Populo

Thomas Berger
ThB@gymel.com

Version 1.16
Stand: 19.05.2003

Inhaltsverzeichnis

Kapitel 1

Einführung

populo bearbeitet strukturierte Rechercheanfragen (wie sie etwa von Web-Browsern initiiert werden) für allegro-Datenbanken und liefert ein geeignet aufbereitetes Ergebnis zurück. Für die eigentliche Recherchearbeit bedient es sich des *avanti*-Servers und seiner Auftragssprache. *populo* ist Free Software unter der Gnu General Public Library Licence.
Downloadmöglichkeiten bestehen unter `http://www.gymel.com/populo/`.
Genauso wie *avanti* kann auch *populo* verschiedene Datenbanken betreuen.
Um die von *populo* zu leistenden Aufgaben besser verstehen zu können, folgt zunächst eine Beschreibung der notwendigen Arbeitsschritte. Die Eingabe der Rechercheanfrage erfolgt über einen (WWW-)Client. Dieser schickt seine Anfrage an das *populo*-Programm `populo.pl`, welches diese Anfrage aufbereitet und einen entsprechenden Auftrag für *avanti* formuliert. Das von *avanti* produzierte Ergebnis muß von *populo* nun noch in geeigneter Weise in eine HTML-Seite gepackt und zurück an den Browser geschickt werden.
In der Praxis gibt es viele Anforderungen an dieses an sich recht einfache Strickmuster:

- Die Anbindung einer Datenbank sollte möglichst flexibel auf Änderungen in der (realen und virtuellen) Pfadstruktur des WWW-Servers reagieren können. Auch aus der Vollanzeige von Datensätzen heraus soll es die Möglichkeit geben, per Link neue Recherchen zu initiieren (assoziative Suche).

  Wünschenswert ist also eine symbolische Notation für URLs innerhalb der Anbindung.

- Die Präsentation eigener Daten soll an das Erscheinungsbild der eigenen Website angeglichen sein und angleichbar bleiben. Dies erfordert eine klare Trennung zwischen Suchmaschine und Layout der Darstellung. Die Komponenten, die das Layout enthalten, sollen von Nicht-Datenbankfachleuten wartbar sein und zwar möglichst mit gängigen Tools (HTML-Editoren etc.) und WYSIWYG[1]-fähig sein.

  Die Layout-Komponenten der Datenbankanbindung sollen also möglichst „reines“ HTML sein.

- Mehrere Datenbanken mit gleichen Parametern sollen auch gleichartig zugreifbar sein. Umgekehrt sollen (man denke an verschiedensprachige Rechercheoberflächen) auf ein und dieselbe Datenbank mehrere parallele Zugriffsmöglichkeiten bestehen.

  Wünschenswert ist also eine freie Zuordnung von „WWW-Datenbanken“ zu realen Datenbanken analog dem symbolischen Mechanismus, den *avanti* bietet.

[1] What You See is What You Get

- Nachdem eine Recherche Null oder zuviele Treffer ergeben hat, soll automatisch eine neue Recherche abgeschickt werden, etwa um die Einzeltreffer zu zählen. Dazu ist die *avanti*-Steuersprache hervorragend geeignet, die Information muß dem Benutzer aber entsprechend aufbereitet präsentiert werden, damit er intellektuell folgen kann.

  Es sind also für einen Job mehrere Ausgabeseiten wünschenswert

- Im extrem häufigen Fall von „kombinierten Recherchen“ mit nur einem Begriff, der zudem zu Null oder zuvielen Treffern führt, soll automatisch auf eine Registersuche umgeschaltet werden.

  Es ist also wünschenswert, daß verschiedene Rechercheeinstiege auf dieselbe Art der Anzeige führen können.

- Hat man Daten über eine Systematik erschlossen, ist es eine komplexe Aufgabe, diese als Rechercheinstrument zu einzusetzen. Erfolgversprechend ist dabei meist eine kombinierte Anzeige aus voller Information des Systematikdatensatzes plus Anzeige von (evtl. mehreren) relevanten Registerausschnitten.

  Wünschenswert ist also eine einfache Kombinierbarkeit verschiedener Grundelemente.

populo versucht auf folgende Weise, diesen Anforderungen an die Flexibilität gerecht zu werden:

1. Es gibt ein einziges Grundprogramm `populo.pl`, welches für alle Anfragen zuständig ist.. Diesem Programm werden als fixe Elemente stets der Name der „Datenbank“ und die Art der Anfrage mitgeteilt.

2. Eine „Datenbank“-spezifische Konfigurationsdatei enthält Informationen über die reale Datenbank und einige *perl*-Routinen, die die Aufbereitung der Information über die Möglichkeiten von *avanti* hinaus zu unterstützen (Etwa um Zeilen aus einem Register „Exemplardaten“ in fünf getrennte Komponenten zu zerlegen).

3. Vorlagendateien (Job-Templates) für die diversen strukturell verschiedenenen Rechercheanfragen an die Datenbank. Dies sind Dateien, die den Syntaxregeln der *avanti*-Sprache genügen. Viele Elemente (etwa welcher Suchbegriff, welches Register etc.) sind natürlich erst während der Laufzeit des Programms bekannt und müssen in der Template-Datei durch Platzhalter angezeigt werden.

4. Vorlagendateien (HTML-Templates) für die diversen strukturell verschiedenenen Präsentationen von Rechercheergebnissen. Dies sind Dateien, die den Syntaxregeln von HTML genügen. Viele Elemente (etwa wieviele Treffer, welche und wieviele Registerzeilen etc.) sind natürlich erst nach Ausführung der konkreten Suchanfrage durch *avanti* während der Laufzeit des Programms bekannt und müssen in der Template-Datei durch Platzhalter angezeigt werden.

Die notwendigen Rechercheangaben werden *populo* als Paare *Name*=*Wert* übergeben. Im weiteren Programmverlauf stehen diese Angaben unter Rückgriff auf die Routine `IniVar` zur Verfügung.
Während die Benennung der Übergabeparameter im allgemeinen völlig frei ist, haben zwei dieser HTTP-Felder feste Bedeutung für *populo*:

1. Die Angabe der Datenbank erfolgt stets als Wert des Felds **db** (wird kein solches angegeben, nimmt *populo* als Default die (nicht vorhandene) Datenbank **nodb**).

2. Die Angabe der auszuführenden Aktion („Jobtyp“) erfolgt stets durch ein HTTP-Feld der Form `t_`*xxx*`[_`*yyy*`]` (typischerweise der *name*-Teil des Submit-Buttons). Hierbei ist *xxx* durch den Jobtyp und der optionale *yyy*-Teil durch einen eventuellen Jobsubtyp zu ersetzen.

   Die Angabe eines Jobsubtyps ermöglicht es, die Angabe eines zusätzlichen Parameters von der Wahl eines Submit-Buttons abhängig zu machen. Gibt es beispielsweise in einem Formular zwei Knöpfe „vorwärts blättern“ und “rückwärts blättern“, so ist es sinnvoll, beide auf eine Jobdatei `blaetter.job` zu führen. Die Knöpfe belegen bei Betätigung aber nur *ein* HTTP-Feld unterschiedlich, daher wird die Art des Knopfes zusätzlich zur gewünschten Auftragsart als `t_blaetter_next` und `t_blaetter_prev` in das *name*-Tag hineincodiert: Der Variable *JobTyp* wird dann `blaetter` zugewiesen und *JobSubtyp* der Wert `next` oder `prev`.

   Ist kein solches HTTP-Feld übermittelt worden, so wird der Jobtyp anhand der datenbankspezifischen Konfigurationsdatei initialisert.

Diese Angaben sind in *populo* im weiteren Verlauf unter den Variablennamen **Db**, **JobTyp** und **JobSubtyp** zugänglich.
Damit die Verarbeitung der Anfrage ordnungsgemäß vorgenommen werden kann, muß es zur Datenbank *Db* eine korrespondierende datenbankspezifische Konfigurationsdatei *Db.pl* geben und zu *JobTyp* eine auftragsspezifische Templatedatei *JobTyp.job*. Damit korrespondiert (falls nicht *JobTyp* im Verlauf der Verarbeitung der Anfrage modifizert wird) eine Datei *JobTyp.htm* mit dem Template für die Ausgabe. In welchen Verzeichnissen diese Templatedateien zu finden sind, ist (datenbankspezifisch) in der Konfigurationsdatei spezifiziert.
Abgesehen von den zwei grundlegenden HTTP-Feldern (und den zugehörigen *populo*-Variablen) ist noch folgendes festgelegt:
Ist keine Datenbank angegeben oder eine nichtexistierende (d.h. es wurde keine Konfigurationsdatei gefunden), so schaltet *populo* automatisch um auf die „festverdrahtete“ Datenbank `nodb` mit dem einzigen *JobTyp* `nodb`

Kapitel 2

Man spricht *populo*

Die grundlegende Designentscheidung bei *populo* besteht darin, den diversen Zuständen der Suchmaschine nicht unterschiedliche Programme entsprechen zu lassen, in deren Programmcode Elemente der *avanti*-Jobs und der HTML-Ausgabeseiten eingebettet sind, sondern Vorlagedateien für die *avanti*-Aufträge und Ausgabeseiten, die von *einem* Programm interpretiert werden. Damit diese Interpretation stattfinden kann, ist aber eine zusätzliche Steuersprache erforderlich.
Dieser Ansatz vermindert einerseits die Flexibilität (aus der Sicht des Programmierers, der mit einer universellen Programmiersprache vorher alles hätte machen können) und erhöht sie andererseits (aus der Sicht des HTML-Codierers oder des Codierers von Aufträgen für *avanti*, der schnell neue Zustände in die Suchmaschine integrieren kann oder vorhandene abwandelt). Dieses Sowohl-als auch gilt ebenso für die Komplexität: Die Zustände sind durch die Auslagerung in verschiedene Dateien einfacher identifizierbar und von ihrer Notation her „reiner“ (*.htm*-Dateien enthalten nur HTML, *.job*-Dateien enthalten Avanti-Befehle, *.pl*-Dateien enthalten (fast) kein Auftragselemente oder Layoutbeschreibungen), der Preis ist aber die zusätzliche Steuersprache, die beherrscht sein will.
Aufgrund der Anforderungen läßt sich schnell ableiten, welches die Grundfunktionalität der Steuersprache sein muß:

- Zugriff auf die übergebenen HTTP-Felder, damit die Benutzereingaben berücksichtigt werden können.
- Auf Ergebnisausgaben von *avanti* zugreifen, etwa die Trefferzahl *lastnum* nach Recherchen.
- Variable Zeichenketten (nach etwaiger Konversion) in Ergebnisse einbauen: Beispiel sind Suchbegriffe oder Trefferzahlen.
- Der **if**-Befehl von *avanti* reicht zur Steuerung nicht aus: Schon beim Formulieren von Abfragen sollten Varianten (`qrix` vorwärts oder rückwärts) darstellbar sein. Für die Ausgabe gelten dieselben Anforderungen: Es ist nicht angemessen, eine komplette *.htm*-Seite zu duplizieren, nur damit dort `ein Treffer` statt `1 Treffer` dargestellt wird.
- Für die Darstellung von Ergebnissen mit variabler Zeilenzahl (Registerauszüge, Kurzlisten, Titelanzeigen) ist eine Schleifenkonstruktion erforderlich: „Für jede Zeile des Ergebnisses formatiere . . . “.

Die Kennzeichnung von Elementen der *populo*-Sprache erfolgt immer durch spezielle Zeichenfolgen, die mit 'PO' beginnen. Danach folgt ein je nach Bedeutung des Sprachelementes unterschiedliches Begrenzungszeichen, der zu interpretierende Teil, und als Abschluß noch einmal das Begrenzungszeichen.
Der einfachste Fall sind im normalen HTML-Text auftretende Variablen. Dabei ist nur der Variablenname anzugeben, was einfach durch die Zeichenfolge

PO!*Variablenname*!

geschieht.
Gelegentlich können dabei allerdings noch kleinere Modifikationen notwendig sein, z. B. eine Umwandlung einzelner Zeichen in HTML-Code. Für solche Zwecke stellt *populo* verschiedene Funktionen bereit. Die notwendige Kennzeichnung erfolgt fast genauso:

PO!*Funktionenname*([*Argument(e)*])!

Beispiel: Nehmen wir an, wir hätten verschiedene Variablen, die wie folgt belegt sind:

```
   Name = 'Paul'
Strasse = 'Pockelsstraße 5'
```

und außerdem in der *.htm*-Datei die Stelle

Person PO!*Name*! wohnt PO!*Htm*(*Strasse*)!.

populo macht dann daraus (**Htm** ist eine von *populo* bereitgestellte Umwandlungsfunktion):

```
Person Paul wohnt Pockelsstra&szlig;e 5
```

Wenn man sich das obige Beispiel genauer ansieht, stellt sich natürlich die Frage, wie man Variable überhaupt mit Werten belegt. Allgemeiner gefragt: Wie bringt man *populo* dazu, zwischendurch gewisse Aufgaben auszuführen, ohne daß dies im Ergebnis zu sehen ist? Die Antwort darauf sind die *populo*-Routinen.
Die Syntax dafür ist

PO&*Routinenname*([*Argument(e)*])&

Routinen können ohne Argumente auftreten, aber auch hier sind runde Klammern notwendig. Jede dieser Aufrufe muß in einer eigenen Zeile stehen, da der Rest einer solchen Zeile ignoriert wird (dies kann als Möglichkeit für Kommentare ausgenutzt werden).
Beispiel: Um die im letzten Beispiel benutzte Variablenbelegung zu erhalten, kann u. a. der folgende Teil in der Template-Datei stehen:

```
PO&Set(Vorname, 'Paul')&
PO&Set(Name, 'Müller')&
PO&Set(Strasse, 'Pockelsstraße 5')&
PO&Set(Ort, 'Braunschweig')&
```

Unglücklicherweise gibt es einen ganzen Zoo von Zuweisungsoperationen, die hier nur kurz aufgelistet werden sollen:

Set für die Zuweisung fester Zeichenketten an Variable

Assign für die Zuweisung der Inhalte von Variablen an andere Variable

Append für das Anhängen von Zeichenketten oder Inhalten von Variablen an eine Variable

IniVar für die Initialisierung von Variablen aus HTTP-Feldern

Collect für das Einsammeln paralleler HTTP-Felder

Poke für das nachträgliche Einfügen von Werten in komplexe Strukturen

`LinkIni` für einfaches Zusammenbauen von Links aus HTTP-Feldern

Es wurde schon erwähnt, daß noch eine konditionale Abarbeitung und eine Schleifenkonstruktion nötig ist. Diese fallen unter den Oberbegriff Kontrollstrukturen und beziehen sich in der Regel auf mehrere Zeilen, die in einer *populo* verständlichen Weise eingeschlossen werden müssen. Die Syntax sieht im Großen und Ganzen so aus:

`PO?`*Kontrollstruktur* `[ ( [` *Argument(e)* `] ) ]?`
Anweisungszeilen
`[PO?...?]`
Anweisungszeilen
`PO?END`*Kontrollstruktur* `[ ...]?`

In den *Anweisungszeilen* dürfen weitere Konstrollstrukturen eingeschachtelt sein.
Auch hier ein Beispiel zu Erläuterung: Es wurde nach Datensätzen mit einer bestimmten Bedingung gesucht, und das erhaltene Ergebnis soll jetzt angezeigt werden. Die Anzahl der gefundenen Datensätze sei dabei in der Variablen **Anzahl** festgehalten worden. Denkbar ist dafür die folgende Einleitung:

```
PO?IF( Anzahl > 1 )?
  Es wurden PO!Anzahl! Datens&auml;tze gefunden.
PO?ELSIF( Anzahl == 1 )?
  Es wurde ein Datensatz gefunden.
PO?ELSE?
  Es wurde kein Datensatz gefunden.
PO?ENDIF?
```

Unter den bisherigen Sprachelementen fehlt noch die Konstruktion, mit der auf Ergebnisse zurückgegriffen werden, die erst während der Abarbeitung des Auftrags durch *avanti* festgelegt werden. Dazu wird die Möglichkeit ausgenutzt, daß mit *avanti* eine beliebige Ausgabe mittels

```
write " ... " newline
```

erzeugt werden kann. Nun muß *populo* diese Zeile nur noch als für sich bestimmt erkennen können. Dies geschieht auf die übliche Weise, diesmal mit den Begrenzungszeichen `':'`:

```
PO:...:
```

Anstelle von `...` steht nun üblicherweise der Aufruf einer Initialisierungsroutine. Die Konstruktion mit `':'` sorgt dafür, daß die Initialisierung bis zum Zurücklesen des Resultats von *avanti* verzögert wird. Dieser Zeitpunkt liegt immer noch *vor* der Ausgabe der Zeilen, d. h. auf das Ergebnis einer solchen Zuweisung, auch wenn sie am Ende des Rechercheergebnis erfolgt, kann schon in der ersten Zeile des Ausgabe-Templates zugegriffen werden.
Beispiel: Folgende Zeile könnte in einer *.job*-Datei vorkommen:

```
write "PO:Set(Txt, Int(" l ") Treffer zu 'PO!Begr!'):" n
```

Mit der Belegung „Blume“ für *Begr* wird die entsprechende Zeile in

```
write "PO:Set(Txt, Int(" l ") Treffer zu 'Blume'):" n
```

umgewandelt und als Bestandteil des Auftrags an *avanti* übermittelt. Im Ergebnis des Auftrags tauch diese Zeile dann als einzelne Zeile der Form

```
PO:Set(Txt, Int( 0005 ) Treffer zu 'Blume'):
```

auf und beim Zurücklesen durch *populo* wird dann die Zuweisung von „`5 Treffer zu 'Blume'`“ an die Variable *Txt* durchgeführt.
Im Zusammenhang mit der speziellen, reservierten Variablen **LineTyp** ist diese Konstruktion unabdinglich:

Während des Zurücklesens des Resultats von *avanti* wird jede Zeile entsprechend dem *aktuellen* Wert von *LineTyp* vorbehandelt. Liefert eine Anfrage mehrere Sorten Ausgabe, also etwa, weil unter der Kontrolle von Avanti entschieden wird, ob ein Registerausschnitt oder eine Kurztitelliste geliefert wird, so kann die Initialisierung von *LineTyp* nicht beim Absenden des Jobs erfolgen, sondern erst später:
Beispiel: Um *populo* mitzuteilen, daß *avanti* als nächstes Registerzeilen liefert, muß in der *.job*-Datei an der entsprechenden Stelle die Zeile

```
...
find PER PO!Begriff!
if ok jump liste
//  hier also Null Treffer, wir sparen uns einen leeren Schirm
//                    und bieten Browsing an:
write "PO:Set(LineTyp, 1):" newline //  Registerzeilen folgen
qrix PER PO!Begriff!
jump daswars
:liste
write "PO:Set(LineTyp, 3):" newline //  Kurztitelzeilen folgen
list recnum
:daswars
...
```

Für typische Arten von Avanti-Resultaten sind Vorverarbeitungen schon vordefiniert:

0	unspezifiziert, Fehlermeldungen	
1	Registerauszüge	`qrix f 1`
(2)	erkannte Verweise in (1)	nur Ausgabe!
3	Kurztitellisten	`list recnum`
4	Vollausgabe	`download`
5	Satznummern intern speichern	`list internal`
11	Registerauszüge, noch Kurztitel holen	`qrix f 1`
(12)	erkannte Verweise in (12)	nur Ausgabe!
13	Satznummern für vollständige Kurzanzeige	`list internal`
14	Zwischenspeichern vollständiger Kurzanzeigen	`list internal`
(21)	geholte Kurztitelzeile zu 11	nur Ausgabe!
(22)	geholte Kurztitelzeile zu 12	nur Ausgabe!

