Inhaltsverzeichnis

1. Einleitung

Diese Dokumentation gibt einen ersten Einblick in den Aufbau von Kleio Webservern. Sie soll für angehende Webprojekte als eine Art Hilfestellung dienen. Grundlegend werden hierin alle wichtigen Kleio Programmelemente erklärt.

Zu Beginn werden mit Hilfe von Übungsdateien einfache Abfragen an dem Datenbanksystem Kleio geübt.

Im zweiten Teil der Dokumentation wird am Beispiel einer konkreten Bilddatenbank näher auf weiterführende Konzepte und auf bilddatenbankspezifische Probleme eingegangen.

Die Übungsdateien gehen auf die wissenschaftliche Übung: „Hybride und verteilte Datenbanken: Kleio“ von Prof. Manfred Thaller im Sommersemester 2004 zurück. Es wurden jedoch eigens für diese Dokumentation zwei neue XML Datenbanken („maler.xml“ und „abstraktion.xml“) angefertigt und dementsprechend alle Query Dateien umgearbeitet. Ein Vorteil der neuen XML Datenbanken ist, dass ihr Dateninhalt in deutscher Sprache abgefasst wurde.

1. Historie

Das Datenbanksystem Kleio wurde Ende der 70er Jahre am Max-Planck-Institut für Geschichte in Göttingen erarbeitet. Es fußt auf dem Konzept der quellennahen Datenverarbeitung und ist speziell für die Auswertung von historischen Quellen in der historischen Forschung entwickelt worden.

Bei Kleio handelt es sich im Besonderen um ein nicht-relationales Datenbankmanagementsystem, für das ein kontext-sensitives Datenmodell entwickelt wurde, welches sich an den Vorstellungen semantischer Netzwerke orientiert. Um den Besonderheiten des historischen Quellenmaterials zu begegnen, bietet das Programm verschiedene datentechnische Hilfsmittel an.

Die Daten in der historischen Forschung kennzeichnen sich durch:

- extrem irreguläre Strukturen (d.h. Felder mit stark wechselnden Längen),
- einen hohen Anteil von fehlenden Feldern,
- die häufige Wiederholung von einzelnen Gruppen von Feldern,
- unterschiedlich starke Tiefen von Hierarchien,
- üblicherweise große Datenbasen,
- komplexe Semantiken der Datenbasen (Gebrauch des nicht dezimalen Währungssystems, oder von Feldern, die Familien- bzw. Ordensnamen enthalten),
- in historischen Dokumenten finden sich ebenso eine Reihe von Informationen, die mit den aus anderen Programmiersprachen bekannten Datentypen nicht erfassbar bzw. nicht interpretierbar sind.

Grundsätzlich kann man also zwei Aspekte der Beschreibung von historischen Daten hervorheben: Einerseits, dass die Daten in den historischen Wissenschaften komplexer sind und andererseits, dass sie generell in einer viel größeren Datenmenge vorkommen. Diesen besonderen Anforderungen an ein Datenbankprogramm kann kommerzielle Software nicht folgen.[1]

2. Wie arbeitet man mit Kleio?

Das Datenbankmanagementsystem Kleio liegt derzeit in der Version 10.0 vor.[2] Zur Installation von Kleio 10 ist ein normal ausgestatteter PC mit Windows, Linux oder Macintosh/OS X erforderlich. Als Mindestanforderung an den Rechner gelten, dass der Rechner ein Pentium 1 oder 2 sein sollte, mit 48 MB RAM. Derzeit wird darüber hinaus keine weitere Software zur Bearbeitung benötigt. Es gibt unter Kleio 10 zur Abbildung der Datenbasen eine grafische Oberfläche, von deren Einsatz allerdings abgeraten wird, da

[1] Vgl. Grotum: Das digitale Archiv, S. 28-30.
[2] Gegenwärtig wird die Kleioversion 10.1 in den neueren Projekten der HKI verwendet. Weitere Projekte an der HKI werden unter diesem Link beschrieben: http://www.hki.uni-koeln.de/projekte/projekte-b.html.

sie einen X Emulator voraussetzt und gegenwärtig umgearbeitet wird.[3]

Als grundsätzliche Hauptbetriebsart wird Kleio vom Benutzer somit anhand einer spezifischen Query Language im Batch-Betrieb genutzt. Alle Datenbankkommandosprachen bestehen aus Gruppen von Befehlen, wobei bei Kleio festzuhalten ist, dass die DDL (Definitionssprache) und die DML (Manipulationssprache) des DBMS weitgehend zusammenfallen.

