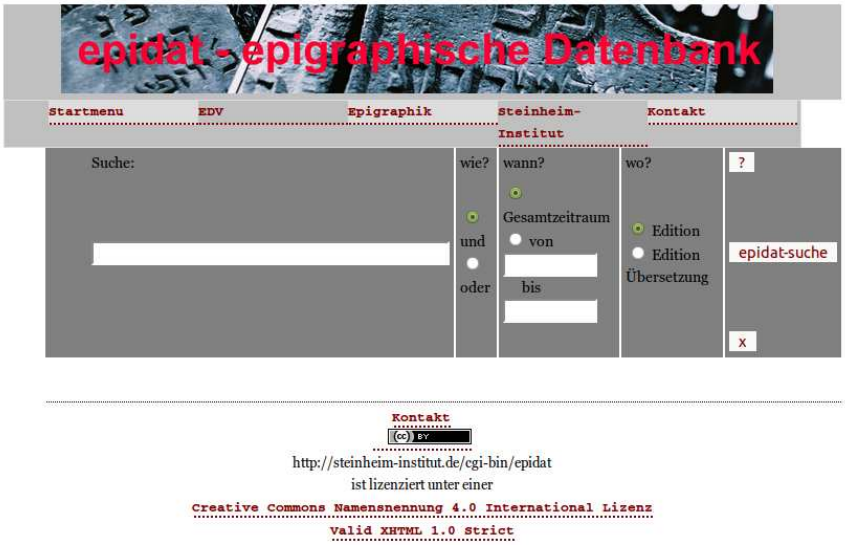
Als eine Erweiterung der o.g. Option, TUSTEP als System-Kommando aufzurufen, können die vom Nutzer erwarteten Eingaben anstatt über ein Terminal auch über eine GUI erfolgen, die z.B. mittels HTML/CSS gestaltet und auf dem Localhost laufen kann. Alternativ kann man GUIs mit anderen Programmiersprachen wie Tcl/Tk, Java o.ä. erstellen, die nach der erfolgten Eingabe aller Angaben durch den Benutzer die TUSCRIPT-/TUSTEP-Prozedur starten und hier alle notwendigen Parameter mit übergeben.



HTML/CSS-GUI zur Eingabe von Daten für zwei unterschiedliche Verarbeitungsskripte, realisiert mithilfe des TUSTEP-Servers auf dem Localhost

Webservice

Abschließend sei auf die Möglichkeit hingewiesen, TUSTEP auf einem Webserver zu installieren und hierüber TUSCRIPT-Funktionalitäten bereitzustellen. Denkbare Einsatzmöglichkeiten sind z.B. die automatische Konvertierung von Textdaten des Nutzers, die einem definierten Schema entsprechen, auf dem Server verarbeitet und an den Nutzer zurückgeliefert werden oder auch die Abfrage einer Datenbank:



Recherche in epidat-Datenbank des Steinheim-Instituts Essen, umgesetzt mit TUSCRIPT via CGI, http://www.steinheim-institut.de/cgi-bin/epidat.

Handbücher, Dokumentationen

TUSTEP liefert bei der Installation die aktuellen *PDF-Fassungen der Beschreibungen* im Installationsverzeichnis mit:

- handbuch.pdf – allgemeine Beschreibung
- satzmakro.pdf – Standardmakro `#*SATZ`
- config.pdf – Installation & Konfiguration
- importexport.pdf – Standardmakros `#*IMPORT, #*EXPORT`

Die Handbücher können über Standardmakros aus TUSTEP heraus aufgerufen werden. Hierzu sind folgende Kommandos notwendig:

```
#*ZEBE – zeigt die allgemeine Beschreibung an
#*ZEBE,satzmakro – Beschreibung von #*SATZ
#*ZEBE,config – Installation und Konfiguration
#*ZEBE,import oder – Beschreibung von #*IMPORT/EXPORT
#*ZEBE,export
```

Den letzen Dokumentationsstand erhält man in der *Online-Hilfe*. Diese kann mit dem Kommando

`#HILFE` bzw. im Editor mit der Anweisung

`hilfe`

oder `CTRL+O` aufgerufen und durchblättert werden. Das Nachschlagen in der Online-Hilfe eignet sich v.a. für erfahrenere Nutzer, die wissen, an welchen Stellen sie im Handbuch nachschlagen müssen.

Ein schellerer Zugriff auf die Dokumentation ist mit dem Kommando

`#SUCHE`

möglich. Von mehreren Benutzungsarten und Auswahloptionen wird hier nur eine besonders praxisrelevante Zugriffsart vorgestellt:

Nach dem Wechsel in die Suche wird per `ALT+I` der erste *I*ndex ausgewählt und der Suchbegriff (oder auch mehrere) aus dem Index ausgewählt. Anschließend kann man sich die Treffer inklusive *U*mgebung in der Beschreibung mit der Tastenkombination `ALT+U`. Zur Vollansicht gelangt man entweder durch Anklicken der betreffenden Trefferabschnittsüberschriftszeile mit der Maus oder mit `ALT+V` (≙ *V*olltext), nachdem man den Cursor im Treffer positioniert hat. Mit `ALT+I` gelangt man zurück zum Index und kann neue Suchbegriffe eingeben. Die Suche kann über `ESC` oder die Schaltfläche `Gehe zu` verlassen werden.

Online-Informationsquellen

TUSTEP-Wiki: http://tustep.wikispaces.com/.

Informationssammlung mit Kurzeinführungen, Programmbeispielen sowie Tipps & Tricks.

Homepage von TUSTEP bei der Universität Tübingen: http://www.tustep.uni-tuebingen.de.

Hier kann TUSTEP kostenlos als Open Source für Linux, Mac OS und Windows bezogen werden.

Homepage der International TUSTEP User Group: http://www.itug.de/.

Hier kann u.a. die TUSTEP-Mailingliste subskribiert werden, über welche Anfragen zu konkreten Problemstellungen sowie TUSTEP- und allgemeine DH-Nachrichten und Veranstaltungshinweise kommuniziert werden. Des Weiteren sind hier Informationen zu TUSTEP-bezogenen Veranstaltungen und Kursen zu finden.

Literatur

Alberding, Stefanie; Schneider, Matthias: Barrierefreiheit in den Digital Humanities. Probleme und Lösungen am Fall TUSTEP, in: Friederike Kerkmann/Dirk Lewandowski (Hrsg.): Barrierefreie Informationssysteme. Zugänglichkeit für Menschen mit Behinderung in Theorie und Praxis, Berlin u.a. 2015, S. 126–139.

Ott, Wilhelm: Strategies and Tools for Textual Scholarship: The Tübingen System of Text Processing Programs (TUSTEP), in: Literacy and Linguistics Computing, Jg. 15, Nr 1/2000, S. 93–108.