Falls für eigene Zwecke (untergliederte Register etc.) stärkere Differenzierungen erforderlich sind, sind weitere *LineTyp*en zu definieren, dazu muß aber typischerweise an mehreren Stellen in die datenbankspezifische Konfigurationsdatei eingegriffen werden.

komplexes Beispiel (Job): In Abhängigkeit von den Eingaben soll ein Registerauszug erstellt werden:

```
//PO&IniVar(Startwert,s1)&        Startwert aus HTTP-Feld s1
//PO&IniVar(Endwert,s2)&          Endwert aus HTTP-Feld s2
//PO&IniVar(Register,ix)&         Register aus HTTP-Feld ix
qrix f 1
qrix m 99
//      MaxShowRegister stammt aus der Konfigurationsdatei
qrix n PO!MaxShowRegister!
//    Der Knopf 'rückwaerts blättern' habe im HTML-Formular für
die
```

```
//             Eingabe den Namen t_..._prev bekommen
//PO?IF( JobSubtyp eq "prev")?
  qrix -
//PO?ENDIF?
write "PO:Set(LineTyp,1):" newline //          WICHTIG!
//       alternativ: Vorher Zeichenkette zusammenbasteln,
//              braucht aber ebenfalls ein 'if'...
//PO?IF( Endwert )?
  qrix PO!Register! PO!Startwert! @ PO!Endwert!
//PO?ELSE
  qrix PO!Register! PO!Startwert!
//PO?ENDIF?
```

komplexes Beispiel (zugehörige Ausgabe):

```
<HTML>
<BODY>
<!-- "Link" wird vorinitialisiert: -->
<!-- PO&LinkIni(Link,db)& -->
<!-- %RegInfo aus der Konfigurationsdatei haben wir mit einem
  Feld Caption für den Klartext zum symbolischen
  Registernamen versehen -->
<!-- PO&Localize(RegInfo, Register, R_)& -->
<P>Gesucht wurde PO!Htm(Startwert) <BR>
  im Register PO!R_Caption!</P>
<PRE>
<!-- PO?LOOP? -->
```

Weil *LineTyp* vor `qrix` auf 1 gesetzt war, haben wir jetzt Zugriff auf die Registerbestandteile als *L_Count* (Zählung), *L_Recno* (Satznummern) und *L_0* (Registerinhalt). Wir formatieren die Zeilen als Links „in die Datenbank“ vom Anfragetyp `showrec`:

```
  (PO!L_Count!) <A HREF="PO!Link!&t_showrec=xyz&recs=PO!L_Recno!">
    PO!Htm(L_0)!</A>
<!-- PO?ENDLOOP? -->
</PRE>
Tsch&üuml;s!
</BODY>
```

Kapitel 3

Referenzteil

Nachdem in den vorigen Kapiteln eine Übersicht über *populo* gegeben wurde folgen nun Präzisierungen und Syntaxregeln für die *populo*-Steuersprache.
verwendete Notationen:

- **abc** kennzeichnet einen festen (Variablen-)Namen (z. B. **JobTyp**) (nicht mit dem Inhalt der Variablen zu verwechseln)
- *abc* Diese Teile sind geeignet zu ersetzen, je nach Zusammenhang können dies Variablen- oder Strukturnamen sein.
- <*abc*> wie *abc*, allerdings sind auch Zeichenketten und komplexe Ausdrücke gestattet.
- „abc“ sonstige Hervorhebungen

[abc] optionale Teile

3.1 Funktionen, Routinen und Variable

Zur Kenntlichmachung von *populo*-Sprachelementen dienen die Zeichenfolgen

PO!...! für Variablen und Funktionen,

PO&...& für Routinen (die im Gegensatz zu Funktionen keinen Rückgabewert haben) und

PO?...? für Kontrollstrukturen.

Zusätzlich gibt es für `.job`-Dateien noch die Zeichenfolge

PO:...: für Variablen und Befehle, die von *populo* erst nach Erhalt des Ergebnisses von *avanti* ausgewertet werden sollen.

Die Konstruktionen **PO&...&** und **PO?...?** müssen jeweils in einer eigenen Zeile stehen, da alles außerhalb dieser Zeichenfolgen ignoriert wird. Insbesondere sollten diese Zeilen keine weiteren relevanten Informationen enthalten! (Kommentare sind hingegen möglich.)
Bei allen Sprachelementen sind zwischen '`PO`' und dem ersten Begrenzungszeichen keine Leerzeichen (oder sonstige Zeichen) erlaubt, Leerzeichen zwischen den verwendeten Begrenzungszeichen und dem jeweiligen Inhalt werden allerdings ignoriert. Die Konstruktionen **PO!...!** und **PO:...:** dürfen das jeweilige Begrenzungszeichen ('`!`' bzw. '`:`') nicht in ihrem Inneren

enthalten, bei Zeilen mit den Konstruktionen **PO&...&** oder **PO?...?** darf das jeweilige Begrenzungszeichen (’&’ bzw. ’?’) hingegen im Rest der Zeile nicht mehr auftreten. [1]

3.1.1 Variable und Strukturen

`PO!`*Variable*`!`

Variable enthalten stets Zeichenketten, die jedoch auch als Wahrheitswerte (’’ und ’0’ sind „falsch“, alle anderen Werte wahr) und Zahlwerte benutzt werden können.
populo kennt zwei verschiedene Arten von Variablen: neben den üblichen, einfachen Variablen (Skalare) gibt es indizierte Variable. Während ein Index von Variablen in anderen Programmiersprachen häufig innerhalb irgendeiner Art von Klammern angegeben wird, sieht dies bei *populo* etwas anders aus: Der Index wird mit Unterstrich ’_’ an den Variablennamen, der dabei nur aus einem Großbuchstaben bestehen darf, angehängt. Der komplette Variablenname inklusive ’_’ und Index verhält sich dann wie der Name einer einfachen Variable, nur daß keine Zuweisungen daran erfolgen dürfen. Indizierte Variablen kommen in *populo* hauptsächlich im Zusammenhang mit Strukturen vor, etwa beim Zugriff via der Routine **Localize**, aber auch bei den Kontrollstrukturen `STRUC` und **LOOP**. Beide Arten von Variablen sollten nicht mit den von *populo* selbst verwendeten (*perl*-)Variablen oder den Namen der HTTP-Felder verwechselt werden.
Gültige Namen von Variablen dürfen nur aus Buchstaben (allerdings ohne Umlaute) und Ziffern bestehen. Außerdem müssen sie immer mit einem Großbuchstaben beginnen.

populo-Strukturen bieten die Möglichkeit, ähnlich einer Liste inhaltlich zusammengehörende Information in ihrem Zusammenhang zu belassen.
perl-intern sind sie als sogenannte Hashes realisiert, d. h. durch Zeichenketten („Schlüssel“) indizierte Listen. Sie werden entweder direkt innerhalb der datenbankspezifischen *Db.pl* oder mit Befehlen von *populo* selbst definiert. Ihr Aufbau ist

```
%Struktur = (
'#Default#' => Defaultschlüssel,
'#Labels#' => [Label1, Label2, ...],
  Schlüssel1 => [Wert11, Wert12, ...],
  Schlüssel2 => [Wert21, Wert22, ...],
  .
  .
  .
);
```

Die Zeilen mit `#Default#` und `#Labels#` sind optional, während der Laufzeit von *populo* wird meist noch ein weiterer Schlüssel `#Labelsnr#` hinzugefügt. Die eckigen Klammern ’`[...]`’ gehören hier zur Syntax von *perl*, sie kennzeichnen in *perl* anonyme Arrays[2].

---

[1] Insbesondere dürfen **PO!...!**-Zeichenfolgen keine weiteren **PO!...!**-Zeichenfolgen enthalten, wohl aber können **PO:...:**-Zeichenfolgen **PO!...!**-Zeichenfolgen enthalten. Diese Regel verhindert auch Bestandteile der Art `PO!Htm('Dumpfbacke!')!`. Hier hilft dann nur der Umweg `PO&Set( Error, 'Dumpfbacke!' )& PO!Htm(Error)!`

[2] Hier ist vielleicht eine Bemerkung zur Verwendung des Zeichens ’’ im Schlüsselteil der verwendeten Strukturen angebracht: Wenn der verwendete Schlüssel als *perl*-Befehl mißverstanden werden kann, sind die Zeichen ’’ am Anfang und Ende notwendig, ansonsten kann auch auf sie verzichtet werden. (Im Beispiel leitet ’#’ sonst einen *perl*-Kommentar ein, die Zeilen würden einfach ignoriert.) Wer immer auf der sicheren Seite sein will, hat eben etwas mehr Schreibarbeit.

Der Spezialschlüssel #Default# gibt eine Vorzugsauswahl an, die in der Routine **IniVar** und bei der Kontrollstruktur **STRUC** berücksichtigt wird.
Der Schlüssel #Labels# enthält Namen für die einzelnen Elemente der Wertelisten, die die Routine **Localize** benutzt, um die entsprechenden Einträge wieder zugänglich zu machen.
Alle *Label* und der Name der Struktur selbst müssen den Regeln für gültige Variablennamen entsprechen (vgl. den Abschnitt Variablen), insbesondere müssen sie mit Großbuchstaben beginnen!
Der Name *Struktur* darf nur einmal vorkommen, insbesondere darf es auch keine gleichnamige Variable geben. *Schlüssel* und *Label* dürfen innerhalb einer Struktur nur einmal, *Werte* natürlich mehrfach auftreten.
Zu Strukturen gehört fast immer ein oder mehrere in der Konfigurationsdatei als *perl*-Array anzugebende *Indexsysteme*, die eine Vorzugsreihenfolge der Schlüssel angeben, z. B.

@*Reihenfolge* = (*Schlüssel1*, *Schlüssel2*, ...);

Indexsyteme können im Rahmen der *populo*-Sprache nicht manipuliert werden.
Bemerkung: In der datenbankspezifischen Datei *Db.pl* können auch andere assoziative Vektoren vorkommen, die der hier beschriebenen Syntax nicht entsprechen müssen, da sie nicht als Strukturen verwendet werden. Ein Beispiel dafür ist der Vektor %Defaults, der immer definiert sein muß.
Das folgende Beispiel stammt aus der mitgelieferten Datei avdemo.pl:

```
@CodierKeys = ('html', 'ansi', 'dos'); # ein Indexsystem
%Codierung = ( # Struktur
'#Default#'=> 'html',
'#Labels#' => ['Erlaeuterung', 'Includetabelle'],
   html => ['HTML-Codierung', 'd'],
   ansi => ['ANSI-Codierung', 'p-ansi'],
   dos => ['DOS-Codierung', 'p-dos'],
);
```

Beim Aufbau eines Formulars wird man die so definierte Struktur etwa wie folgt einsetzen: Typischerweise bereits in der Job-Datei wird der Wert des HTTP-Feldes cod (falls vorhanden) in die Variable *SelCodierung* übertragen, sonst der beim #Default#-Schlüssel der Struktur *Codierung* angegebene:

```
//PO&IniVar(SelCodierung,cod,Codierung)&
```

Bei der Ausgabe wird man dann ein Feld mit Radio-Buttons für die Liste der möglichen Codierungen aufbauen:

```
<FORM ACTION="PO!Pop!" METHOD="POST">
<INPUT TYPE="HIDDEN" NAME="db" VALUE="PO!Db!">
...
<!-- PO?STRUC(Codierung, CodierKeys, SelCodierung)? -->
```

Der folgende Abschnitt wird jetzt für jeden Eintrag in *CodierKeys* durchlaufen, dabei ist der Schlüssel via *S_Key* zugänglich, die dazugehörige Liste in den Variablen *S_Erlaeuterung* und *S_Includetabelle*, entsprechend den Labels Erlaeuterung und Includetabelle. Zusätzlich stehen die beiden Wahrheitswerte *S_Select* (stimmt der aktuelle Schlüssel mit *SelCodierung* überein?) und *S_Default* (stimmt der aktuelle Schlüssel mit dem #Default#-Eintrag von *Codierung* überein?) zur Verfügung.

```
<!-- PO?IF(S_Default)?            kursiv, falls default
<i>
   PO?ENDIF -->
```

```
<!-- PO?IF(S_Select)?          "checked" falls zutreffend
<INPUT TYPE="RADIO" NAME="cod" VALUE="PO!S_Key!" CHECKED="yes">
   PO?ELSE? -->
<INPUT TYPE="RADIO" NAME="cod" VALUE="PO!S_Key!">
<!-- PO?ENDIF? -->
  PO!Htm(S_Erlaeuterung)! <BR>
<!-- PO?IF(S_Default)?          kursiv aus nicht vergessen
</i>
   PO?ENDIF -->
<!-- PO?ENDSTRUC -->
...
</FORM>
```

3.1.2 Funktionen

PO!*Funktion*(*Argument* [,*Argument* ...])!
PO! ...*Funktion*(*Argument*) ...!
PO&*Routine*(...*Funktion*(*Argument*) ...)&
PO*Kontrollstruktur*(...*Funktion*(*Argument*) ...)?

Funktionen nehmen genau ein Argument und liefern ein Resultat, welches bei Auswertungen anstelle des Funktionsaufrufs eingesetzt wird.

3.1.3 Routinen

PO&*Routine*(*Argument* [,*Argument* ...])&
write ...PO:*Routine*(*Argument* [,*Argument* ...]): newline

Routinen nehmen mehrere Argumente auf und wirken nur durch ihre Seiteneffekte, d. h. es wird typischerweise eine oder mehrere Variable modifiziert, die vordefiniert sind oder deren Name als Argument übergeben wurde.
Zeilen mit **PO&...&**-Konstruktionen werden sofort vollständig entfernt, insbesondere darf stets nur eine solche Konstruktion in einer Zeile vorliegen.
Wird in *.job*-Dateien die zweite Form mit ':' benutzt, so wird die Auswertung der Routine auf den Zeitpunkt des Zurücklesens des Resultats von *avanti* verlegt, somit können von Avanti berechnete Zeichenketten verarbeitet werden.
Beispiel: in *allegro.job* für die Beispieldatenbank gibt es vielfältige Verzweigungen. Welcher Art die von *avanti* zurückgelieferte Ausgabe ist, wird erst während der Ausführung des Jobs festgelegt, so daß auch *populo* dies „durch *avanti*“ mitgeteilt werden muß:

```
//           Es folgen Zeilen von Typ 5 = Satznummern
write "PO:Set(LineTyp,5):" newline
list internal
```

3.2 Kontrollstrukturen

PO?*Kontrollstruktur*[(*Argument* [,*Argument* ...])]?
...
PO?END*Kontrollstruktur*[(*beliebiger Text*])?

Kontrollstrukturen bestehen mindestens aus einem ein- und einem ausleitenden Teil.
Zeilen mit **PO?...?**-Konstruktionen werden sofort vollständig entfernt, insbesondere darf stets nur eine solche Konstruktion in einer Zeile vorliegen.
Die Bedingungsprüfungen bei **IF** und **LOOP** sind so zu verstehen: Hier kann entweder eine Variable (leer vs. nicht leer), eine spezielle Testfunktion (z. B. **CheckMe**) oder ein direkter Test eingesetzt werden. Ein solcher direkter Test besteht aus zwei Argumenten und einem dazwischenstehenden Vergleichsoperator. Zur Verfügung stehen die aus *perl* stammenden Operatoren '<', '<=', '==', '!=', '>=', '>', ' lt ', ' le ', ' eq ', ' ne ', ' ge ' und ' gt ', sowie ' = ' und ' ! ', ausserdem die logischen Verkettungsoperatoren ' && ' und ' || '.

3.2.1 IF-ELSIF-ELSE-ENDIF

PO?IF(<*Bedingung1*>)?
Anweisungen1
[PO?ELSIF(<*Bedingung2*>)?
Anweisungen2]
[...]
[PO?ELSE[*Kommentar*]?
Anweisungen_else]
PO?ENDIF[*Kommentar*]?

Prototyp: #e# für **IF** und **ELSIF**

Entspricht einer üblichen if-els(e)if-else-endif-Konstruktion.

Die PO?ELSIF(...)?- und PO?ELSE[...]?-Teile sind optional.
Die Bedingungen werden nacheinander ausgewertet, bis eine Bedingung zutrifft. In diesem Fall wird der zugehörige Anweisungsteil durchgeführt, der Rest übersprungen. Trifft keine Bedingung zu, wird der optionale Anweisungsteil zwischen **ELSE** und **ENDIF** ausgeführt.

3.2.2 LOOP-ENDLOOP

PO?LOOP[(<*Bedingung*>)
?
...
PO?ENDLOOP[*Kommentar*]?]

Prototyp: #[e#

Eine Schleifenkonstruktion, die die von *avanti* zurückgelieferten Ergebniszeilen verarbeitet (solange *Bedingung* erfüllt ist).

Die *Anweisungen* werden für jede der von *avanti* zurückgelieferten Ergebniszeilen durchlaufen. (Ist *Bedingung* angegeben, nur solange, wie *Bedingung* auf die Ergebniszeilen zutrifft, u. U. also keinmal). Die Zeilen waren bereits beim Empfangen aufbereitet und verschiedenen Typen zugeordnet worden, jetzt liegt diese Zuweisung in der Variablen *LineTyp* für Tests bereit, die Inhalte der Zeilen verhalten sich wie eine nach **L_** lokalisierte Struktur, handelt es sich etwa um Registerzeilen oder Kurztiteleinträge, so enthält **L_Recnums** die Satznummer, **L_0**, **L_1** etc. enthalten die von den im datenbankspezifischen Perlscript definierten Hooks &ParseRegisterLine (6.1.3) und &ParseSTLentry (6.1.4) aufbereiteten Zeilen des Resultats von *avanti*.

3.2.3 STRUC-ENDSTRUC

PO?STRUC(*Struktur*, *Indexsystem* [, [<*Auswahl*>][, *Shortcut*]])?
Zeilen
PO?ENDSTRUC[*Kommentar*]?

Prototyp: #bb[eb#

Eine Schleifenkonstruktion über bestimmte Schlüssel der *Struktur*.