Diese konsolenorientierte Arbeitsweise hat einerseits den Vorteil, dass der Benutzer den vollen Leistungsumfang des Datenbanksystems nutzen kann. Der Nachteil ist, dass er sich noch näher in die spezielle Abfragesprache einarbeiten muss.

2.1. Das Quellenmaterial

Ausgangspunkt aller Abfragen bei den Übungen im ersten Teil der Dokumentation sind die beiden XML Datenbanken „maler.xml“ und „abstraktion.xml“.

Informationen zur Datenbank „maler.xml“:

- 12 definierte Namen von Elementen,
- 2 definierte Namen von Gruppen,
- 65 enthaltene Dokumente,
- 104 enthaltene Gruppen,
- 622 enthaltene Elemente.

Informationen zur Datenbank „abstraktion.xml“:

- 9 definierte Namen von Elementen,
- 2 definierte Namen von Gruppen,
- 65 enthaltene Dokumente,
- 94 enthaltene Gruppen,

[3] Vgl. zur älteren Menüsteuerung: Gross, Gabriele: Kleio. Eine Einführung in die Menüsteuerung.

- 457 enthaltene Elemente.

Das Verhältnis zwischen den Elementen im Quellenmaterial kann mit Hilfe eines Baumdiagramms grafisch veranschaulicht werden.

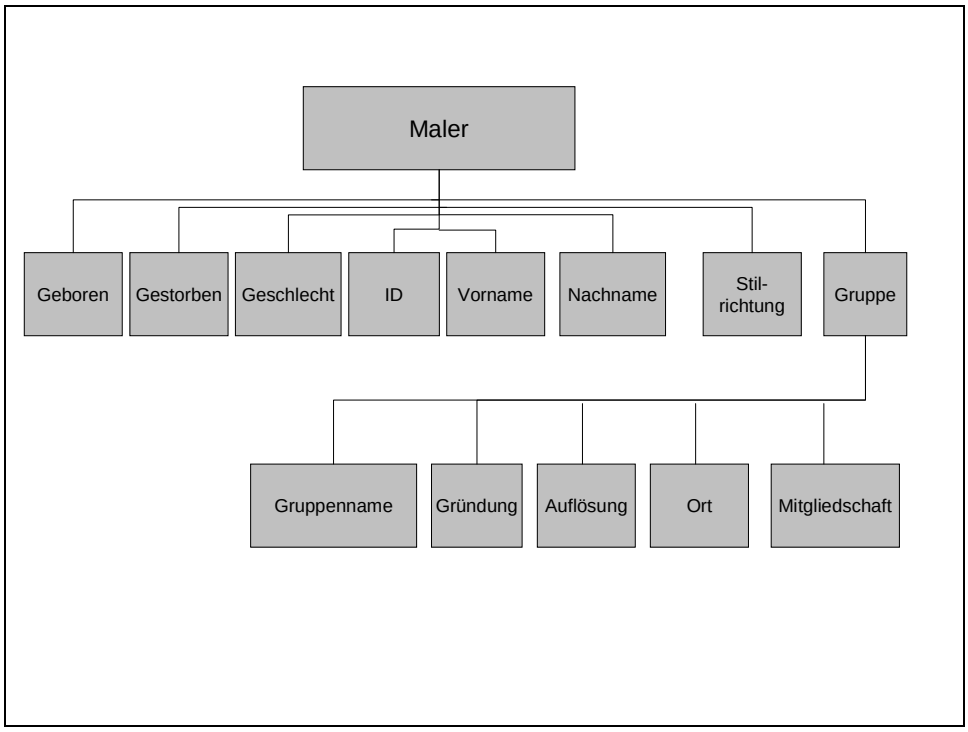


Abbildung 1: Baumdiagramm zu maler.xml.

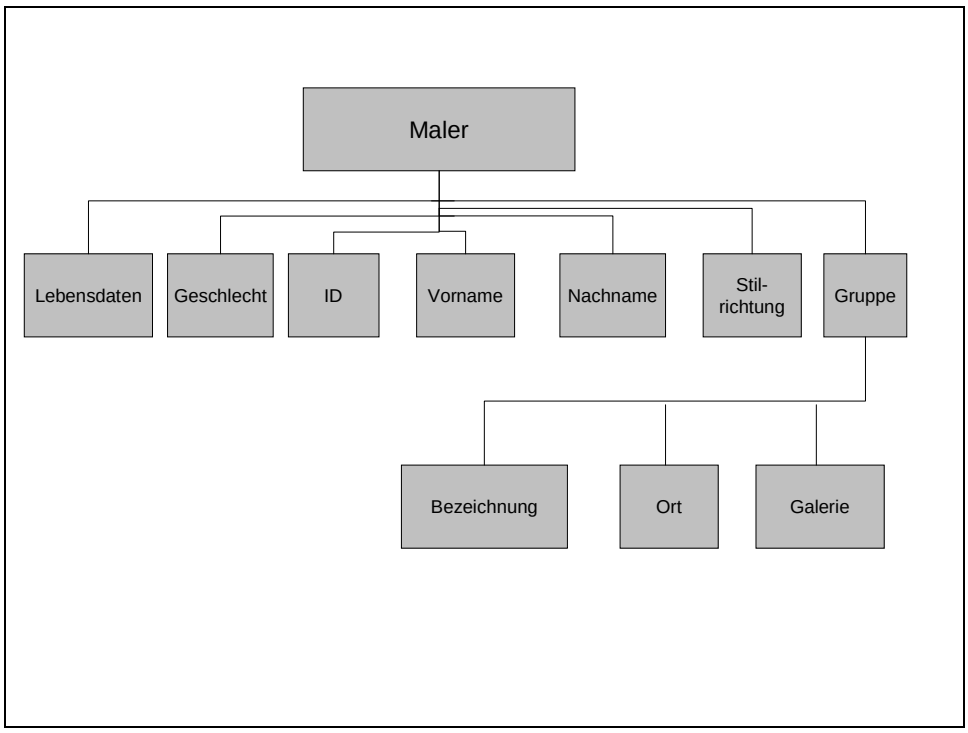


Abbildung 2: Baumdiagramm zu abstraktion.xml.

2.2. Gliederung von Informationen in den Datenbanken

Alle Kleio Datenbanken gliedern sich zunächst in:

a) Dokumente:

Die oberste Ordnungseinheit sind die Dokumente. So werden alle Informationen der Datenbank in Dokumenten abgelegt. Diese Dokumente bestehen hierbei aus hierarchisch verschachtelten Gruppen, zwischen denen beliebig viele zusätzliche, nichthierarchische Beziehungen bestehen können.

Beispiel: Es gibt in der Datenbank „maler.xml“ ein Dokument mit dem Namen „maler“.

b) Gruppen:

Die Dokumente sind erneut unterteilt in Dinge, die Gruppen heißen. Gruppen begreift man zunächst als Dinge der realen Welt. Einheiten, die Informationen gruppieren, nennt man Gruppen. So wären sie bei einer relationalen Datenbank beispielsweise so etwas wie Records. Gruppen können ebenso weitere Gruppen enthalten, die Anzahl der Ebenen ist dabei egal. Beziehungen zwischen Gruppen in unterschiedlichen Datenbanken nennt man auch Bridges. Gruppen enthalten - abgesehen von anderen Gruppen - Elemente, die

beliebig lang sein können.

Beispiel: In der Datenbank „maler.xml“ sind die beiden Gruppenelemente „maler“ und „gruppe“ vereinbart.

c) Elemente:

Zunächst geht man davon aus, dass Dinge Elemente sind, wenn sie Daten enthalten. Elemente haben einen bevorzugten Datentyp, der bestimmt, wie mit dem Inhalt gearbeitet wird. Elemente, die in Gruppen vorkommen, werden namentlich aufgeführt. Ferner können Elemente verschiedenen Datentypen entsprechen und zu Gruppen zusammengefasst werden.