Für jeden Schlüssel aus *Indexsystem* wird *Struktur* nach *Shortcut* (falls nicht angegeben: **S_**) lokalisiert und die *Zeilen* ausgewertet.
Ist der aktuelle Schlüssel identisch mit dem in der optionalen *Auswahl* angegebenen, so ist *S_Select* wahr; stimmt er mit dem im Spezialschlüssel **#Default#** von *Struktur* spezifizierten überein, so ist *S_Default* wahr. Der aktuell durchlaufene Eintrag von *Indexsystem* ist als **S_Key** zugänglich.

3.2.4 WHILE-ENDWHILE

PO?WHILE(*Bedingung*)?
Zeilen
PO?ENDWHILE[*Kommentar*]?

Prototyp: #[b#

Eine allgemeine Schleifenkonstruktion mit Bedingungsprüfung.

Solange *Bedingung* erfüllt ist, werden die *Zeilen* wiederholt.

3.3 Ablauf

Zu fast jedem (gültigen) *JobTyp* gehören die (*JobTyp*-spezifischen) Dateien *JobTyp*.job und *JobTyp*.htm.
JobTyp.job soll eine korrekte Jobdatei für *avanti* sein, allerdings ohne die festen Zeilen

& *Aufrufpfad*

am Anfang und

@ db=*Datenbank* id=*User*/*Password*
AVANTI:EOJ

am Ende, die automatisch von *populo* generiert werden.
JobTyp.htm ist eine korrekte HTML-Datei, insbesondere ohne die Bestandteile

Content-type: text/html
<*Leerzeile*>

des HTTP-Headers.
Zusätzlich können beide Dateien eine Reihe von Variablen und Befehlen enthalten, die den konkreten Programmablauf steuern. Variablen und Funktionen können dabei direkt im jeweiligen Text stehen, Routinen und sonstige Steuerbefehle sollten dateispezifisch auskommentiert sein (also mit '//' bei *.job*-Dateien und eingeschlossen in '<!-...->' bei *.htm*-Dateien).
Eine Anfrage wird nun wie folgt bearbeitet:

1. Die Eingabe wird interpretiert, insbesondere das Feld db wird ausgewertet und in die Variable *Db* überführt.

2. Zu *Db* wird die Konfigurationsdatei *Db.pl* geladen.

3. *JobTyp* und *JobSubTyp* werden (unter Rückgriff auf ein eventuelles Default) aus den HTTP-Feldern `t_...` bestimmt.
4. unter Benutzung der Pfadliste aus `$pathpraefix` wird die zugehörige Templatedatei *JobTyp.pl* geöffnet.
5. Die Templatedatei wird zeilenweise interpretiert und der Auftrag für *avanti* generiert. Besteht dieser nur aus Kommentarzeilen (sinnvoll beim Aufbau von Formularen oder der Anzeige von Hilfsseiten), wird er nicht abgeschickt, ansonsten an *avanti* übermittelt.
6. Das Resultat des Auftrags wird zeilenweise zurückgelesen und dabei entweder **PO:...:**-Konstruktionen aufgelöst oder entsprechend den deklarierten *LineTyp*en und den im *perl*-Hash `%LineFlow` benannten Routinen vorverarbeitet.
7. ist die Variable *RedoJob* gesetzt worden, so wird bei 4. fortgefahren.
8. Die Verbindung zu *avanti* wird beendet.
9. Nicht gecachte Kurztitelzeilen werden direkt geladen, falls erlaubt.
10. Das Resultat wird zwischenverarbeitet, um ggfls. Kurztitelzeilen in Registerauszüge einzusetzen oder Links im Ergebnis (**LI:**-Notation) aufzubereiten.
11. Der HTTP-Header für die Ausgabe wird gesendet.
12. Anhand des aktuellen *JobTyp* und unter Benutzung der Pfadliste aus *$pathpraefix* wird die Templatedatei *JobTyp.htm* für die Ausgabe geöffnet.
13. Die Templatedatei wird zeilenweise interpretiert und ausgegeben. Die vorverarbeiten Zeilen des Resultats von *avanti* werden dabei durch die **LOOP**-Konstruktionen aufgebraucht.
14. Eventuell angefallene Debug-Informationen werden ausgegeben[3].

3.4 alphabetische Liste der reservierten Namen

3.4.1 AddResult (Routine)

`PO&AddResult(`<*LineTyp*>`,` <*Inhalt*>`)&`
Prototyp: `#ee#`

Emuliert *avanti*-Ausgabe.

Inhalt wird in das Resultat als Zeile vom Typ *LineTyp* eingefügt.
Beispiel: Beim Blättern in Ergebnismengen wird direkt auf die Kurztitelliste zugegriffen, dabei kommt es nur darauf an, nach dem Aufruf von **Paginate** den Teilbereich der Satznummern, wie er in *P_Recno* enthalten ist, so unterzubringen, daß später automatisch die Kurztiteleinträge geholt werden. Nach Ausführung etwa von

`//PO&Paginate(`*PageNo*`,`*MaxShowResult*`,`*JobSubtyp*`,`*Recnums*`)&`

sind dabei äquivalent:

```
find # PO!P_Recno!
write "PO:Set(LineTyp,3):" newline
```

[3] Misfeature: erst hinter `</BODY>`!

```
list recnum
```

bzw.

```
find # PO!P_Recno!
write "PO:Set(LineTyp,13):" newline
list internal
```

und

```
//PO&AddResult(13, P_Recno)&
```

3.4.2 AHtm (Funktion)

`PO!HtmA(`*<Argument>*`)!`

Prototyp: `#e#`

wandelt das *Argument* von ASCII nach HTML (7-bittig) um.

...etwa weil man die Konfigurationsdateien unbedingt mit DOS-Umlauten beschicken möchte... Für ANSI-Werte siehe 3.4.29.

3.4.3 Append (Routine)

`PO&Append(`*Variable* `[,`*<Wert>* `[,...] ])&`

Prototyp: `#ve+#`

Hängt an den Wert von *Variable* die Ergebnisse der Auswertungen der *Werte* durch `POeval` an (alles nur Zeichenketten). Gab es *Variable* noch nicht, wird sie vorher mit '' initialisiert.

3.4.4 Assign (Routine)

`PO&Assign(`*Variable*`,` *<Wert>*`)&`

Prototyp: `#v[e+#`

Weist *Variable* das Ergebnis der Auswertung von *Wert* durch `POeval` zu.

3.4.5 AvantiErrHook (Hook)

Aus dem Resultat identifizierte Avanti-Fehlermeldungen werden an diese Routine übergeben, sofern sie definiert ist.

Beschreibung siehe 6.2.1.

3.4.6 AvantiErrTrans (Konfigurationsvariable)

```
%AvantiErrTrans = (
'130' => 'kein Ergebnis bei', ...
)
```

Für die in der Variablen aufgeführten Errorcodes von Avanti wird der Text bis zum ersten Doppelpunkt durch den angegebenen Text ersetzt.

3.4.7 AvantiHost (Konfigurationsvariable)

```
$AvantiHost = '127.0.0.1';
```

```
$AvantiHost = 'avanti.biblio.etc.tu-bs.de';
```

IP-Nummer oder DNS-Name des zu befragenden Avanti-Servers.

3.4.8 AvantiPort (Konfigurationsvariable)

```
$AvantiPort = 4949;
```

Portnummer des zu befragenden Avanti-Servers.

3.4.9 CheckMe (Funktion)

`PO!CheckMe(`<*Nummernvorrat*>`,` <*Testvorrat*>`)!`

Prototyp: `#ee#`

Ist (eine der Satznummern aus) *Testvorrat* in *Nummernvorrat* enthalten?

Liefert den Rückgabewert '`1`' (wahr), wenn mindestens eine der Satznummern aus *Testvorrat* auch in *Nummernvorrat* vorkommt, ansonsten '`0`' (falsch).
Einsatz typischerweise in den Bedingungs-Argumenten von Kontrollstrukturen.

3.4.10 CheckPattern (Funktion)

`PO!CheckPattern(`<*Zeichenfolge*>`,` *Teststring*`)!`

Prototyp: `#ee#`

Ist *Teststring* in *Zeichenfolge* enthalten?

Im Prinzip obsolet, da IF-Konstruktionen Zeichenkettenvergleiche mittels regulärer Ausdrücke erlauben.

Liefert den Rückgabewert '`1`' (wahr), wenn *Teststring* im ersten Argument *Zeichenfolge* vorkommt.
Einsatz typischerweise in den Bedingungs-Argumenten von Kontrollstrukturen.

3.4.11 Collect (Routine)

`PO&Collect(`*Struktur*`,` *feldmuster*`,` *Label*`)&`

Prototyp: `#bbb#`

Faßt bestimmte an *populo* übergebene Parameter (HTTP-Felder) in einer *Struktur* zusammen.

Erzeugt in der Struktur *Struktur* zu allen HTTP-Feldern, die mit *feldmuster* beginnen, Schlüssel mit dem Namen des HTTP-Feldes ohne den Anfang *feldmuster*. Der Wert des jeweiligen HTTP-Feldes wird diesem neuen Schlüssel als Wert zugewiesen.
Gab es die Labels `Key` und *Label* noch nicht, so werden sie an eine bestehende Liste zu **#Labels#** angehängt. Gab es den Schlüssel **#Labels#** noch nicht, so wird einer initialisiert. Gab es die Struktur *Struktur* noch nicht, so wird sie hiermit erzeugt (wohl der Normalfall).
Beispiel: Ein Abfrageformular stellte fünf Eingabefelder zur Verfügung mit jeweils einem korrespondierenden Pulldown-Menue, in dem der Benutzer den „Typ“ seiner Eingabe auswählen

kann. Diese Felder haben im Formular die Namen 'text_1' bis 'text_5' für die Textfelder und 'pull_1' bis 'pull_5' für die korrespondierenden Pulldowns. Den Nummern 1 bis 5 entspreche die Deklaration

```
@IndexSystem = ('1', '2', '3', '4', '5');
```

in der Konfigurationsdatei.
Die folgende Jobdatei sammelt diese Informationen dann wie folgt ein:

```
//PO&Collect(Eingabe,text_,Begriff)&
//PO&Collect(Eingabe,pull_,Register)&
```

und später kann dann so darauf zugegriffen werden:

```
Gesucht werden soll:
PO?STRUC(Eingabe,IndexSystem)?
PO!S_Begriff! aus Index PO!S_Register!
PO?ENDSTRUC?
```

3.4.12 CollectQuery (Funktion)

`find PO!CollectQuery(`<*Logik*>`,` <*Trunkierungszeichen*>`)!`

Prototyp: `#ee#`

Setzt die Suchanfrage für *avanti* zusammen.

Aus der Struktur `Query` mit Indexsystem `QueryNummern` wird entsprechend der gewählten Verknüpfungs-*Logik* und dem *Trunkierungszeichen* eine Suchanfrage für *avanti* zusammengesetzt (als Rückgabewert dieses Befehls). Suchbegriffe, zugehöriges Register, Verknüpfungslogik und die Trunkierung können vorher noch durch in der in *Db.pl* im Hash `MaskenSpecial` angegebene Unterprogramme modifiziert werden. Die modifizierten Suchbegriffe werden in der Struktur `Query` unter dem jeweiligen Schlüssel unter dem Label `Query` gespeichert.

3.4.13 ContentType (Variable)

`//PO&Set(ContentType,text/x-sgml)&`

Vorbelegung für HTTP-Header.

Vor der Ausgabe wird durch **PrintHeader** der HTTP-Content-Type angegeben. Belegen von **ContentType** gibt statt des Defaults 'text/html':

`Content-type:` *ContentType*
Leerzeile

3.4.14 DateForm (Konfigurationsvariable)

`sprintf($DateForm, $mday, $mon, $year+1900, $hour, $min);`

`printf(3)`-Template für das Datumsformat in *Modified* (3.4.48) und *DbTime* (3.4.17).

3.4.15 Db (Variable)

`PO!Db!`

Die Variable **Db** dient zur Identifikation der Datenbank und wird automatisch von *populo* aus dem Namen des Konfigurationsskripts oder dem HTTP-Feld `db` bestimmt.

3.4.16 DbSubmitted (Variable)

`PO!DbSubmitted!`

Die Variable **DbSubmitted** dient zur Identifikation der Datenbank und wird automatisch von *populo* aus dem HTTP-Feld `db` oder aus **Db** (3.4.15) bestimmt.

3.4.17 DbTime (Variable)

`PO!DbTime!`

Datum der letzten Änderung der Konfigurationsdatei für die aktuelle Datenbank *Db*-Datei.

Tip: Wenn Sie eine separate Datenbank haben, die regelmäßig aktualisiert wird, können Sie nach dem Auflegen der neuen Datenbank den Datumsstempel der `.pl`-Datei mit

`touch -r` *Pfad/Db*`.TBL` *Db*`.pl`

auf den der Datenbank setzen. In den `.htm`-Templates können Sie dann mit `PO!DbTime!` stets das „Datum der Datenbank“ einblenden.

3.4.18 Debug (Konfigurationsvariable)

```
$Debug = 1;
```

Falls gesetzt, werden diagnostische Meldungen aus den Kategorien entsprechend **WantDebug** (3.4.79) ausgeben.
Eine Beschreibung der definierten Konfigurationsklassen findet sich im Abschnitt 5.5 über Debugmöglichkeiten.

3.4.19 Declare (Routine)

`PO&Declare(`<*Variable*> `[,`<*Variable*> `[,...] ])&`

Prototyp: `#v+#`

Die angegebenen Variablen werden eingeführt. Wichtig ist dies vor Schleifenkonstruktionen, da bei Variablen, die nur innerhalb von Schleifen vorkommen, die Heuristik die Variablennamen fälschlich als Texte erkennt (auf Meldungen *Scalar failure* achten).

3.4.20 Defaults (Struktur)

```
%Defaults = (
JobTyp => 'maske',
'index' => 'PER', ...);
```

Diese Struktur ist in der Konfigurationsdatei definiert und definiert Vorgaben für den Fall, daß HTTP-Felder nicht besetzt sind.

3.4.21 Dec (Hilfsfunktion)

`PO!Dec(`<*Argument*>`)!`

Prototyp: `#e#`

Erniedrigt den (ganzzahligen) Wert von *Argument* um 1.

Dies ist nützlich für einfache Arithmetik, etwa bei Jahreszahlenbereichen für Suchbegriffe. *avanti* versteht aber inzwischen auch '`<=`' und '`>=`' bei der Formulierung von Restriktionen.

3.4.22 EndeRes (Hilfsunktion)

`PO!EndeRes(`*Präfixtext*`,`*Bereich*`))!`

Prototyp: `#ee#`

Falls sich aus *Bereich* eine (vierstellige) Jahreszahl als „Endejahr“ identifizieren läßt, wird diese mit vorangesetztem *Präfixtext* sowie Operator (’=’, ’<=’, ’<’) zurückgegeben, falls erforderlich mit führenden Nullen.

Korrespondiert zu **StartRes** (3.4.72). Nützlich, falls Jahreszahlenbereiche in *einem* Feld einzugeben sind, aber zwei Restriktionen ergeben sollen.
Beispiel bei 3.4.72.

3.4.23 ENV (Funktion)

`PO!ENV(`*Environmentvariable*`)!`

Prototyp: `#b#`

Zugriff auf Environment

Gibt den Inhalt der Environmentvariablen *Environmentvariable* zurück, falls diese belegt ist, sonst ’’.

3.4.24 Everything (Funktion)

`PO!Everything(`<*feld*> `[,`<*feld*> `[,...] ])!`

Prototyp: `#[b+#`

Alle Eingabewerte in URL-encodierter Form (auch wenn ursprünglich als Formular mittels POST übertragen oder interaktiv erfragt), mit *Ausnahme* der Felder, deren Namen als Argumente aufgeführt sind.

Vorsicht: Dies mag für manche Server oder Browser viel zu lang sein.
In einem gewissen Sinn ist die Funktion *Everything* komplementär zur Routine *LinkIni* (3.4.44).

3.4.25 Fetchit (Funktion)

`find # PO!Fetchit(`<*Force*>`)!`

Prototyp: `#e#`

sortierte Liste der Satznummern, zu denen noch keine Kurztitelzeilen zwischengespeichert sind (Argument <*Force*> = 0), bzw. aller Satznummern zu Kurztitelzeilen (Argument <*Force*> = 1).

Eingelesene Kurztiteleinträge werden intern zwischengespeichert.

3.4.26 Future (Funktion)

`PO!Future(`<*Zeitoffset*>`)!`

Prototyp: `#b#`

<*Zeitoffset*> ist dabei eine Folge von positiven oder negativen ganzen Zahlen mit angehängtem `’’s’’`, `’’m’’`, `’’h’’`, `’’d’’` oder `’’w’’` (wobei auch Großbuchstaben erlaubt sind).

Gibt eine Zeitangabe nach RFC 1123 zurück, die um <*Zeitoffset*> in der Zukunft liegt.
Beispiel: Folgender Code produziert einen Expires-Header, der „Gültigkeit 23 Stunden ab Absenden“ besagt:

```
<META HTTP-EQUIV="expires" CONTENT="PO!Future(+1D -1h)!">
```

3.4.27 GetBrowser (Funktion)

`PO!GetBrowser()!`
Prototyp: `##`

Auswertung von HTTP_USER_AGENT.

Gibt einen Namen für den benutzten Browsers zurück, falls dieser identifiziert werden konnte (z. Z. nur ’Internet Explorer’ und ’Netscape’).

3.4.28 GetRecnums (Routine)

`PO&GetRecnums(`*Variable*`, [`<*feld*>`, [`<*Extra*>`]])&`
Prototyp: `#v[be#`

Allegro-Satznummern verarbeiten

Bereitet die Datensatznummern aus dem HTTP-Feld *feld* (und der optionalen Angabe in *Extra*) für *populo*-interne Bedürfnisse auf und weist sie der Variablen *Variable* zu. Die Anzahl erhaltener Datensatznummern wird in der Variablen ***Variable*num** hinterlegt (die vorher vom Anwender deklariert werden muß).

3.4.29 Htm (Funktion)

`PO!Htm(`<*Argument*>`)!`
Prototyp: `#e#`

wandelt das *Argument* von ANSI nach HTML (7-bittig) um.

Vgl. **AHtm** (3.4.2) für ASCII-Werte.

3.4.30 Inc (Hilfsfunktion)

`PO!Inc(`<*Argument*>`)!`
Prototyp: `#e#`

Erhöht den (ganzzahligen) Wert von *Argument* um 1.

Dies ist nützlich für einfache Arithmetik, etwa bei Jahreszahlenbereichen für Suchbegriffe. *avanti* versteht aber inzwischen auch ’`<=`’ und ’`>=`’ bei der Formulierung von Restriktionen.

3.4.31 IniVar (Routine)

`PO&IniVar(`*Variable*`)&`
`PO&IniVar(`*Variable*`, `*feld*`)&`
`PO&IniVar(`*Variable*`, [`*feld*`], `*Struktur*`)&`
Prototyp: `#v[bb#`

Nur falls Variable noch nicht mit einem Wert belegt ist:

Sorgt für die Initialisierung einer benötigten *Variable* aus dem HTTP-Feld *feld*, notfalls mit einem geeigneten Default-Wert.