Beispiel: In der Datenbank „maler.xml“ wären als Elemente mit definiertem Namen anzuführen: „geboren“, „gestorben“, „geschlecht“, „id“, „vorname“, „nachname“, „stilrichtung“, „gruppenname“, „gruendung“, „aufloesung“, „ort“ und „mitgliedschaft“.

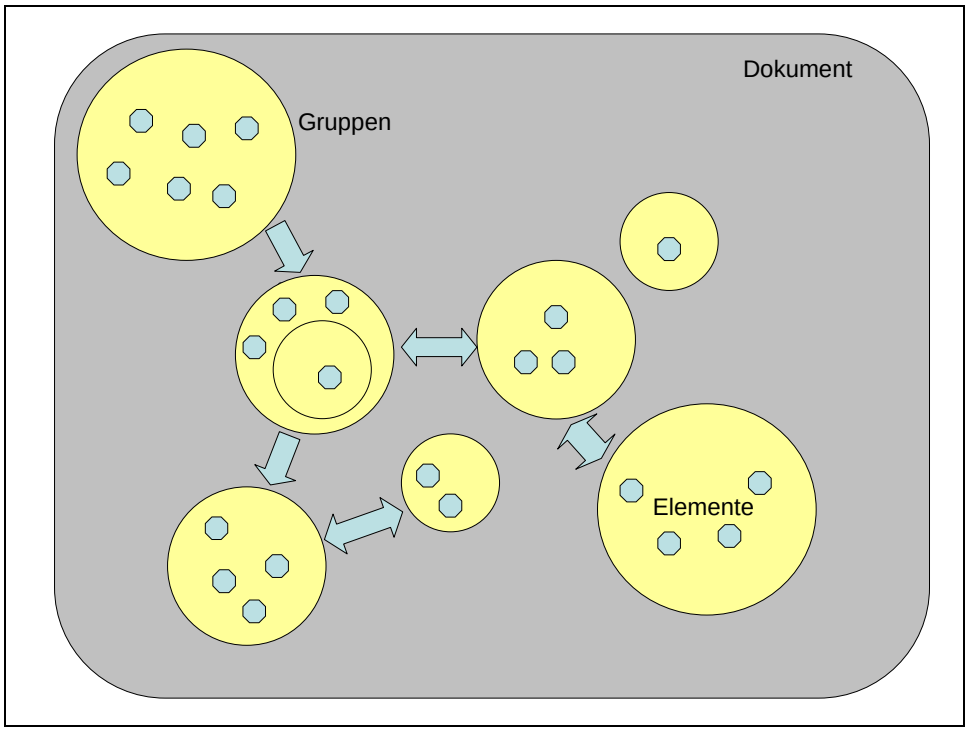


Abbildung 3: Modell der Basisstruktur.

2.3. XML und Kleio

Die eigentlichen Eingabedaten, bzw. das zu verwaltende Quellenmaterial für eine Kleio Datenbank wird heute vornehmlich im XML-Format bereitgestellt. XML stellt dabei die häufigste Möglichkeit dar, Daten einzulesen. Bei älteren Datenbank-Projekten (und früheren Versionen von Kleio) gab und gibt es noch die Möglichkeit, Eingabedaten aus einer einfachen ASCII-Datei heraus einzulesen.

Bei den Eingabedaten im XML-Format ist hervorzuheben, dass Kleio ein DBMS ist, das Daten auf der Basis von Graphen verwaltet. Diese Graphen sind eine geeignete Struktur, zur Bearbeitung von XML-Hierarchien. Wie wir bereits wissen, besitzt Kleio ein Datenmodell demzufolge Datenbanken als Netze mit bestimmten semantischen Eigenschaften untersucht werden können. Diese in Kleio verwendeten Netze realisieren äußerst allgemeine Graphen, in denen relationale Entwürfe, wie auch XML-basierte Datendesigns als Spezialfälle realisiert werden können. Damit hat Kleio sämtliche Eigenschaften eines „native XML DBMS“.[4]