Die Routine versucht, der *Variable* einen der folgenden Werte zuzuweisen (in dieser Reihenfolge)[4]:

1. Wert des HTTP-Feldes *feld*
2. den **#Default#**-Wert der Struktur *Struktur*
3. den Vorgabewert für *feld* aus dem in der datenbankspezifischen Datei *Db.pl* definiertem Hash **%Defaults**
4. den Leerstring ''

3.4.32 INP (Funktion)

`PO!INP(`*HTTP-Feld*`)!`
Prototyp: `#b#`

Zugriff auf Eingabe

Gibt den Inhalt des Eingabefelds *HTTP-Feld* zurück, falls diese belegt ist, sonst das Default aus der Konfigurationsdatei, sonst ''.

3.4.33 Int (Funktion)

`PO!Int(`<*Argument*>`)!`
Prototyp: `#e#`

wandelt das *Argument* in eine ganze Zahl um

Insbesondere werden führende Nullen entfernt.

3.4.34 Iso (Funktion)

`PO!Iso(`<*Argument*>`)!`
Prototyp: `#e#`

wandelt das *Argument* von Ostwest-Windows nach ISO 8859-1 um.

Vgl. 3.4.29 für gleichzeitiges Konvertieren nach HTML.

3.4.35 JobSubtyp (Variable)

`PO!JobSubTyp!`
siehe PO!JobTyp!

3.4.36 JobTemplExt (Variable)

`PO&Set(JobTemplExt, .`<*String*>`)&`
Extension der *Job-Templates*, Default ist *.job*.

[4] Falls eine angegebene *Struktur* keinen Schlüssel **#Default#** hat, wird u. U. ein undefinierter Wert zugewiesen!

3.4.37 JobTyp (Variable)

`PO!JobTyp!`
`PO!JobSubtyp!`

Die Variable *JobTyp* bezeichnet die auszuführende Aktion, und korrespondiert mit *.job* und *.htm*-Dateien, *JobSubtyp* bietet eine Möglichkeit, zusätzliche Information weiterzugeben.

Der Wert von *JobTyp* wird aus den HTTP-Feldern bestimmt, die mit '`t_`' beginnen:
Kommt ein Feld `t_`*Text1*`_`*Text2*`=`*Wert* vor, so wird *Text1* der Variablen **JobTyp** und *Text2* der Variablen **JobSubtyp** und zugewiesen. Danach erfolgt die weitere Ausführung mit der (datenbankspezifischen) Jobdatei *JobTyp.job*.
Während der Vorverarbeitung der Job-Datei oder beim Zurücklesen des Resulats kann an **JobTyp** zugewiesen werden, die Präsentation der Ausgabe erfolgt dann mit der *.htm*-Datei, die dem aktuellen Wert von **JobTyp** entspricht (siehe auch PO!RedoJob!).
Da viele WWW-Browser ein Formular auch dann abschicken, wenn keiner der Submit-Knöpfe vom Anwender betätigt wurde, gilt folgende Zusatzregel:
Ein Feld `t_`*Typ* mit Wert `default` oder `ignore` wird nur dann berücksichtigt, wenn kein anderes Feld `t_...` übergeben wurde.
Für Java-Script-Einsatz mit mehr hidden fields gilt folgende Zusatzregel:
Ein Feld `t_`*Typ* wird nur dann berücksichtigt, wenn es einen Wert hat.
Beim Einsatz von graphischen Submit-Buttons werden typischerweise simultan zwei HTTP-Felder `t_`*Typ*`.x` und `t_`*Typ*`.y` übergeben, die Suffixe '`.x`' und '`.y`' werden von *populo* automatisch entfernt, dürfen umgekehrt aber auch nicht Bestandteil von Jobtypen sein.
Folgende reservierte *JobTyp*en dienen zum Abfangen von Fehlerzuständen:

`noans` Von *avanti* wurde kein (vollständiges) Resultat übermittelt.

`noconn` Es konnte nicht mit *avanti* verbunden werden.

`nojob` Es wurde keine *.job*-Datei zum spezifizierten *JobTyp* gefunden.

`notyet` Es konnte nicht mit *avanti* verbunden werden, der Start des Servers wird aber von *populo* initiiert, die Anfrage wird nach einigen Sekunden automatisch wiederholt.

Alle diese *JobTyp*en erfordern nur ein Ausgabetemplate (*.htm*-Datei), kein Jobtemplate *JobTyp.job*.

3.4.38 NoStat (Konfiguration)

`@NoStat = qw(`*<HTTP-Feld>* `...);`

Argument für **Everything** (3.4.24) im Kontext von **Stat** (3.4.73): Die aufgelisteten HTTP-Felder werden *nicht* in die für Auswertungszwecke in den *avanti*-Job codierte Liste der Aufrufparameter integriert.

3.4.39 L_ (Shortcut)

`PO!L_0!`
`PO!L_Recno!`

Unter diesem Shortcut sind die Elemente der Ergebniszeilen während der Ausführung einer **LOOP**-Schleife zugänglich (3.2.2)..

3.4.40 Lin (Funktion)

`PO!Lin(`<*Argument*>`)!`

Prototyp: `#e#`

wandelt das *Argument* in URL-codierte Form um (Als *Lin*k).

3.4.41 LineFlow (Konfiguration)

```
%LineFlow = (
'1' => \&DifferentiateRegister, # qrix, qrix title+
...
);
```

Entsprechend der Werte der Variablen **LineTyp** (3.4.42) werden die von *avanti* gelieferten Zeilen durch die hier definierten Unterprogramme weiterverarbeitet.

Beispiele und Vorbelegung siehe 6.3.1.

3.4.42 LineTyp (Variable)

`PO!LineTyp!`

Die Variable **LineTyp** dient zur Kennzeichnung der von *avanti* gelieferten Ergebniszeilen entsprechend ihrer Art.

Alle nach der Setzung von *avanti* gelieferten Zeilen werden dem jeweiligen Typ zugeordnet.

Vordefiniert sind:

0	unspezifiziert
1	Registerzeile
2	Verweisungszeile
3	Kurztitelzeile mit Satznummer
4	Datenanzeige (expandiert Links)
5	interne Satznummern (nicht anzeigen)
6-9	frei
11	Registerzeile, noch zu expandieren
12	Verweisungszeile, noch zu expandieren
13	Kurztiteleintrag
14	Kurztiteleintrag (nicht anzeigen)
15-19	frei (werden expandiert)
21	Expandierte Registerzeile
22-30	frei

Beispiel: Während der Bearbeitung der Demo-Datenbank `avdemo` ändern sich die Linetypen etwa wie folgt:

Fluss	JOB	->&waitjob	-> &expandjob
acindex	1	-> (1,2)	
acexpand	11	-> (11, 12)	-> (11[,12]),21
regsrch	5	->	-> 21
allegro	5	->	-> 21
result	3	-> 21	

3.4.43 LinkEscape (Hook)

In den Parameterdateien vorbereitete interne Links innerhalb der Datenbank werden ausgearbeitet.

Beschreibung siehe 6.2.3.

3.4.44 LinkIni (Hilfsroutine)

`PO&LinkIni(`*Variable* `[,`*feld* `[,...] ])&`
Prototyp: `#v[b+#`

Initialisiert *Variable* als URL mit den Werten der angegebenen *feld*er (enthält als Beginn den Programmnamen und ist schon korrekt escaped)

3.4.45 Localize (Routine)

`PO&Localize(`*Struktur*`,` *Schlüssel* `[,`*Shortcut*`])&`
Prototyp: `#be[b#`

Der Inhalt des Schlüssels *Schlüssel* der Struktur *Struktur* wird zugänglich gemacht (Ist *Schlüssel* der Name einer Variablen, wird ihr Wert genommen).

Dabei werden Elemente, für die es eine Entsprechung unter **#Labels#** von *Struktur* gibt, unter dem jeweiligen Namen aus **#Labels#** (mit vorangestelltem *Shortcut*, falls dieser angegeben wurde) als Variable zugänglich gemacht, alle anderen über ihre Position (nur falls *Shortcut* angegeben wurde)[5].
Beispiel:
Es war eine Struktur

```
%Hallo = (
#Labels#=> ['Name', 'Vorname'],
  Nachbar => ['Meyer', 'Ernst', 'Universitätsbibliothek',
    'Pockelsstr. 3', 'Braunschweig'],
  Unterbar=> ['Müller', 'Paul', 'Pockelsstr. 5',
    'Braunschweig'],
);
```

definiert und wir wollen auf die Informationen zu einer bestimmten Person zugreifen:

```
PO&Set(Wanted, Unterbar)&
PO&Localize(Hallo, Wanted, W_)&
```

Danach haben wir Zugriff auf:

```
Der Name ist PO!W_Vorname! PO!Htm(W_Name)!, die Adresse ist
PO!W_0!
PO!W_1!
PO?IF(W_2)?
PO!W_2!
PO?ENDIF?
```

was uns die Ausgabe liefert:

[5] *Shortcut* muß ein Großbuchstabe gefolgt von einem Unterstrich '_' sein. Hierdurch entstehen die indizierten Variablen.

```
Der Name ist Paul M&uuml;ller, die Adresse ist
Pockelsstr. 5
Braunschweig
```

Hätten wir im Beispiel den dritten Parameter `W_` ausgelassen, hätten wir nur Zugriff auf die ersten beiden Elemente von `Unterbar` gehabt, und zwar mit

`PO!Name! und PO!Vorname!`

3.4.46 MaskenSpecial (Struktur)

`%MaskenSpecial = ('PER' => 'Persify,Trunkify,SR',...);`

Diese Struktur ermöglicht es, beim Zusammenbauen von Suchbefehlen via **CollectQuery** in Abhängigkeit vom symbolischen Registernamen eigene Hooks für die Aufbereitung von Suchbegriffen anzugeben. Jeder der durch Komma getrennten Werte muß der Name eines in der Konfigurationsdatei definierten Hooks sein.

Beispiele siehe 6.3.2.

3.4.47 Message (Routine)

`PO&Message(`<*Wert*> `[,`<*Wert*> `[,...] ])&`

Prototyp: `#e+#`

Die *Werte* werden unter der aktuellen Seite in der Reihenfolge ihres Auftretens zusammen mit den von *populo* generierten Meldungen ausgegeben. Praktisch evtl. für das Debuggen von *.job*-Dateien.

3.4.48 Modified (Variable)

`PO!Modified!`

Datum der letzten Änderung der aktuellen *.htm*-Datei

siehe auch *DbTime*.

3.4.49 P_ (Shortcut)

```
PO!P_0!
PO!P_Recno!
```

Dies ist das Default-Shortcut für die Resultate von **Paginate** (3.4.51).

3.4.50 OutTemplExt (Variable)

`PO&Set(OutTemplExt, .`<*String*>`)&`

Extension der *Ausgabe-Templates*, Default ist *.htm*.

3.4.51 Paginate (Routine)

`PO&Paginate(`*<Ausgangsseite>*`, `*<Seitengröße>*`, `*<Aktion>*`, `*Nummern*` [,`*Shortcut*`])&`

Prototyp: `#eeee[b#`

Paginate unterteilt die durch ',' getrennten *Nummern* in Blöcke von jeweils *Seitengröße*, und initialisiert Variable für den Block, der sich aus *Ausgangsseite* durch Anwendung der *Aktion* (`reorder`, `first`, `previous`, `next` oder `last`) ergibt.

Shortcut ist optional, wird keiner angegeben, wird als Default `P_` genommen. (*Shortcut* muß aus einem Großbuchstaben gefolgt von einem Unterstrich '_' bestehen.)
Nummern muß der Name der Variablen sein, die die durch ',' getrennte Liste der internen Nummern anzuzeigender Datensätze enthält (als Zeichenkette).
Seitengröße muß der Name der Variablen sein, die die maximal zulässige Anzahl anzuzeigender Datensätze pro Ausgabeseite enthält. Diese beiden Werte zusammen bestimmen, welche Datensätze auf welcher Seite anzuzeigen sind.
Ausgangsseite gibt die zuletzt angezeigte Seite an, *Aktion* wie diese Seitenzahl zu verändern ist:

`first` erste Seite

`previous` oder **-** vorige Seite (Dekrement)

`next` oder **+** nächste Seite (Inkrement)

`last` letzte Seite

<***Zahl***> Seite wird explizit gesetzt.

Alle anderen Werte von *Ausgangsseite* führen zu keiner Modifikation des Ausschnitts.
Paginate bestimmt nach diesen Angaben:

***Shortcut*`No`** Nummer der neuen Seite (*Ausgangsseite* modifiziert)

***Shortcut*`Recno`** die Liste der (maximal *Seitengröße*) auf dieser Seite befindlichen Elemente aus *Nummern*

***Shortcut*`StartNo`** fortlaufende Nummer des ersten Satzes auf dieser Seite (von 0 gezählt)

***Shortcut*`EndNo`** fortlaufende Nummer des letzten Satzes auf dieser Seite (von 0 gezählt)

***Shortcut*`Tot`** Nummer der letzten Seite

***Shortcut*`IsFirst`** Ist die aktuelle Seite die erste?

***Shortcut*`IsLast`** Ist die aktuelle Seite die letzte?

3.4.52 Passwd (Konfigurationsvariable)

`PO!Passwd!`

Wird am Ende des *avanti*-Jobs als zweiter Wert für `id=.../...` übertragen.

3.4.53 PathInfo (Konstante)

`PO!PathInfo!`

Inhalt des `PATH_INFO`-Environments, d.h. von in der URL an den Scriptnamen per '/' angehängter weiterer Informationen.

3.4.54 Poke (Routine)

`PO&Poke(`*Struktur*`, <`*Schlüssel*`>, `*Label*`, <`*Wert*`>)&`
Prototyp: `#bebe#`

Ermöglicht die Veränderung (oder Erzeugung) der *Struktur*.

Schreibt in der Struktur *Struktur* den Wert *Wert* in die Liste zu dem Schlüssel *Schlüssel*, und zwar an die Stelle, die *Label* in der Liste zu dem Spezialschlüssel **#Labels#** einnimmt. (Gab es *Label* noch nicht in der Liste **#Labels#**, wird es hinten angehängt. Gab es noch keinen Spezialschlüssel **#Labels#**, so wird vorher einer initialisiert.)

3.4.55 Pop (Konstante)

`PO!Pop!`
Name der aktuellen Instanz von *populo*.

Wenn in den Ausgabe-Templates *.htm* alle URLs mit Bezug auf *populo* via

```
<a href="PO!Pop!..."> ...</a>
```

gebildet werden, sind diese Seiten immun gegen Umbenennung des Skripts `populo.pl`, Verlagerung in andere (virtuelle oder reale) Verzeichnisse oder Verlagerung auf einen anderen Server. Wird allerdings mit `PATHINFO`3.4.53 gearbeitet, so expandieren die Browser meistens falsch, und die Referenzen sollten mittels Voranstellung der vollen Pfadangabe *PopPath*3.4.56 erfolgen:

```
<a href="PO!PopPath!PO!Pop!PO!PathInfo!..."> ...</a>
```

3.4.56 PopPath (Konstante)

`PO!PopPath!`
Virtueller Pfad der aktuellen Instanz von *populo*.

3.4.57 Powered (Konstante)

`PO!Powered!`
Netter String `Allegro Avanti Populo` *Version*.

3.4.58 PrepareQuery (Routine)

`PO&PrepareQuery(`*Struktur*`, `*Indexsystem*`, `*Label1*`, `*Label2*`)&`
Prototyp: `#bbbb#`

Bereitet die Suchanfragen für *avanti* auf.

Indexsystem muß der Name einer in der Konfigurationsdatei definierten Liste sein, die Schlüssel zu der Struktur *Struktur* enthält,
Label1 und *Label2* müssen Labels von *Struktur* sein und tragen die Bedeutung „Register“ und “Suchbegriff“.
Aus den Rohdaten in *Struktur* werden unter *Label2* abgelegte Suchanfragen zum unter *Label1* abgelegten Register aufbereitet (getrennt). Das Ergebnis wird in einer neuen Struktur `Query` hinterlegt.
`Query` erhält die Labels `OrigKey`, `Register`, `Begriff`, `Treffer`, und `Query`. Jede isolierte Suchanfrage wird unter einem eigenen Schlüssel hinterlegt und dessen Rubrik `Treffer`

mit der Anzahl 0, Rubrik `Query` vorläufig noch gar nicht initialisiert. Die erzeugten Schlüssel werden im zugehörigen Indexsystem **QueryNummern**, deren Anzahl in der Variablen **Querynum** hinterlegt.

3.4.59 PrintVariablesShort (Funktion)

`PO!PrintVariablesShort(`*Perl-Hash*`)!`

Prototyp: ##

Gibt alle Schlüssel / Wertepaare des angegebenen assoziativen Vektors in einer HTML-aufbereiteten Form zurück.

3.4.60 Query (Struktur)

`write PO:Poke(Query,...,Treffer,Int( lastnum )): n`

Diese Struktur wird von **PrepareQuery** initialisiert und von **CollectQuery** benutzt. Eigene Einträge (etwa in das Feld `Treffer` können mit **Poke** hinzugefügt werden.

3.4.61 RealDb (Variable)

`PO!RealDb!`

siehe PO!Db!

RealDb wird vorbesetzt durch den Namen der Datenbank *Db* bereinigt um eventuelle Ziffern am Ende und kann außerdem durch Setzungen in der Konfigurationsdatei *Db* überschrieben werden. Diese Variable wird benutzt, um im Job die Datenbank für *avanti* anzugeben. Dies dient dazu, (testweise) leicht verschiedene Varianten vorzuhalten, die dieselben Template-Dateien benutzen und sich nur in den Initialisierungen *Db.pl* unterscheiden.

3.4.62 Recnums (Variable)

`PO!Recnums!`
`PO!Recnumsnum!`

Recnums enthält die Satznummern nach dem Einlesen von Zeilen vom *LineTyp* 5, getrennt durch ’`,`’.

Recnumsnum enthält die Anzahl der Elemente von Recnums.

Eine Zuweisung an diese Variable ist nicht erlaubt.

3.4.63 RedoJob (Variable)

`PO!RedoJob!`

Steuert erneute Verarbeitung von Jobs bzw. Verarbeitung weiterer Jobs.

Während der Vorverarbeitung der Job-Datei oder beim Zurücklesen des Resulats kann **RedoJob** auf ’`1`’ gesetzt werden, die (nochmalige) Verarbeitung des dem aktuellen Wert von **JobTyp** entsprechenden *.job*-Templates wird so erzwungen.

3.4.64 ResultHook (Hook)

Die aktuelle Zeile (vom *LineTyp* '4') ist in $_ und kann vor der Ausgabe noch individuell modifiziert werden.

Beschreibung siehe 6.2.4.