2.3. Grundlegende Syntax der Befehlssprache

In diesem und im nächsten Abschnitt wird nur eine allgemeine Einführung zum Grundverständnis der Syntax gegeben, in den späteren Übungsdateien wird die Befehlssprache mit jeder neuen Aufgabe an Komplexität hinzugewinnen. Die Kommandosprache bei Kleio kommt grundsätzlich folgendem Muster nach:

Kommando Parametername=Parameterwert

Beispiel:

datebase name=fauzk

Ein Kommandowort leitet einen Befehl oder eine Direktive ein. Durch Parameter werden Befehle oder Direktiven eingehender beschrieben. Zwischen Kommandowort und

[4] Vgl. Thaller: Hybride und verteilte Datenbanken: Kleio. Hier: Netzwerkorientierte / „native XML“ Datenbanken.

Parametername steht ein Leerzeichen. Der Parametername sagt genau, was spezifiziert werden soll. Der Parameterwert charakterisiert, wie die Beschreibung zu erfolgen hat. Kommandos beginnen immer in der ersten Zeile; bei Fortsetzungszeilen bleibt die erste Spalte leer. Ein Parameter darf also nie in der ersten Spalte einer Zeile stehen, weil nur Kommandowörter in diese Zeile geschrieben werden dürfen.

2.4. Zugriff auf Gruppen, Elemente und Funktionen über Pfade

Bei einer Kleio Query kann man folgendermaßen auf die Informationen einer Datenbank zugreifen:

1. Zugriff auf Gruppen:

/ gruppenname

2. Zugriff auf Elemente:

: elementname

3. Zugriff auf Gruppenfunktionen:

/ schlüsselwort [optionale parameterliste]

4. Zugriff auf Elementarfunktionen:

: schlüsselwort [optionale parameterliste]

Selbstverständlich sind noch weitere verschiedene Kombinationen von Pfadzugriffen möglich, die in den nachfolgenden Abschnitten eingehender erläutert werden.

2.5. Definition der Datenstruktur

Nachdem wir über ein adäquates Quellenmaterial in Form der Datenbanken „maler.xml“ und „abstraktion.xml“ verfügen, müssen wir nun dafür sorgen, dass Kleio die Datenstruktur dieser XML Datenbanken auch interpretieren kann.

2.5.1. *.dat Datei:

Als eine Art Einleseaufforderung für das System fungiert zunächst die *.dat Datei. In der *.dat Datei wird der Pfad zu den Rohdaten beschrieben. Ein Beispiel für eine *.dat Datei:

```
read name=fauzk;other=markup;origin="../database/descriptions";total=yes;
     sensitive=no
```

Die Daten werden, wie bereits erwähnt, aus einer XML-Datei gelesen. Es gibt für diese Daten allerdings keine DTD im Sinne von XML, weil die für Kleio zentralen Punkte mit den Sprachmitteln einer DTD nicht ausgedrückt werden könnten. Ein Ersatz für eine DTD findet sich in der *.mod Datei.

Bei der Beschreibung der *.dat Datei wird mit read name=fauzk die Datenbank mit dem Namen fauzk zum lesen bestimmt. other=markup besagt, dass es sich dabei um Markup handelt. In dem Fall werden die Dateien als XML-Dateien betrachtet, es kann jedoch gleichfalls Markup gelesen werden, das zwischen XML und SGML angesiedelt ist. Die Anweisung total=yes gibt einen Hinweis auf das Directory, welches mit origin= den Pfad beschreibt, indem sich die zu verarbeiteten Daten befinden. Die Anweisung sensitive=no bedeutet, dass die Groß- und Kleinschreibung ignoriert wird.

Weitere Anweisungen nach sensitive=no wären optional self=yes und replace=single. self=yes ordnet an, dass falls unidentifizierte Charakter Identifier vorkommen, so werden diese dann ohne Fehlermeldung akzeptiert.

Es wird zunächst grundlegend davon ausgegangen, dass Dokumente eindeutige Identifikationsnummern besitzen müssen. Bei der Anweisung replace=single gibt es mehrere Dokumente mit einer einzigen Identifikationsnummer, angenommen man hätte für drei Dokumente drei Mal die gleiche Identifikationsnummer (D1=xy1, D2=xy1, D3=xy1).