3.4.65 ReuseSocket (Konfigurationsvariable)

`$ReuseSocket = `*Zahl*`;`

Falls auf 0 gesetzt, wird pro Verbindung nur ein *avanti*-Job abgewickelt, ist *Zahl* negativ, wird die Verbindung theoretisch unendlich oft weitergenutzt, bei positiver *Zahl* wird sie *Zahl* oft wiederbenutzt.

Normalerweise sollte die Vorgabe (von 25) beibehalten werden, außer zum Testen von Kommunikationsproblemen. Da *populo* jedoch als CGI-Skript nicht persistent ist, gilt diese Nachnutzung der Verbindung nur innerhalb eines Aufrufs des Skripts, nicht darüber hinaus.

3.4.66 ReverseRegister (Konfiguration)

`%ReverseRegister = ('1' => 'PER', 'PER:' => 'PER')`

Diese Struktur ist in der Konfigurationsdatei definiert und definiert die Umsetzung von registerübergreifenden Verweisen und vorbereiteten Hyperlinks auf symbolische Registernamen von *avanti*.

Nur in der Form `'ABC:' => 'DEF'` hier aufgelistete symbolische Register *ABC* werden von *LinkEscape* (6.2.3) erkannt, d.h. nur solche Register können von Parameterdateien für Hyperlinks herangezogen werden.

3.4.67 S_ (Shortcut)

```
PO!S_0!
PO!S_Key!
```

Dies ist das Default-Shortcut für die Lokalisierungen während der Verarbeitung von Schleifen über die **STRUC**-Konstruktion (3.2.3).

3.4.68 SanifyRegister (Routine)

`PO&SanifyRegister(`*RegisterStruktur*`, `*RegisterSchlüssel*` [, `*Suchstring*`]&`

Prototyp: `#bb[e#`

Initialisiert verschiedene Variablen für ein Register. *RegisterStruktur* muß die Labels `RegStart`, `RegEnde` enthalten.

Diese Routine ist im Prinzip obsolet, da *avanti* inzwischen Pseudoregister transparent behandelt.

Die Routine kann eingesetzt werden, um vor der Recherche in einem allgemeinen Register den Suchbegriff auf die Registergrenzen (etwa 'A' – 'z' einzugrenzen, die Benutzung ist aber nicht zwingend.
Die Inhalt des Schlüssels *Registerschlüssel* der Struktur *RegisterStruktur* wird unter den Variablen **RegStart**, **RegEnde** zugänglich gemacht. Außerdem wird die Variable *Suchstring* so modifiziert, daß ihr Wert innerhalb der durch **RegStart** und **RegEnde** definierten Grenzen liegt.

Die Struktur *RegisterStruktur* muß in *Db.pl* definiert worden sein und als Schlüssel die symbolischen Registernamen für *avanti* enthalten sowie folgende **#Labels#** mit den Bedeutungen:

`RegStart` Kleinster erlaubter Eintrag im Register

`RegEnde` Größter erlaubter Eintrag im Register

Beispiel: Sei Register `|9I` für *avanti* symbolisch notiert durch `ISB`, so könnte sich dies in der Konfigurationsdatei durch einen Eintrag

```
%RegisterStruktur = (
  '#Labels#'=> ['RegStart', 'RegEnde'...],
...
    ISB => ['0', '999'...],
...
);
```

für *RegisterStruktur* manifestieren. Sinn macht dies aber nur, falls sich im Bereich `|9I` auch noch nicht-numerische Eintäge (evtl. unter einem anderen symbolischen Registernamen) befinden.

3.4.69 Set (Routine)

`PO&Set(`*Variable* `[,`*Wert*`])&`
Prototyp: `#v[e#`

Weist *Variable* den (konstanten) *Wert* zu, oder den Leerstring '', falls kein zweiter Parameter angegeben wurde.

3.4.70 SocketTimeout (Konfigurationsvariable)

`$SocketTimeout = `*Zahl*`;`
Anzahl der Sekunden, die *populo* auf (Teil-)Resultate von *avanti* wartet bzw. die *populo* versucht, einen Job an *avanti* abzusenden.

Das Timeout *Zahl* ist keine absolute Schranke für die Bearbeitungszeit, sondern ist eine Setzung für einzelne Teiloperationen: Nur wenn *Zahl* Sekunden absolut nichts an *avanti* geschickt oder von *avanti* gelesen werden kann, bricht *populo* die Verarbeitung ab.
Zu beachten ist, daß *avanti* selbst eine Konfigurationseinstellung für die Maximale Anzahl von Sekunden hat, die ein Job von Anfang bis Ende seiner Ausführung benötigen darf und daß der WWW-Client des Benutzers und evtl. der Webserver weitere, eigene Timeouts verwalten. `SocketTimeout` ist im Prinzip nur relevant in Situationen, wo *avanti* einen Job abbricht, aber die Verbindung nicht schließt.

3.4.71 SplitChecknums (Routine)

`PO&SplitChecknums(`*Variable*`, [`*feld*`, [<`*Extra*`>, [<`*Sortnums*`>]])&`
Prototyp: `#v[bee#`

Macht zunächst dasselbe wie **GetRecnums** (3.4.28), d.h. in *Variable* werden Satznummern aus den Feldern *feld* und ggfls. der Variablen *Extra* eingelesen.
Zusätzlich erfolgt ein Vergleich mit der internen Struktur `Thisnums` (die beim Einlesen der aktuell gelieferten Datensätze belegt wurde). Hierin auch enthaltene Satznummern werden in

der Variablen ***Variable*thisnum**, die anderen in der Variablen ***Variable*elsenum** hinterlegt. Die jeweiligen Anzahlen sind unter den Variablen ***Variable*thisnum** und ***Variable*elsenum** verfügbar. Alle Hilfsvariablen müssen vom Anwender vorher deklariert worden sein.
Ist das optionale, vierte Argument *Sortnums* angegeben, so werden die Satznummern in ***Variable*elsenum** noch einmal entsprechend der Nummernreihenfolge in *Sortnums* sortiert und aufgespalten in die Nummern ***Variable*prenum**, die *vor* dem ersten Auftreten einer der Nummern in *Sortnums* in der Menge `Thisnums` liegen und in ***Variable*postnum** mit den anderen Nummern. Die zusätzlichen Variablen ***Variable*prenum** und ***Variable*postnum** werden analog besetzt.
Beispiel: In einer Kurztitelliste sollen einzelne Sätze markiert werden können, bei Umsortierungen bleibt die Markierung erhalten, die Anzeige der Markierten Sätze soll entsprechend der aktuellen Sortierung der Kurzliste erfolgen.
Im ersten Job:

```
// Satznummern aus HTTP-Feld 'recnums' bestimmen
PO&GetRecnums(Recnums,recnums)&
// Sätze selektieren und sortieren lassen
find # PO!Recnums!
order ...
// Anzeige: „Es folgen viele Satznummern“
write "PO:Set(LineTyp,5):" newline
list internal
// Paginierung ausloesen
write "PO:Set(PageNo,1):" newline
write "PO:Paginate(PageNo,MaxShowResult,'first',Recnums):" newline
// Satznummern nach Avanti füttern und Kurztitel zeigen lassen
write "PO:Set(JobTyp,hickup):" newline
write "PO:Set(RedoJob,1):" newline
```

Beim Zurücklesen der durch *avanti* sortierten Satznummern wird durch den Aufruf von *Paginate* (3.4.51) eine Teilmenge zur „Seite“ 1 bestimmt, in einem zweiten Job (*hickup*) müssen nun die Kurztitel zu dieser „Seite“ herangeschafft werden:

```
find # PO!P_Recno!
// Anzeige: „Es folgen bekannte Satznummern mit Kurztiteln“
write "PO:Set(LineTyp,3):" newline
list recnum
//PO&Set(JobTyp,allegro)&
```

Im Ausgabetemplate *allegro.htm* wird nun die Verarbeitung der Ankreuzungen geregelt, es gibt viele Formularfelder namens *reccheck*, zunächst folgt ein verstecktes Feld mit den sortierten Satznummern aus den (nach Blätteroperationen möglichen) vorigen Seiten in *Reccheckpre*, dann die einzelnen Checkboxes mit korrekten Vor-Ankreuzungen, dann ein weiteres verstecktes Feld mit den sortierten Satznummern aus *Reccheckpost*. Die aktuellen Ankreuzungen sind (ebenfalls korrekt sortiert) in *Reccheckthis*:

```
<!- evtl. zusätzliche Ankreuzungen ->
PO&IniVar(Extracheck,extracheck)&
<!- Aufteilen entsprechend Reihenfolge aus Recnums ->
PO&SplitChecknums(Reccheck,reccheck,Extracheck,Recnums)&
...
<form action="PO!Pop!" method="POST">
<!- Gesamtmenge der Sätze aufbewahren ->
<input type="hidden" name="recnums" value="PO!Recnums!">
```

```
PO?IF(Reccheckelsenum)?
PO!Reccheckelsenum! weiter(e) Titel bereits markiert
PO?ENDIF?
...
PO?IF(Reccheckpre)?
<input type="hidden" name="reccheck" value="PO!Reccheckpre!">
PO?ENDIF?
...
PO?LOOP?
...
<!- zu einzelnem Titel entscheiden, ob bereits angekreuzt ->
<input type="checkbox" name="reccheck" value="PO!L_Recno!"
PO?IF(CheckMe(Reccheckthis,L_Recno))?
checked="checked"
PO?ENDIF?
>
...
PO?ENDLOOP?
...
PO?IF(Reccheckpost)?
<input type="hidden" name="reccheck" value="PO!Reccheckpost!">
PO?ENDIF?
...
</form>
...
```

3.4.72 StartRes (Hilfsunktion)

`PO!StartRes(`*Präfixtext*`,`*Bereich*`))!`

Prototyp: `#ee#`

Falls sich aus *Bereich* eine (vierstellige) Jahreszahl als „Startjahr“ identifizieren läßt, wird diese mit vorangesetztem *Präfixtext* sowie Operator (’=’, ’<=’, ’<’) zurückgegeben, falls erforderlich mit führenden Nullen.

Korrespondiert zu 3.4.22. Nützlich, falls Jahreszahlenbereiche in *einem* Feld einzugeben sind, aber zwei Restriktionen ergeben sollen.
Beispiel (in einem Job):

```
...
PO&IniVar(Restriktion)& // Leere Variable ’Restriktion’
PO&IniVar(Jres,jres)& // Jres aus Aufruf-Feld belegen
PO&Assign(Tmp, StartRes(’ and PYR’, Jres))&
// Tmp ist jetzt leer oder von der Form
// ’ and PYR>=JJJJ’
PO&Append(Restriktion, Tmp)&
PO&Assign(Tmp, EndeRes(’ and PYR’, Jres))&
// Tmp ist jetzt leer oder von der Form
// ’ and PYR<=JJJJ’
PO&Append(Restriktion, Tmp)&
// Restriktion enthaelt jetzt null bis zwei Restriktionen
```

```
...
```

3.4.73 Stat (Konfigurationsvariable)

```
$Stat = 1;
```

Falls gesetzt, wird der komplette QueryString (*Everything*3.4.24) als Kommentar im Job für *avanti* übertragen, so daß es in der Logdatei landet. Dabei werden die in **@NoStat** (3.4.38) aufgeführten Felder jedoch nicht mitprotokolliert.

3.4.74 Stl (Konfigurationsvariable)

Länge der Kurztiteleinträge der aktuellen Allegro-Datenbank. Diese Wert ist von der datenbankspezifischen Konfigurationsdatei *Db.pl* zu setzen.

3.4.75 STLmap (Hook)

Die Felder der gelesenen Kurzliste werden (für das Durchsortieren in erweiterten Registern) gemäß der hier programmierten Vorschrift in Sortierform gebracht.

Beschreibung siehe 6.2.5.

3.4.76 Timestamp (Variable)

`PO!Timestamp!`
: aktuelles Datum und Uhrzeit.

3.4.77 User (Konfigurationsvariable)

`PO!User!`
: Wird am Ende des *avanti*-Jobs als erster Wert für `id=.../...` übertragen.

3.4.78 Version (Konstante)

`PO!Version!`
: aktuelle Versionsnummer von *populo*.

3.4.79 WantDebug (Konfigurationsvariable)

`@WantDebug = qw(`*Loggingklasse* `...);`

Falls **$Debug** (3.4.18) gesetzt, werden diagnostische Meldungen entsprechend den in **Wantdebug** angegebenen Kategorien ausgegeben.
Eine Beschreibung der definierten Konfigurationsklassen findet sich im Abschnitt 5.5 über Debugmöglichkeiten.

Kapitel 4

Zeichensätze

4.1 Der Zoo

Browser und Webserver kommunizieren heutzutage mittels ISO 8859-1 als verabredetem Zeichensatz, sofern nicht UTF-8. Benutzereingaben und Werte aus Formularen sind dabei URL-encodiert, d.h. 7bittig, andere Zeichen sind als **%xy** codiert. Der Server liefert an den Browser HTML, 8bittige Zeichen aus ISO 8859-1 sind im Prinzip legal, bevorzugt werden aber die HTML-Entitäten dafür, fast alle Browser verstehen aber auch die Angabe von Unicode-Zeichen als **&#nnnn;**, wobei $nnnn$ auch größer als 255 sein darf.
allegro-Datenbanken sind typischerweise im proprietären OSTWEST-Zeichensatz codiert, die Umsetzungstabelle **o.apt** in den gleichfalls proprietären Zeichensatz allegro-Windows („allegro.ttf“) wird von *avanti* standardmäßig eingesetzt, wenn etwa Eingaben bearbeitet werden oder eine Kurzliste gezeigt wird. Die für die Vollanzeige von Datensätzen eingesetzten Parameterdateien produzieren HTML (bzw. ISO-8859-Zeichen mit HTML-Markup).
Wir haben nun also in folgenden Situationen folgende Zeichencodierungen, die geeignet umzusetzen sind:

Benutzereingaben via Browser: ISO 8859-1, URL-encodiert

Texte aus dem Konfigurationsskript: Undefiniert, vorzugsweise HTML oder ISO 8859-1 oder allegro-Windows.

Direkt gelesene Kurzzeileneinträge: OSTWEST

Unumcodierte Parameterausgabe mit *y0*: OSTWEST

von *avanti* gelieferte Kurzzeilen und Registerabschnitte: allegro-Windows

von *avanti* erwartete Suchbegriffe: allegro-Windows

für Sortierungen zu benutzen: Kleinbuchstaben und Ziffern

von Parameterdateien gelieferte Ausgabe: HTML oder ISO 8859-1

für den Browser produzierte Werte von Formularfeldern: HTML oder ISO 8859-1

zu produzierende Hyperlinks: ISO 8859-1, URL-encodiert

von Parameterdateien gelieferte Werte für *populo*-Hooks: OSTWEST oder HTML oder ISO 8859-1.

4.2 Die Realisierung

Interner Zeichensatz von *populo* ist der Zeichensatz allegro-Windows, im folgenden „win“ genannt.
HTTP-Felder (CGI-Übergabeparameter) werden stets von ISO 8859-1 nach „win“ konvertiert, dies regelt das interne Unterprogramm **iso2win**. Dieses Verhalten betrifft die Funktionen `PO!IniVar()!`, `PO!Collect()!` und `PO!INP()!`. Für im Konfigurationsscript oder im Job angegebene *Defaults* von Eingabefeldern wird *keine* Umsetzung gemacht, hier wird erwartet, daß sie bereits im Zeichensatz *win* codiert sind.
Die Funktion `PO!Htm()!` konvertiert von *win* nach HTML 4.0., dies regelt das interne Unterprogramm **win2htm**.
Direkt gelesene Kurzlisteneinträge aus der .STL-Datei (warum nicht von avanti liefern lassen?) werden automatisch mittels des internen Unterprogramms **dos2win**, das der *allegro*-Umcodierung durch **o.apt** entspricht, nach *win* umgewandelt. Dieselbe Umwandlung führt auch die Funktion `PO!AHtm()!` durch (deprecated).
Von *avanti* gelieferte Kurzlisteneinträge, erweiterte oder nicht erweiterte Registerabschnitte werden nicht umgewandelt (im Ausgabetemplate ist möglichst `PO!Htm()!` auf die Bestandteile anzuwenden).
Von *avanti* gelieferterte Parametrierte Ausgabe wird 1:1 durchgereicht, sollte also möglichst echtes HTML sein.
Die Funktion `PO!Lin()!` für die Produktion von Hyperlinks wandelt ihr Argument mittles des internen Unterprogramms **win2iso** nach ISO 8859-1 um und URL-encodiert das Resultat (`PO!LinkIni()!` hingegen codiert nicht um, da es direkt die übergebenen Felder benutzt).
Vorgaben für Formularfelder (auch Hidden Fields) sollten im Template möglichst mittels Behandlung durch `PO!Htm()!` vorbereitet werden, das löst auch das Problem mit aufeinandertreffenden Doppelanführungszeichen aus Template (Einschliessende Anführungszeichen für das Element-Attribut) und Daten (Anführungszeichen als Bestandteil der Daten).
Ein besonderes Problem sind die aus den in die Ausgabe eingebetteten Konstruktionen der Form **LI!...!...!** zu produzierenden Hyperlinks, die vom internen Unterprogramm **lnkescp** aufbereitet werden für die Weiterverarbeitung durch das vom Konfigurationsscript zu liefernde Unterprogramm **LinkEscape**. Der erste Bestandteil dieser Konstruktion ist ein optionaler Suchbegriff, der zweite Bestandteil die Präsentationsform des Hyperlinks. Dieser zweite Bestandteil wird von *populo nie* modifiziert, es wird erwartet, dass er durch die Parameterdatei bereits als HTML formatiert ist.

1. In der ausführlichen Form werden zwei Bestandteile angegeben, der erste Bestandteil ist ein Suchbegriff, der durch die Parameterdatei typischerweise nicht umcodiert (**y0**) erzeugt wird. *populo* wandelt den Suchbegriff zunächst mittles des internen Unterprogramms **dos2iso** nach ISO 8859-1, und URL-encodiert das Resultat anschliessend.

2. In der verkürzten Form muß *populo* den Suchbegriff aus der HTML-Darstellung des anzuzeigenden Textes zurückcodieren, dafür wird das interne Unterprogramm **htm2iso** benutzt, das berücksichtigt, dass die Parameterdatei HTML-Entitäten (wie **&**) erzeugt hat und (in manchen Fällen berücksichtigt, daß die Parameterdatei) evtl. auch numerische HTML-Codes für Zeichen ausserhalb des Bereichs von ISO 8859-1 (etwa **ă** für ein a mit Macron) produziert: Diese werden auf den Grundbuchstaben reduziert, um halbwegs verwertbare Suchbegriffe zu gewährleisten.

Schließlich wird für Sortierungen der Kurztitelliste das interne Unterprogramm **win2sort** eingesetzt.

4.3 Overrides

Bei einem anderen internen Zeichensatz als dem OSTWEST-Font von *allegro* müssen tendenziell alle internen Umcodierungsvorschriften ausgetauscht werden, im einzelnen sind dies:

1. **win2htm** für die „reguläre“ Aufbereitung durch `PO!Htm()!`
2. **iso2win** für das Einlesen der Benutzereingaben
3. **win2iso** als Grundlage für Hyperlinks `PO!Lin()!`
4. **win2sor** für Kurzlistensortierungen (wird nicht automatisch benutzt, sie muss durch **&STL-map** bzw. **&stlsort** im Konfigurationsskript aufgerufen werden).

   Der Name **ansi2sort** ist aus Kompatibilitätsgründen ein Alias für **win2sor**.
5. **dos2iso** als Grundlage für die Hyperlinks aus der Langform der **LI!...!...!**-Konstruktionen
6. **htm2iso** als Grundlage für die Hyperlinks aus der Kurzform der **LI!...!...!**-Konstruktionen
7. **dos2win** für direkte Kurztitelzugriffe und die (überflüssige!) Funktion `PO!AHtm()!`

Durch geschickte Formulierung von Jobs und der Ausgabeparameterdateien kann man auf die letzten beiden Umcodierungsarten verzichten.
Es ist also in diesem Jargon:

dos der zugrundeliegende Zeichensatz der Anwendung von *allegro*.

win der mit **o.apt** aus *dos* gemappte Arbeitszeichensatz von *avanti*, gleichzeitig der Arbeitszeichensatz von *populo*.

iso ISO 8859-1 als Kommunikationszeichensatz mit dem Benutzer

htm ISO 8859-1 und/oder HTML-Entitäten

Diese Routinen werden während der Laufzeit von *populo* definiert und können im Konfigurationsskript wie folgt umdefiniert werden:
%CustomTR ist ein Hash, dessen Werte anonyme Subroutinen sind, dabei ist der „Grundzustand“ theoretisch:

```
%CustomTR = (
   'win2htm' => \&tr_i1_ht,
   'iso2win' => \&tr_i1_al,
   'win2iso' => \&tr_al_i1,
   'win2sor' => \&tr_al_so,
   'dos2iso' => \&tr_ow_i1,
   'htm2iso' => \&tr_ht_i1,
   'dos2win' => \&tr_ow_al,
);
```

Dieser Grundzustand kann durch geeignete Setzungen und selbstdefinierte Routinen im Konfigurationsskript modifiziert werden. Analog zu `&tr_i1_ht` etc. aus `populo.pl` operieren diese Unterprogramme dabei auf **$_** und werden nur aufgerufen, wenn **$_** definiert ist.

Kapitel 5

Konfiguration

5.1 Aufsetzen von *populo*

Nachdem die Archive `poppop.lzh` mit `populo.pl` und `nodbpop.lzh` mit den Fehlerseiten in ein CGI-Verzeichnis entpackt wurde, sind ggfls. einige Anpassungen vorzunehmen: Dies betrifft alle *.htm*-Dateien `no`*xxx*`.htm` in diesem Verzeichnis, insbesondere den Verteiler in `nodb.htm`, falls nicht ein eigenes Startdokument erstellt wird.
Für den Aufruf von *populo* gibt es zwei Varianten:

1. Direkter Aufruf von `populo.pl` (oder einer Kopie davon), der *Name* der Datenbank muß dann im Aufrufparameter `db` enthalten sein, dazu korrespondierend muß dann *Name.pl* existieren.

2. Aufruf des Konfigurationsskriptes *Name.pl* für die Datenbank *Name*, das mit folgendem Code abschließt:

   ```
   if ( defined $Pop ) { # wir wurden aufgerufen
   1; und geben „Erfolg“ zurueck
   }
   else { # wir rufen selber auf
   $Db = 'Name';
   require („populo.pl“); # oder der heutige Name von populo
   }
   ```

   (vgl. auch den Abschnitt 5.3 über das Anbinden eigener Datenbanken)

5.2 Aufsetzen der Demo-Datenbank

Das Archiv `demopop.lzh` ist in das Verzeichnis mit `populo.pl` zu entpacken, so daß die Unterverzeichnisse `avdemo`, `cat` und `inmages` entstehen.
`cat` enthält die Templates für Jobs und Ausgabe, es braucht dem Server nicht bekanntgegeben zu werden.
`avdemo` enthält die Hilfsseiten für die Datenbank, sollen diese sichtbar sein, ist dieses Verzeichnis als `/avdemo` für WWW-Lesezugriffe freizugeben.

`popimg` enthält die GIFs für die Datenbank, sollen diese sichtbar sein, ist dieses Verzeichnis als `/popimg` für WWW-Lesezugriffe freizugeben (oder die Images sind in ein als `/popimg` freigegebenes Verzeichnis zu kopieren)..
Damit Direktzugriffe auf die Kurztiteltabelle funktionieren, ist die (dem Vorschlag bei der Installation von *avanti-w* entsprechende) Setzung

```
$DbPfad = "c:/avanti-w/avdemo";
```

eventuell anzupassen. *Es gibt aber normalerweise keinen Grund für Direktzugriffe auf die Kurztiteltabelle!*

5.3 Anbinden einer eigenen Datenbank

Empfehlenswert ist, zunächst unter dem gewünschten Namen eine Kopie der Beispielkonfigurationsdatei `avdemo.pl` zu erstellen und diese dann schrittweise zu modifizieren.
Wird eine Konfigurationsdatei *Name.pl* in das CGI-Verzeichnis gelegt, das auch `populo.pl` enthält, so ist dies als Anbindung zunächst ausreichend, allerdings müssen die folgenden Variablen und Unterprogramme in der datenbankspezifischen *.pl*-Datei definiert sein:

`$User` Username für *avanti*. Nutzbar als Variable.

`$Passwd` Password für *avanti*. Nutzbar als Variable.

`$pathpraefix` Unterverzeichnis mit den Template-Dateien für die Datenbank. Mehrere Verzeichnisangaben werden durch `;` getrennt und in der angegebenen Reihenfolge durchsucht.

Vorschlag: `$pathpraefix = '`*Unterverzeichnis*`/;./';`

`%Defaults` Voreinstellungen für HTTP-Felder, typischerweise mindestens mit dem Schlüssel `JobTyp` für die Einstiegsseite

Minimum: `%Defaults = (JobTyp =>'`*Begrüssungsseite*`');`

`%MaskenSpecial` Listen von selbstdefinierten Unterprogrammen für die Aufbereitung von Suchbegriffen für kombinierte Suchen.

Minimum: `%MaskenSpecial = ();`

Vgl. 6.3.2

`@Verweisungsformen` Mögliche Verweisungsformen in Registerauszügen.

Vorschlag: `@Verweisungsformen = ('s.a. ->', '->');`

`$Stl` Länge der Kurztiteleinträge, deaktivieren der Kurzliste via `$Stl = 0;`

`%STLexpand` Zustandsübergänge von Linetypen, vor allem beim manuellen Nachholen von Kurztitellisteneinträgen.

Minimum: `%STLexpand = ('5' =>` *Wert_größer_10*`);`

`%STLsuppresscaption` Automatische Unterdrückung von gewissen LineTypen:

Minimum: `%STLsuppresscaption = ('5' => 1);`

`sub OpenSTL` Öffnen der Kurztitelliste für Direktzugriffe (falls Direktzugriffe erforderlich sind, also eher nicht). Vgl. 6.1.2

sub CloseSTL Schließen der Kurztitelliste (falls Direktzugriffe erforderlich sind, also eher nicht) Vgl. 6.1.1

sub ParseSTLentry Aufbereiten der Kurztitelliste. Es wird oft reichen, `@STLStruktur` und `%STLPositionen` geeignet anzupassen. Vgl. 6.1.4

sub ParseRegisterline Aufbereiten von Registerzeilen, Zustandsübergänge Vgl. 6.1.3

sub LinkEscape Expandieren von Links aus parametrierten Vollanzeigen Vgl. 6.2.3

In Verbindung mit dem Standardunterprogrammen der Beispieldatenbank sind auch noch folgende Variable erforderlich:

$DbPfad Pfad der Datenbank für Direktzugriffe auf die .STL (Falls leer, sind Direktzugriffe verboten, die Kurztitelinformationen werden dann von *avanti* erfragt.)

$DbName Name der Datenbank für Direktzugriffe auf die .STL

%STLPositionen Positionen für die einzelnen Elemente der .STL in der Form *Name* `=>` `[`*Start*`,` *Länge*`]`.

Minimum: `%STLPositionen = (alles => [1,` *n*`]);`

@STLStruktur Indexsystem für `%STLPositionen`, Benennungen für Kurztitellistenelemente.

Minimum: `@STLStruktur = ('alles');`

%ReverseRegister Rückauflösungen von numerischen Registerbezeichnungen, wie sie in registerübergreifenden Verweisen in der Datenbank vorkommen, zu symbolischen Namen von *avanti*.

Minimum: `%ReverseRegister = ();`

Folgende Variable werden von `populo.pl` vorinitialisiert, können aber in der Konfigurationsdatei überschrieben werden:

$RealDb Name der Datenbank für *avanti*.

$AvantiHost IP-Adresse des *avanti*-Servers (für die aktuelle Datenbank)

$AvantiPort IP-Port des *avanti*-Servers (für die aktuelle Datenbank)

$ReviveCmd Auszuführendes Kommando, falls der Server nicht erreichbar ist.

In der Konfigurationsdatei der Beispieldatenbank werden diverse Konstanten gesetzt, auf die dann (als Variable oder Strukturen) durch die *.job*- und *.htm*-Dateien zugegriffen wird.

$DbLabel Überschriftszeile. Nutzbar als Variable.

$MaxShowResult Zeilenmaximum bei der Darstellung von Ergebnismengen

$MaxWriteThrough Bei weniger als so vielen Treffern soll automatisch in die Vollanzeige geschaltet werden.

%RegInfo Registerinformationen (Schlüssel sind die symbolischen Namen für *avanti*) mit Hilfstexten, Namen von Hilfsseiten, Überschriftszeilen.

@RegisterKeys Indexsystem für `%RegInfo`

%RegConstraints Eine Struktur, die den Vorgaben von `SanifyRegister` entspricht, d.h. zu den symbolischen Registern werden Start- und Endwerte sowie eventuelle Präfixe (bei Subregistern) definiert.

%VarInputDef Definition der Maskenfelder für das Formular „kombinierte Suche“

@MaskenVariable Indexsystem für `%VarInputDef`

@MaskenRegister Indexsystem für `%VarInputDef`

%Darstellung Erläuterung und Namen der Parameterdateien für Ausgabevarianten von Vollanzeigen

@ParamKeys Indexsystem für `%Darstellung`

%Sortierung Erläuterung und *avanti*-Sortierbefehl für Sortiervarianten in Kurzlisten und Vollanzeigen

SortKeys Indexsystem für `%Sortierung`

%Codierung Erläuterung und Name der Includetabelle für Zeichencodierungen bei Vollanzeigen

@CodierKeys Indexsystem für `%Codierung`

5.3.1 Anbinden der Kurztitelliste

Wird mit der Kurztitelliste der Datenbank operiert, müssen auch folgende Vorbereitungen getroffen sein:

$Stl Länge der Kurztitellisteneinträge

$DbPfad für Direktzugriffe auf die Kurztitelliste: realer Pfad der Datenbank

$DbName für Direktzugriffe auf die Kurztitelliste: realer Name der Datenbank

sub OpenSTL Öffnen der Kurztitelliste für Direktzugriffe, setzt `$STLisopen`. Vgl. 6.1.2

sub CloseSTL Schließen der Kurztitelliste für Direktzugriffe, setzt `$STLisopen` zurück. Vgl. 6.1.1

sub ParseSTLentry Kurztitelzeile holen und aufbereiten. Vgl. 6.1.4

sub stlsort Sortierroutine für Kurztiteleinträge (deprecated, Vgl. 6.2.6)

sub STLmap Aufbereitungsroutine für die Sortierung von Kurztiteleinträgen. Vgl. 6.2.5

@STLStruktur, %STLpositionen Konfiguration für `ParseSTLentry`. Vgl. 6.1.4

%STLsuppresscaption Festlegungen für die *LineTyp*e (größer 10), für die die Angabe der ursprünglichen Registerzeile unterdrückt werden soll.

%STLexpand Festlegungen für den *LineTyp* von separat beschafften Kurztitelzeilen.

Normalerweise sollten inzwischen Direktzugriffe auf die .STL nicht erforderlich sein, es gibt nämlich inzwischen auch für „erweiterte Register“ den *avanti*-Befehl `qrix title+`, der hinter jedem Indexeintrag die zugehörigen Kurztitel auswirft!

5.4 Hilfsdateien für festverdrahtete Zustände

Damit auch unvollständige Aufrufe sinnvoll funktionieren, ist eine „leere“ Datenbank `nodb` fest eingestellt. Diese benötigt folgende Dateien:

`Nodb.pl` Grundlegende Setzungen

`Nodb.job` Einziger (und leerer) Job

`Nodb.htm` Einzige Ausgabedatei (enthält typischerweise einen Verteiler für die angebotenen Datenbanken)

Um interne Fehler etc. abzufangen, gibt es eine Reihe von vordefinierten Jobtypen, die ggfls. automatisch besetzt werden:

`noconn` mit `Noconn.job` und **Noconn.htm**: Keine Verbindung.

`nojob` mit `Nojob.job` und `Nojob.htm`: Kein gültiger Jobtyp

Ist in `populo.pl` die Variable `$ReviveCmd` definiert (Vorgabe: `perl avwrest.pl`), so wird im Fall von „keine Verbindung“ ein

`Avwrest.pl` Neustart von *avanti*

versucht. Das kurze Perlscript `avwrest.pl` ist den lokalen Bedingungen anzupassen.

5.5 Debugging

`populo.pl` kann auch über die Kommandozeile gestartet werden. Nach dem Start werden über die Tastatur (oder via Umlenkung von STDIN mit ’<’!) Zeilen mit Paaren *Name=Wert* in genau dieser Form eingegeben, `<Strg>-Z` (oder das Dateiende) startet dann die Verarbeitung. Ist in `populo.pl` die Variable `$PopDebug::Allow` gesetzt, so ist des Protokollieren gewisser Zwischenergebnisse eingeschaltet, Welche hiervon ausgegeben werden sind, muß durch einen Aufruf von `PopDebug->init()` spezifiziert werden. Mögliche Parameter sind (in ’’ eingeschlossen und mit ’,’ getrennt anzugeben):

`Mess` Pleonastisch, von `populo.pl` automatisch freigegeben: Meldungen bezüglich erkannter Syntaxfehler

`Params` Anzeige der Übergabeparameter

`Showjob` Anzeige des erzeugten Auftrags für *avanti*

`Showresult` Anzeige des Original-Ergebnisses von *avanti*

`ShowSTL` Anzeige der explizit geholten Kurztitelzeilen

`Envir` Anzeige des Environments

`Conf` Anzeige der wichtigsten Konfigurationsinformationen

`Showcode` Anzeige des intern generierten Perl-Codes

`Parseval` Parsing-Informationen bei interpretierten Ausdrücken

Besser ist es jedoch, statt des Aufrufs von `PopDebug->init()` die gewünschten Parameter in der Konfigurationsdatei in das Array `@WantDebug` zu schreiben:

```
@WantDebug = (’Params’, ’Showjob’);
```

Mit dieser Konstruktion nämlich kann Debugging angefordert werden, bevor `populo.pl` (und damit das Modul `PopDebug`) überhaupt geladen sind.

5.6 Wichtige Routinen in `populo.pl`

5.6.1 POeval

Dieser Abschnitt ist obsolet. Die `POeval` existiert nicht mehr, Funktionen und Routinen besitzen „Prototypen“, anhand derer die Anzahl der benötigten Argumente und ihre Art (’b’ wie *bare*: Textstring wird erwartet, oder ’e’ wie *evaluate* oder *expression*: Variablenname oder Ausdruck wird erwartet) deklariert wird. Ein Unterschied besteht nur im ambivalenten Fall, daß eine Zeichenkette mit Großbuchstaben beginnt und nicht in einfache oder doppelte Anführungszeichen eingeschlossen ist: In diesem Fall gibt die Art des Prototyps an, ob es sich in jedem Fall um eine Zeichenkette handelt (’b’) oder ob eine Variable gemeint ist (’e’).

`POeval(`*<Argument>*`)`
: `POeval` sorgt für die korrekte Interpretation von *populo*-Sprachteilen. `POeval` ist ein *populo*-interner Befehl und sollte daher nur innnerhalb von selbst definierten Routinen oder Funktionen aufgerufen werden. (Solche Definitionen gehören selbstverständlich in die datenbankspezifische Datei *Db.pl*)

Alles innerhalb der Begrenzungen `PO!...!` und `PO:...:` wird als Argument an `POeval` weitergereicht, außerdem verschiedene Argumente einiger Routinen, Funktionen und Kontrollstrukturen. (Intern sind diese Argumente Zeichenketten.)
Zunächst entfernt `POeval` jeglichen Leerraum am Anfang und Ende des *Argument*es.
Ist *Argument* der Name einer gültigen *populo*-Variable[1], gibt `POeval` den Wert der Variablen zurück (u. U. auch einen undefinierten Wert, falls keine entsprechende Zuweisung stattgefunden hat.)
Enthält *Argument* eine Vergleichsoperation (`<`, `<=`, `==`, `!=`, `>=`, `>` für numerische Vergleiche und `lt` , `le` , `eq` , `ne` , `ge` , `gt` für Zeichenkettenvergleiche, oder `&&` bzw. `||` für logisches Und bzw. Oder, oder `=` bzw. `!` für Stringvergleiche mit regulären Ausdrücken), so wird *Argument* an der letzten dieser Stellen getrennt, beide Seiten einzeln mit `POeval` ausgewertet, und der Wahrheitswert der abschließend ausgeführten Vergleichsoperation zurückgegeben.[2]
Ist *Argument* eine mit `''` umschlossene Zeichenkette, so werden die `''` enfernt und der Rest zurückgegeben.
Ist *Argument* eine Routine oder Funktion, werden überflüssige Leerzeichen entfernt und die entsprechende Funktion ausgeführt. Zurückgegeben wird der Rückgabewert der Funktion. (Hierauf ist bei eigenen Befehlsdefinitionen zu achten!)
Trifft keine der obigen Bedingungen auf *Argument* zu, wird *Argument* unverändert zurückgegeben. Dies trifft insbesondere bei numerischen Argumenten von Vergleichsoperatoren zu, kann aber auch bei Syntaxfehlern auftreten.

5.6.2 QueryParse

`QueryParse(0)`

`QueryParse(1)`
: Umstrukturierung eines Suchstrings mit „AND“, „OR“ und „NOT“ in einen *avanti*-Suchbegriff, dieser enthält zur bequemen Weiterverarbeitung die Feste Zeichenkette **XxX-Reg-XxX** an allen Stellen, wo ein Registername für *avanti* einzusetzen ist.

[1] Gültige Variablen beginnen grundsätzlich mit einem Großbuchstaben
[2] Vorsicht: `POeval` reißt dabei u. U. auch Zeichenketten oder Befehle auseinander! Eine Klammerung ist derzeit nicht möglich.