Per Standard ist replace=modify definiert.

2.5.2. *.mod Datei:

Die Strukturvereinbarungen befinden sich in einer so genannten *.mod Datei (das Kürzel mod ist mnemonisch und steht hier für Datenmodell). Im Modell der *.mod Datei wird beschrieben, wie die einzelnen Daten einer Datenbank zu interpretieren sind. Die Rohdaten können dadurch also besser eingelesen werden.

Mit Hilfe der *.mod Datei werden die Informationsgruppen in ihren Abhängigkeiten definiert, die Elemente den jeweiligen Gruppen zugeordnet, die Elemente in ihren Eigenschaften (z.B. Datentypen) definiert und die logischen Objekte, die einzelne Elemente beschreiben, werden ebenfalls definiert.

In unserem Fall sieht die fauzk.mod Datei folgendermaßen aus:

```
database name=fauzk;first=vdIb;overwrite=yes;identification=no

transparent name=storage,fileroot,bibref;type=ignore
transparent name=ebindheader,filedesc,titlestmt,publicationstmt

part name=vdIb;identification=no;
     part=bibid,binding,frontmatter,body,backmatter;
     also=titleproper,author,pubplace,date

part name=bibid;
     also=value,name

part name=binding,frontmatter,body,backmatter;
     part=contentitem

part name=contentitem;
     part=itemstmt,div;

part name=itemstmt;
     part=bibid;
     also=titleproper,author,pubplace,date

part name=div;
     part=itemstmt,page,div;
```

```
     also=level,head

part name=page;
     part=image,text;
     also=property;

part name=image;
     also=extref,seqno,nativeno

part name=text;
     also=type;
     position=text

exit name=fauzk
```

Beschreibung der fauzk.mod:

```
database name=fauzk;first=vdIb;overwrite=yes;identification=no
```

Zu Beginn der *.mod Datei wird mit datebase name=fauzk die Datenbank definiert. Mit first=vdib wird festgelegt, dass die Gruppe vdib den Beginn eines Dokuments anzeigt. Es kann mehrere Gruppen geben, die die Eigenschaft haben den Beginn eines Dokuments anzuzeigen, aber es muss mindestens eine angegeben werden. vdib ist die oberste Hierarchieebene. overwrite=yes bewirkt, dass der bestehende Inhalt stets durch einen neu erstellten (bzw. durch neue Abfragen) überschrieben werden darf. Um identification=no zu erklären, muss man sich erstmal näher mit Unterschieden zwischen Relationalen und Objektorientierten Datenbanken beschäftigen. Ein Unterschied zwischen Relationalen und Objektorientierten Datenbanken ist, dass bei Objektorientierten jedes Objekt eine während der Lebenszeit der Datenbank unveränderliche „Identität“ besitzt. Kleio verhält sich genauso. Für den Benutzer tritt die Identität vor allem dadurch zu Tage, dass jedes Dokument, das in die Datenbank eingelesen wird, eine eindeutige Kennnummer erhält. Um sicherzustellen, dass diese Identität gleich bleibt, auch wenn die Datenbank mehrmals hintereinander aus den XML-Daten neu kompiliert wird, wird standardmäßig angenommen, dass das erste Element der ersten Gruppe eines jeden Dokuments eine solche Identität enthält. Durch identification=no beim Database Befehl wird angeordnet, dass die Dokumente einfach durchgezählt werden, anders ausgedrückt, dass die Daten keine Identifikationsnummern enthalten. In unserem Fall macht das Sinn, weil die Bindeeinheiten auf die sich die Inhaltsverzeichnisse der vdib Dokumente beziehen, keine langfristigen Entitäten, sondern eher bibliothekarische Zufälle sind. Grundsätzlich *sollte*

man bei Dokumenten allerdings in den Daten Identifikationsnummern bereitstellen: Da dies so ist, steht das identifikation=no nochmals bei der Parse Direktive zu vdib, um endgültig klarzumachen, dass wir hier vom empfohlenen Vorgehen abweichen.

```
transparent name=storage,fileroot,bibref;type=ignore
transparent name=ebindheader,filedesc,titlestmt,publicationstmt
```

transparent name= besagt, dass bestimmte Teile des Markups ausgeblendet werden sollen. Bestimmte Tags, Inhalte bestimmter Tags und/oder Attribute werden ignoriert, indem diese als "tranparente Symbole" definiert werden. Beim Kompilieren werden diese Tags und deren Inhalt übergangen. Das Ziel ist dabei, einen selektiven Zugriff auf die Daten der Datei zu erlangen.