In der zweiten Form wird die Stopwortliste in **$Stopwords** berücksichtigt, Default für **$Stopwords** ist eine aus **swl1.apt** gewonnene Wortliste, die jedoch um „and“, „nicht“, „not“, „oder“, „or“ und „und“ bereinigt wurde.

Es handelt sich hier nicht um „AND“, „OR“ und „NOT“ als binäre (boole’sche) Operatoren, sondern um unäre (Prefix-) Operatoren im Stil von Internet-Suchmaschinen!
Resulat der Umwandlung einer Benutzereingabe ist ein normierter Suchbegriff in den Formen (*Reg* steht hier stets für die Zeichenkette **XxX-Reg-XxX**)

```
Reg "AND-Begriff1" [AND Reg "AND-Begriff2" ...]
bzw.
Reg "OR-Begriff1" [OR Reg "OR-Begriff2" ...]
bzw.
( Reg "AND-Begriff1" [AND Reg "AND-Begriff2" ...] )
OR Reg "OR-Begriff1" [OR Reg "OR-Begriff2" ...]
bzw.
( Reg "AND-Begriff1" [AND Reg "AND-Begriff2" ...] )
NOT Reg "NOT-Begriff1" [NOT Reg "NOT-Begriff2" ...]
bzw.
( Reg "OR-Begriff1" [OR Reg "OR-Begriff2" ...] )
NOT Reg "NOT-Begriff1" [NOT Reg "NOT-Begriff2" ...]
bzw.
( ( Reg "AND-Begriff1" [AND Reg "AND-Begriff2" ...] )
OR Reg "OR-Begriff1" [OR Reg "OR-Begriff2" ...] )
NOT Reg "NOT-Begriff1" [NOT Reg "NOT-Begriff2" ...]
```

Beispielaufruf:

```
$Stopwords = join("|", qw(0 1 2 3 4 5 6 7 8 9 der die das la li lu));

\dots

%MaskenSpecial = ( # Unterprogr. z. Vorbeh. der Suchbegriffe
  STW => ’TitAndify’,
  \dots
);

\dots

sub TitAndify { # darf veraendern: $register $logik $trunk
  $trunk = "";
  &QueryParse(1);    # kill stopwords
  s/XxX-Reg-XxX/$register/g;
# alternativ:
# Simultan mit SR in FTX und $register suchen:
# s/XxX-Reg-XxX \"([^"]+)\"/ ( $register "$1" or FTX &"$1" ) /g;
  $register = "";
}
```

5.6.3 PrintHeader

`PrintHeader()`
Ausgabe des HTTP-Headers

Der Standard-Content-Type „text/html“ kann durch Setzen der Variablen `ContentType` modifiziert werden.

Kapitel 6

Hooks

Hooks sind definierte Schnittstellen, an denen in einer Datenbankkonfigurationsdatei definierte Unterprogamme ausgeführt werden.
Nachfolgend wird beschrieben, welche dieser Schnittstellen existieren, mit welchen Übergabeparametern sie von *populo* aufgerufen werden und welche Resultate es erwartet.

6.1 Vorgeschriebene Hooks

6.1.1 sub CloseSTL

Nur erforderlich, falls die Kurzliste mit **OpenSTL** (6.1.2) geöffnet wird.
Ordnungsgemäßes Schließen der Kurztitelliste, der Zustand wird in *$STLisopen* verbucht.

```
sub CloseSTL {
  return unless $STLisopen > 0;
  close (STL);
  $STLisopen = 0;
}
```

6.1.2 sub OpenSTL

Nur erforderlich, falls **$STL** ungleich 0 und die Kurzeinträge nicht via `qrix title +` bzw. `list recnum` durch *avanti* geliefert werden.
Ordnungsgemäßes Öffnen der Kurztitelliste, Fehler werden nach Möglichkeit abgefangen, der Zustand wird in *$STLisopen* verbucht.

```
  return '' unless (defined $DbPfad && $DbPfad);
  if ( open (STL, "<$DbPfad/$DbName.stl") ) {
      $STLisopen = 1}
  else {
      PopDebug->Mess("öffnen von $DbPfad/$DbName.stl"
                          . " hat nicht funktioniert!\n");
      $STLisopen = -1;
    };
  return '';
}
```

6.1.3 ParseRegisterLine

Die Routine wird beim Einlesen der *LineTyp*en 1 und 11 aufgerufen, Übergeben wird eine nur geringfügig aufbereitete Rohform, wie sie vom `qrix`-Kommando von *avanti* geliefert wird: Die drei Aufrufparameter sind

1. Trefferzahl
2. Registerzeile
3. Satznummern (durch ’,’ getrennt)

Zurückgegeben werden müssen ebenfalls drei Variable:

1. Eine Zahl, die auf *LineTyp* addiert wird
2. Ein Hash (mit den durch *ParseRegisterline* bestimmten Eigenschaften der Zeile, beispielsweise mit einem Key *Toomuch*)
3. Ein Array, das die (zur Ausgabe) bestimmten identifizierten Teile der Registerzeile enthält.

Eine triviale Routine sähe so aus:

```
sub ParseRegisterline {
  my ($count, $line, $recs) = @_;
  my $toobig = ($count =~ /^\>/); # Groesserzeichen bemerken
  $count =~ s/^\>//;              # Groesserzeichen entfernen
  return (0, {count=>$count, toobig=>$toobig}, [$_]);
}
```

6.1.4 ParseSTLentry

Diese Routine zerlegt Kurztitelzeilen in ihre Elemente. Einfaches Beispiel:

```
@STLStruktur = (’TitelAut’, ’Jahr’, ’Signatur’);

%STLPositionen = (
# Key           Start, Laenge
  TitelAut  => [1,    55],
  Jahr      => [58,   4],
  Signatur  => [64,   9],
);

sub ParseSTLentry {
  my ($localtyp, $line, $recno) = @_;
  my ($key, $value, $start, $len, @returnarray);
  foreach $key ( @STLStruktur ) {
      ($start, $len) = @{$STLPositionen{$key}};
      ($value = substr ($line, $start, $len)) =~ s/\s+$//;
      push (@returnarray, $value);
    };
  return ($localtyp, {}, [@returnarray]);
}
```

6.2 Optionale Hooks

Wenn diese Routinen nicht definiert sind, wird von *populo* ein Standardverhalten generiert.

6.2.1 AvantiErrHook

Aus dem Resultat von *avanti* identifizierte Fehlermeldungen werden an diese Routine übergeben, sofern sie definiert ist.
Die Übergabeparameter sind:

$num Numerischer Fehlercode von Avanti

$origtext Text der Originalzeile von der Fehlernummer bis zum ersten ':'

$num Rest der Originalzeile ab und einschließlich dem ersten ':'

Als Resultat muß die Routine einen (evtl. leeren) Ausgabetext zurückgeben.
Beispiel (Leitet alle Fehlermeldungen ans Ende des Dokuments um)

```
sub AvantiErrHook {
my ($num, $origtext, $extra) = @_;
PopDebug->Mess("\E$num\$origtext$extra\n");
return "";
}
```

6.2.2 InitHook

Wird aufgerufen, nachdem *populo* gestartet wurde, seine Zustandsavariablen initialisiert hat und die Aufrufparameter eingelesen wurden. Hier kann man also in Kenntnis des konkreten Aufrufs Nachinitialisierungen vornehmen, etwa um das Einlesen stets genutzter globaler Variablen nicht in jedem einzelnen Job veranlassen zu nüssen:

```
sub InitHook {      # !LI:<Register>:<Suchbegriff>!<Anzeige>!
  $Debug = $in{'debug'} ? 1 : 0;
}
```

Ein anderes Beispiel wäre eine Sprachumschaltung innerhalb des Scripts:

```
sub InitHook {      # !LI:<Register>:<Suchbegriff>!<Anzeige>!
  $Lang = $in{'lang'} || "";
  if ( $Lang =~ /^(de|en|fr|it)$/ ) {  # alle erlaubten aufzaehlen
      substr($pathpraefix, 0, 0) = "cat_$Lang/;";  # also cat_de ...
    }
}
```

6.2.3 LinkEscape

Ist dieser Hook definiert, so wird Ausgabe vom Typ 4 (Vollanzeige) auf schematische Flips / Hyperlinks der folgenden Form durchsucht (kein Zeilenumbruch!):

`...!LI:`

Qualifier

Register`:!`*Begriff*`! ...`

bzw.
...!LI:

Qualifier

Register:*Suchbegriff*!*Text*!...

Der optionale *Qualifier* sollte dabei analog zur Notation von Flips entweder leer oder '?' oder ' ' sein.
Register wird mittels der Struktur *ReverseRegister* aus der Konfigurationsdatei in ein *avanti*-Register übersetzt, *ReverseRegister* muß daher für jedes als Ziel von Links vorkommende symbolische Register *REG* eine Zuordnung

'REG:' => 'REG',

enthalten.
In der ersten Form wird *Begriff* von HTML nach ANSI zurückgerechnet und URL-enkapsuliert, in der zweiten Form wird *Suchbegriff* von ASCII nach ANSI verwandelt und dann URL-enkapsuliert. In der Praxis wird man also bei expliziter Angabe eines Suchbegriffs diesen durch die Allegro-Parameter mittels `y0` formatieren lassen.
Der Hook **LinkEscape** wird stets mit vier Parametern

Qualifier

Register

Suchbegriff

Anzeigebegriff

aufgerufen und muß einen fertig formatierten Link

...

zurückliefern.
Beispiel (aus `avdemo`):

```
sub LinkEscape {      # !LI:<Register>:<Suchbegriff>!<Anzeige>!
  my ($qualify, $register, $suchbegriff, $anzeige) = @_;
  local ($_);

  # Typ 'acindex' fuer ?-Begriffe, 'allegro' sonst
  my ( $ty ) = $qualify ? 'acindex' : 'allegro';
  if ( $ty eq 'acindex' ) {
          # JobTyp 'acindex' mit Parametern index u. s1
      $_ = "&index=$ix&s1=$suchbegriff"}
  else {  # JobTyp 'allegro' mit Parameter f_Reg
      $_ = "&f_$ix=$suchbegriff"};

  return "<a href=$Pop?db=$Db&t_$ty=x" . $_ . ">$anzeige</a>";
}
```

6.2.4 ResultHook

Die aktuelle Zeile (vom *LineTyp* '4') ist in `$_` und kann vor der Ausgabe noch individuell modifiziert werden.
Keine Rückgabeparameter, `$_` wird modifiziert.

Beispiel (URLs aus Datensätzen ausarbeiten):

```
sub ResultHook {
s!URL: http:(//
                                    /−?. : \w
+)!URL: <a href="$1">http:$1</a>!g;
}
```

6.2.5 STLmap

Diese Routine sortiert Kurztitelzeilen, wenn sie direkt (etwa im Zusammenhang mit einem erweiterten Register) geholt wurden. Die ersten Zeilen sollten analog gestaltet werden, dabei entsprechen `$$aaref[`i`]` und `$$bbref[`i`]` den Elementen der Kurztitelzeile, so wie sie in `@returnarray` von `ParseSTLentry` (6.1.4) zurückgeliefert wurden.
Die Einträge sind ANSI-Codiert, sie können mittels Aufruf von `ansi2sort` aufbereitet werden.
Diese Routine übernimmt die Funktionalität von `stlsort`.
Übergeben wird eine Referenz auf ein Array, dessen 2. Element das Array mit den Feldern des aktuellen Kurzlisteneintrags ist.
Zurückgegeben werden muß ein String.
Beispiel:

```
# $_[0] ist Referenz auf Array...
   my $ref = ($_[0])->[2];
# sortiere nach Verfasser / Titel / Jahr absteigend
   return join("\x01", ansi2sort($$ref[1]),
                       ansi2sort($$ref[0]),
                    (~ $$ref[2])
               );
}
```

6.2.6 sub stlsort

Diese Routine sortiert Kurztitelzeilen, wenn sie direkt (etwa im Zusammenhang mit einem erweiterten Register) geholt wurden. Die ersten Zeilen sollten analog gestaltet werden, dabei entsprechen `$$aaref[`i`]` und `$$bbref[`i`]` den Elementen der Kurztitelzeile, so wie sie in `@returnarray` von `ParseSTLentry` zurückgeliefert wurden.
Ist die Routine `STLmap` definiert, so wird `stlsort` nicht benötigt (`STLmap` ist effizienter).

```
sub stlsort {
   $aaref = $$a[2];        # $aa ist Referenz auf Array...
   $bbref = $$b[2];        # $aa ist Referenz auf Array...
# sortiere nach Jahr / TitelAut
   $aa = ansi2sort(join("\x01", $$aaref[1], $$aaref[0]));
   $bb = ansi2sort(join("\x01", $$bbref[1], $$bbref[0]));
   return $aa cmp $bb;
}
```

6.3 Hook-Verteiler

6.3.1 für die Aufbereitung

Die Struktur **LineFlow** in der Konfigurationsdatei ermöglicht das Einhängen von Spezialbehandlungen beim Einlesen von durch *avanti* gelieferten Ergebnissen.
Vordefiniert sind:

`1`	DifferentiateRegister
`3`	DifferentiateSTL
`4`	ChannelResult
`5`	StoreRecnums
`11`	DifferentiateRegister
`13`	StoreSTLnums
`14`	StoreSTL

Im einzelnen tun diese Routinen folgendes:

DifferentiateRegister

Die aktuelle Zeile wird untersucht, ob es sich um eine „reguläre“ Registeranzeige (’`qrix`’) oder um einen Kurlisteneintrag (bei ’`qrix title+`’) handelt.
Dementprechend werden Trefferzählung, Satznummer(n) und Text identifiziert, der Text wird dann durch **ParseSTLentry** (6.1.4) bzw. **ParseRegisterLine**(6.1.3)
Der Cache der verfügbaren Kurzlistenzeilen wird aktualisiert, ebenso die Liste **Thisnums** der aktuellen Satznummern.

DifferentiateSTL

Die Liste **Thisnums** der aktuell geführten Satznummern wird aktualisiert, der Text wird durch **ParseSTLentry** (6.1.4) aufbereitet.

ChannelResult

Mit ’\’ abschließende Zeilen werden mit der/den Folgezeile(n) zu überlangen Zeilen zusammengefaßt. Damit kann man etwa auch ohne den Allegro-Parameter `zl` auf ’0’ zu setzen in kritischen Situationen (Links!) beliebig lange Zeilen emulieren.

StoreRecnums

Die Zeile wird als Liste von Satznummern interpretiert und in der Variablen **Recnums** hinterlegt, **Recnumsnum** wird aktualisiert.

StoreSTLnums

Die aktuelle Satznummer wird als „benötigt“ im Kurztitelcache vermerkt.

StoreSTL

Die aktuelle Kurztitelzeile wird im Kurztitelcache vermerkt, an dieser Stelle aber nicht in die Ausgabe eingefügt!

6.3.2 für CollectQuery

Die Struktur **MaskenSpecial** in der Konfigurationsdatei ermöglicht das Einhängen von Spezialbehandlungen bei der Formulierung von Suchbegriffen.
Jede dieser Routinen darf dann folgende Variablen manipulieren:

$register symbolisches Register

$logik Logik

$trunk Trunkierung

$_ Suchbegriff

Wird ein komplizierter, gar geklammerter Suchbegriff konstruiert, empfiehlt es sich, in der Behandlung die Variable `$register` zu löschen, dann erfolgt keine weitere Aufbereitung durch **CollectQuery** mehr, insbesondere kein Umschließen des Suchbegriffs durch Anführungszeichen.
Beispiel:

```
%MaskenSpecial = (  # definiert Unterprogramme zur Vorbehandlung
                    # der suchbegriffe
    'PER' => 'Persify,Trunkify,SR',
    'KOR' => 'Korify',
    'TIT' => 'Orify',
    'ISB' => 'Isbify',
);
```

Dabei sind dann etwa

```
sub Isbify {
  s/\s/-/g;    # amerikanische Form der Eingabe...
  substr($_, 11) = '' if length ($_) > 11;
               # kappen, da Prüfziffer nicht indexiert
  $trunk = "?";   # immer trunkiert suchen!
  return;
}
```

oder (damit auch Nach Personen als „Vorname Nachname“ gesucht werden kann:

```
sub Persify {  # Person etwa als VN NN eingegeben?
```

mehr als ein Wort muß sein, damit wir arbeiten:

```
  return unless /\s/;
```

Komma oder Ordnungshilfe: Profi am Werk, nichts tun!

```
  return if /[\,\<\>]/;
```

In Worte zerlegen, Letztes Wort von Komma gefolgt voranstellen:

```
  my @parts = split(/\s/, $_);
  my $bla = pop(@parts) . ", " . join(" ", @parts);
```

Originalbegriff sicherheitshalber auch noch suchen, alles natürlich trunkiert.

```
  $_ = "(PER \"$_\?\") OR ( PER \"$bla\?\")";
```

Zusammengebauter Begriff ist zu komplex, als daß wir **CollectQuery** daran auch noch herumfummeln lassen wollten, so wird dies ausgeschlossen:

```
  $register = $trunk = '';
}
```

oder (ziemlich wüst):

```
sub Korify {
  if ( ! (/^\"/ | /\?/) ) {
```

aus KOR=x y mache (KOR=x y z) OR (TIT=x§) OR (TIT=y§)

```
      tr/"//d;     # alle Anfuehrungen loeschen
```

Es ist '\xA7' das Paragraphenzeichen ANSI-codiert. Es werden nur Worte mit mindestens vier Buchstaben gesucht, um nicht evtl. mit 'AND' nach Stopworten zu suchen!

```
      my $hilf = join("\xA7\" AND TIT \"",
                      grep (/.{4,}/, split(/\s+/, "$_")));
      $hilf =~ tr/<>//d;
      $_ = "(KOR \"$_\?\") OR ( TIT \"" . $hilf . "\xA7\")";
      $register = $trunk = '';
      return;
    };
  tr/"//d;     # erst einmal alle Anfuehrungen loeschen
  $_ = "\"$_\"";
}
```

Kapitel 7

Tricks

Im folgenden einige einfache Lösungen für Standardsituationen:

7.1 Externe Anbindungen

7.1.1 Übergang zu einem Bestellformular anbieten

Die Parameterdatei produziere, wobei nur darauf zu achten ist, daß *allegro* hier keinen Zeilenumbruch einschiebt:

```
#t{'!OR!'}
#usi p{'sign='} P{'#'}
#005 p{'zstit='} P{'#'}
#t{'!'}
```

Im Konfigurationsscript ist dann der *Hook* wie folgt definiert:

```
sub ResultHook {
   s/\!OR\!([^!]*)\!/&orderform($1)/e;
};
```

und es gibt eine zusätzliche Routine *orderform*, die ein HTML-Formular generiert, dessen „Bestellen"-Button dann einen Aufruf von `bestell.pl` (ein typisches Formmail-Skript etwa) auslöst, wobei „Sign" und „zstit" als Parameter übergeben werden:

```
sub orderform {
   local($_) = shift;
   my ($Sign, $zstit) = ("", "");
   /Sign=([^#]+)\#/i && ($Sign = $1);
   /zstit=([^#]+)\#/i && ($zstit = $1);
   return <<"XxX";
<form action="bestell.pl" method="GET">
<input type="hidden" name="Sign" value="$Sign">
<input type="hidden" name="zstit" value="$zstit">
<input type="submit" value="Bestellen" name="Bestellen"></form>
XxX
}
```

7.2 Navigationsfunktionalität

7.2.1 Frameset von aussen aufbauen

Die Idee ist, einen speziellen *JobTyp* einzusetzen, dessen *JobSubtyp* die eigentliche Aktion enthält. Im folgenden `tunnel.job`, der eine Variable *Tunnel* setzt und ansonsten für die Ausgabe fortfährt gemäß dem *JobTyp* „texts/index“, der auch der Standard-*JobTyp* für die Anwendung ist.

```
//PO&IniVar(Tunnel,t_tunnel)&
//PO&Set(JobTyp,texts/index)&
```

Das Template `texts/index.htm` enthält nun die Initialisierung:

```
<!- PO&IniVar(Tunnel)& ->
```

sowie Alternativen bei den Frames, hier also eine Hilfsseite im „statischen“ Fall und alle Übergabeparameter außer natürlich `t_tunnel` im „Tunnel“-Fall:

```
PO?IF(Tunnel)?
<FRAME SRC="PO!Pop!?t_PO!Tunnel!=x&PO!Everything(t_tunnel)!"
PO?ELSE?
<FRAME SRC="PO!Pop!?t_help_hsintro=x"
PO?ENDIF?
...
```

7.3 Beeinflussung der Ausgabe

7.3.1 Parametrierte Kurzliste: Ersatz

Im Job für *populo* steht nun anstatt des üblichen

```
write "PO:Set(LineTyp,3):" newline
list recnum
```

ein Download mit einer geeigneten Parameterdatei

```
xport param l-short
write "PO:Set(LineTyp,44):" newline
download set
```

Die Parameterdatei `l-short.apr` muß dabei folgenden Rahmenbedingungen genügen:

- pro Datensatz exakt eine Zeile
- Das Format ist *Satznummer*, `:`, *Freitext* (Der *Freitext* ist dabei kompatibel zur Programmierung in *ParseSTLEntry*)
- Zeichensatz ist derselbe, wie ihn *avanti* liefern würde, also die Windows-Variante „allegro.ttf“ des Ostwest-Zeichensatzes

7.3.2 Parametrierte Kurzliste II: völlig frei und ohne interne Satznummern

Manchmal kommt man mit den Informationen aus der .STL-Datei nicht ans Ziel, etwa weil man in der Kurzliste bereits Thumbnails einbinden möchte. In diesem Fall benötigt man eine Parameterdatei hier etwa `d-htmkur.apr`, die die gewünschte Information liefert, möglichst

durch TAB (Zeichen 9) gegliedert. Zeilenumbrüche sind legal, sie müssen jedoch durch ein vorangestellten Backslach (\) als artifiziell gekennzeichnet sein.
Im Job für *populo* steht nun anstatt des üblichen

```
write "PO:Set(LineTyp,3):" newline
list recnum
```

kann man dann z.B. folgendes nehmen:

```
xport param d-htmkur
write "PO:Set(LineTyp,44):" newline
download set
```

In der Konfigurationsdatei ist zusätzlich gesetzt:

```
%LineFlow = (
  '44' => \&ParseMyShort,    # Parametrierte Kurztitelzeilen
  );
```

und das hierfür eingeführte Unterprogramm `ParseMyShort` ist nicht optimal (die Schwierigkeit sind aber vor allem die Zeilenumbrüche aus der Parameterdatei):

```
sub ParseMyShort {  # LineTyp 44: Kurztitelzeilen
  return (undef, {}, []) unless $_;
  # Wg. Designfehler in (einziger verfuegbarer Hook ist dieser)
  # muss die gesammelte Zeile stets hinterher manipuliert werden
my $collectedstring;
$collectedstring = $RawResult[$#RawResult][2][0] if $concat;

my $typ = $concat ? 'c' : $LineTyp;
$concat = s/\\\\$//;
$concat = $concat ? 1 : 0;

my @fields = split(/\x09/, $collectedstring.$_);
my $temph;
$$temph{'IDN'} = $fields[0];
if ( $collectedstring ) {
    $RawResult[$#RawResult]->[1] = $temph;
    $RawResult[$#RawResult]->[2] = [$collectedstring, @fields];
  }
return ($typ, $temph, [$_, @fields]);
}
```

7.3.3 RTF-Dateien weiterleiten

Angenommen, eine geeignet gewählte Parmeterdatei produziert RTF (Rich Text), das natürlich mit einem geeigneten Programm (etwa WordView), nach Vorgaben des Benutzers anzuzeigen ist.
Der Job hierfür enthält dann:

```
//PO&Set(JobTyp, lokdump)&
//PO&Set(OutTemplExt, ".rtf")&
//PO&Set(ContentType, "text/rtf")&
```

und das Template `lokdump.rtf` sieht etwa wie folgt aus, d.h. es enthält den RTF-Kopf, eine Überschrift und die abschliessende geschweifte Klammer:

```
{\rtf1\ansi\deff0\deftab720{\fonttbl{\f0\fnil MS Sans Serif;}
{\f1\fnil\fcharset2 Symbol;}{\f2\fswiss\fprq2 System;}
{\f3\fnil Times New Roman;}}{\colortbl\red0\green0\blue0;}
\deflang1031\pard\qc\plain\f3\fs28 .(Ueberschrift).
\par \pard\plain\f3\fs20
\par \plain\f3\fs24\b Ergebnis
\par \plain\f3\fs20
\par Recherche erfolgte f\'fcr Jahr PO!Jahr!, Inst. PO!Sigel!
<!-- PO?LOOP? -->
PO!L_0!
<!-- PO?ENDLOOP? -->
\par \pard\plain\f3\fs20
\par -----
\par }
```

7.3.4 Download-Dialog auslösen

Hierfür muß nicht nur ein geeigneter Header *Content-Type* gesetzt werden, sondern möglichst auch noch ein zusätzlicher Header *Content-Disposition*:

```
PO&Set(ContentType, "application/x-allegro-alg")&
PO&Append(ContentType, '\nContent-Disposition="inline; filename=xy.alg"')&
```

Das Ausgabetemplate sollte in diesem Fall besonders minimal sein:

```
<!- PO?LOOP? ->
PO!L_0!
<!- PO?ENDLOOP? ->
```

Kapitel 8

Was fehlt

An dieser Stelle sollen noch durchzuführende Änderungen an *populo* sowie bekannte Bugs aufgelistet werden.

Parsing Die Trennung der Argumente bei Routinen/Funktionen funktioniert nicht richtig im Fall von Schachtelungen:

`PO&`*Rout*`(`*arg1*`,` *fkt2*`(`*arg_a*`,` *arg_b*`),` *arg3*`)&`

übergibt *fkt1* 4 Argumente, von denen das 2. und 3. zusammengehören! Zuweisungen wie

```
PO&Set(Ich, 'Nachname, Vorname')&
```

funktionieren auch nicht.

Abhilfe: `$Komma=','` in der Konfigurationsdatei setzen und mit `Append` arbeiten, wenn es sein muß.

Quotierungen `PO&Set( var, ein Text)&`

`PO&Set( var, 'ein Text')&`

`PO&Set( var, "ein Text")&`

sollten alle das gleiche machen

Security alle Variablen sollten in ein eigenes Package kommen, um Namenskollisionen zu vermeiden.

SanifyRegister Dieser Befehl initialisiert verschiedene Variablen, allerdings muß er dazu in einer *JobTyp*-Datei aufgerufen werden. Andererseits benutzten gewisse Routinen innerhalb von *populo* diese Variablen. Es könnte also die richtige Initialisierung fehlen!

mod_perl Voraussetzung ist stärkere Kapselung in Module

Unicode die interne Verarbeitung sollte mittelfristig auf Unicode umgestellt werden.

Erweiterungen Prinzipiell ist *populo* als Front-End für *avanti* nicht nur auf den Mechanismus CGI-Eingabe / HTML-Ausgabe beschränkt.

kombinierte Suche *avanti* ist etwas heikel bei kombinierten Suchanfragen mit Klammerungen und Quotierungen. Üblicherweise liefern kombinierte Suchen derzeit drollige Resultate, wenn einer der Teilbegriffe Null Treffer liefert.

Kapitel 9

Revision history

1.17 xx.05.2002, Maintenance-Release Bugfixes: Massive Überarbeitung des Kommunikationsmoduls **Populo::Avanti**.

1.16 19.05.2003, Maintenance-Release

Obacht: Dies ist die letzte Version von *populo*, die noch mit Perl 5.3 funktioniert, zukünftige Versionen werden Perl 5.6 und neuer verlangen.

Obacht: Dies ist die letzte Version von *populo*, die URLs in der „historischen“ Form des Aufrufs `http://foo.bar/populo.pl&db=baz` (`populo.pl` bindet das Konfigurationsskript ein) ermöglicht, in Zukunft wird nur noch die Form *http://foo.bar/baz* bzw. *http://foo.bar/baz.pl* unterstützt (d.h. das Konfigurationsskript bindet `populo.pl` ein).

Bugfixes: '`$`' und '`@`' in '`"`'-Zeichenketten wurden interpretiert, nicht angezeigt.
Die Wildcards aus **TrunkChars** (default; '`$`', '`?`' und '`*`') wurden von der Zeichencodierungsroutine *tr_i1_al* geschluckt, also überall. Nun nur noch von **CollectQuery**.

Features: **InitHook** zur Nachinitialiserung nach Einlesen der Aufrufparameter eingeführt.
2. Argument von **GetRecnums** und **SplitChecknums** nun optional.

1.15 21.10.2002, Maintenance-Release

Bugfixes/Features: **VonBisRes** behandelte '`-1990`' anders ('`<`') als '`0-1990`' ('`<=`')
Konflikt, falls Variable wie Funktion hiess: Es wurde die Variable genommen, auch bei Aufruf **PO!Name(...)!**
SanifyRegister wie angekündigt deaktiviert

1.14 09.07.2002, Maintenance-Release

Bugfixes: Diagnostische ASCII-0 Markierung im *avanti*-Output nur noch bei **Debug = 1**
HTML-Korrektur am diagnostischem Output
Content-Type-Header jetzt mit explizitem '`charset=ISO-8859-1`'

Features: Mehr Umsetzungen von Ostwest nach HTML jenseits ISO 8859-1

1.13 28.05.2002, Maintenance-Release

Features/Bugfixes: Der optionale Übergabewert „SR“ für **&CollectQuery** muss nicht mehr unbedingt '`&`' sein, falls nichtleer
Besseres Handling von Funktionsaufrufen **PO!Fun()!** (ohne Argument und Prototyp **[e**

also).
Hyperlinks unter Perl 5.003 hiessen alle '`lnkescp`'

1.12 28.04.2002, Maintenance-Release

Bugfixes: Übergebene Umlaute wurden mit '`!`' verschoent
Registerübergreifende Verweise funktionierten nicht mehr
win2sor verbessert (Sonderzeichen sortieren nicht mehr)
Doppeltes Säubern von Übergabewerten

Avdemo: Endlich die Möglichkeit zum Rückwärtsblättern eingestellt
Register: Kopfformulare zu einem zusammengefasst
„direktes Download“ als Auswahloption bei Vollanzeige (Verzweigung in **regsrch.job**, Ausgabe durch **download.htm**

1.11 22.04.2002, Maintenance-Release

Bugfixes: Avanti-Treffer wurden manchmal als Oktalzahlen aufgefasst
Kompatibilitaet mit alten Populo-Konfigurationen verbessert
&Collect wandelte Zeichen nicht um

Features: Handling von Übergaben in UTF-8

1.10 04.04.2002, *avanti* Version 1.7 zwingend.

Obacht: Variable müssen nun mit `IniVar` oder `Declare` angemeldet werden, falls sie innerhalb einer Kontrollstruktur benutzt werden.

Features: Die Konfigurationsvariable **@NoStat** enthaelt eine Liste HTTP-Felder, die bei *Stat* nicht mitprotokolliert werden sollen.
Direkte Ausgaben (Kurztitel vor allem) von *avanti* werden nun standardmäßig als im Zeichensatz allegro-Windows (*Allegro.ttf*, d.i. Umcodierung durch `o.cpt` des OSTWEST-Zeichensatzes) interpretiert.
Im Konfigurationsskript kann eine Hash **%CustomTR** definiert werden, der Routinen für eigene Zeichenumsetzungen spezifiziert
ContentType darf nun **n** enthalten, das bewirkt einen Zeilenumbruch, der für mehrzeilige Header nötig ist (etwa „Content-Disposition: ...“)
Schachtelungen von Ausdrücken nun möglich
Die Vergleichsoperatoren '&&' und '||' sowie '`=` ' und '`!` ' sind nun ebenfalls erlaubt.
Beliebte Konfigurationsroutinen **Future** und **QueryParse** aus Anwenderskripten übernommen
Socket-Kommunikation komplett umgeschrieben, Timeouts eingeführt.
Neue Konfigurationsvariable **$ReuseSocket**: Default ist 1, d.h. die Verbindung mit *avanti* kann für mehrere Jobs benutzt werden.
Autoloading von nicht immer benötigten Routinen mag zu einer gewissen Beschleunigung führen.

Bugfixes: Falls wegen Avanti-Problemen Kurztitelzeilen nicht geliefert wurden, werden diese mittels **&OpenStl** explizit aus der STL-Datei geholt. **&OpenStl** ist aber in moderneren Konfigurationsskripten normalerweise nicht mehr enthalten, dies wurde abgefangen.
Falls in Templates die letzte Zeile nicht korrekt mit einem Zeilenumbruch endete, wurde das letzte Zeichen abgeschnitten.
'`!`' ist nun in **PO!...!**-Konstruktionen erlaubt.
Das Ausfiltern von Anführungszeichen aus Suchbegriffen durch **PrepareQuery** erwies

sich als nicht günstig.

Reaktion auf Avanti-Bugs: Beim Protokollieren des QueryString stürzte Avanti bei zu langen Texten ab. Abhilfe durch das Array *@NoStat*, das etwa eine Liste der Felder enthalten sollte, die Satznummern enthalten können.
Klammern in Suchbegriffen müssen großzügig mit Blanks umgeben werden, sonst scheitern manche Suchen (Korrektur in **CollectQuery**)
avanti-Versionen von Herbst 2001 und später haben Probleme mit der Nutzung eines Sockets für mehrere Jobs und Ergebnisse (genauer: die Systemdienst-Version hatte diese Probleme schon immer, die neuen Versionen schließen den Socket während der nächste Job übertragen wird). Abhilfe durch `$ReuseSocket = 0;` in der Konfigurationsdatei.
Bei Schiller-Räuber-Expansionen liefert Avanti in der Ergebnismenge manche Satznummern mehrfach, dies wird in **StoreRecnums** (**LineTyp** = `5`) abgefangen.

Avdemo: HTML standardkonformer gemacht.
Links bei „vielen“ Treffern im Register (`t_acindex`) rufen `t_allegro` nun mit Modus „exakt“ auf.
Aufruf `populo.pl?db=avdemo` löst nun ein Redirect auf `avdemo.pl` aus, Übergabefeld ’`db`’ konsequent eliminiert.

1.09 gab es nicht.

1.08 gab es nicht.

1.07 07.06.1999

Features: Konfigurationsvariable *Stat* bewirkt Protokollierung des QueryString.
Paginate liefert jetzt auch *StartNo* und *EndNo*, damit Trefferlisten einfacher fortlaufend gezählt werden können.
Bindestriche (’`-`’) in Jobtypen jetzt erlaubt.
seltener Fehler in Tests auf Verbindung behoben

Avdemo: *LinkEscape* produzierte Link `<a href=...>` ohne Anführungszeichen
Label `toomuch` bei zuvielen Treffern fehlte
in **acexpand** war der Suchbegriff fuer qrix nicht in `""` (Pech bei Gesamttiteln mit ’ ; ’...)
Hilfsseiten mit `IF(0 == 1)` jetzt statisch und dynamisch sichtbar
Hilfsseiten jetzt im Unterverzeichnis **help** (war: **avdemo**)

1.06 12.12.1998, *avanti* Version 1.5 zwingend.

Interne Verarbeitung beschleunigt fuer grosse Resultate.
Dokumentation der Hooks.

Features: Struktur *AvantiErrTrans* bzw. Routine *AvantiErrHook*
Routine *Message*
Kontrollstruktur *WHILE*

Changes: Bug in *PrepareQuery* bei leeren Mehrfachfeldern
PrintVariablesShort verschoent.
Apostroph wird bei Eingabe nicht mehr getoetet.

Vergleichsoperatoren <= und >= wurden nicht korrekt erkannt.
Paginate erkennt als *Aktion* jetzt auch +, - und numerische Werte (setzen Seitennummer).

Avdemo: *PopDebug->init()* auf *WantDebug* umgestellt.
Hilfsseiten jetzt unteralb von **./cat/**, benötigen kein virtuelles Verzeichnis **/avdemo** mehr.

1.05 9.10.1998

Changes: *SplitFind* wieder abgeschafft (nicht mehr noetig).
STLtosort unterscheidet nicht mehr zwischen Gross-Kleinschr.
Everything war nicht escaped.

Feature; *STLmap* löst *stlsort* ab (mit Schwartzian Transform)

Avdemo: benutzt *STLmap*
Kurztitel verschoent, Sortierung korrigiert.

1.04 9.8.1998 *avanti* Version 1.4 zwingend.

Changes: *RegPraefix* eliminiert.
Jetzt auch Aufruf über Konfigurationsscript möglich.
Festverdrahteter JobTyp *notjet* umbenannt in *notyet* (peinlich).

Avdemo: Anpassung an neue Struktur der Kurztitel

1.03 20.7.1998

Fixes: Security

Neues Feature: *SplitFind*: Lange Befehle werden automatisch zerlegt.

1.02 gab es nicht.

1.01 27.4.1998

Fixes: Debug-Feature **ShowResult** verbessert.
LineTyp 13 wurde faelschlich sortiert.
Quotierung bei Mehrwort-find war erforderlich.
Comment-Fix.

Neue Features: *LineTyp* 14, *Fetchit* für separates Zwischenspeichern von Kurztitellisten.
DbTime eingeführt (Datum der Konfigurationsdatei), *DateForm*.

GetSTLentry in *ParseSTLentry* umbenannt.

Avdemo: Quotierung repariert
neuer Jobtyp **hickexp** (Balkon in `acexpand` für erweitertes Register bei Abfragen mit Remote-*avanti*)

Dokumentation: HTML-Version, Beginn eines Index

Distribution: `poppop.lzh` aufgeteilt in `populo.lzh` und `nodb.lzh`.

1.00 16.4.1998: erste Freigabe mit neuer Sprache

Index