2.5.2.1. Definition von Gruppen

In diesem und in den nächsten Abschnitten werden die besonderen Aufgaben der *.mod Datei näher erläutert.

```
part name=vdIb;identification=no;
     part=bibid,binding,frontmatter,body,backmatter;
     also=titleproper,author,pubplace,date
```

Die Direktive part zeigt jeweils an, welche Gruppen es in der Datenbank gibt. Nach dem name= Parameter werden diese namentlich angeführt. In der ersten Zeile wird vdIb als eine Gruppe definiert. In der nächsten Zeile können die Abhängigkeiten innerhalb der Gruppen, bzw. ihre hierarchische Struktur, vereinbart werden. Durch den part Parameter der Part Direktive wird dies aufgezeigt. Dieser part Parameter erwartet als Parameterwert eine Liste, von den von vdIb abhängigen Gruppen. Folglich sind „bibid“, „binding“, „frontmatter“ etc … vdIb untergeordnet. Nach dem Parameter also folgen Elemente, die nur Tags enthalten. Der Parameter also zeigt an, welche Elemente eine Gruppe enthalten kann, dabei gilt XML-Elemente entsprechen Kleio Elementen.

2.5.2.2. Definition von Elementen

Die Definition von Elementen erläutern wir am Beispiel der „maler.mod“ Datei:

```
item name=geboren;
      usage=date
type name=numbers;
      only=yes
exit name=geboren

item name=gestorben;
      usage=date
type name=numbers;
      only=yes
exit name=gestorben

database name=maler;first=maler;overwrite=yes
date first="19.01.1839"

element name=geboren;type=date;date=geboren
element name=gestorben;type=date;date=gestorben
element name=geschlecht;type=category;category=geschlecht
part name=maler;
      part=gruppe;
      position=id,geschlecht,vorname,nachname,stilrichtung,geboren,gestorben
part name=gruppe;
      position=gruppenname,gruendung,aufloesung,ort,mitgliedschaft
exit name=maler
```

Damit Kleio weiß, welche Elemente in der Datenbank „maler.xml“ vorkommen, müssen wir dies durch den position Parameter der part Direktive vorab mitteilen.

```
part name=gruppe;
      position=gruppenname,gruendung,aufloesung,ort,mitgliedschaft
```

Hier enthält beispielsweise die Gruppe „gruppe“ die Elemente „gruppenname“, „gruendung“, „aufloesung“, „ort“ und „mitgliedschaft“.

2.5.2.3. Definition der Eigenschaften von Elementen

Nachdem dem Programm mitgeteilt wurde, welche Informationen es in den jeweiligen Datenbanken gibt, wird nun erläutert, welche Eigenschaften diese Informationen besitzen bzw. welchem Datentyp sie angehören.

```
element name=gestorben;type=date;date=gestorben
element name=geschlecht;type=category;category=geschlecht
```

Die Definition der Eigenschaften von Elementen wird durch die element Direktive eingeleitet. Beim Element wird ein bevorzugter Datentyp mit Hilfe des type Parameters definiert. Als Datentypen stehen text (Standard), number, date und category zur Auswahl.

2.5.2.4. Definition der logischen Objekte/Datei startup.env

Wenn spezielle Daten mit den normalen Datentypen nicht mehr erfassbar oder interpretierbar sind, so werden sie in Kleio anhand ihrer Merkmale definiert und als logische Objekte verwaltet. Logische Objekte sind Teil der logischen Kleio Umwelt und werden in dieser verwaltet. Jedes logische Objekt gibt Kleio Instruktionen, wie es einen speziellen Teil der Daten verstehen soll.

Folgende Typen zwischen logischen Objekten werden vornehmlich unterschieden:

1. Standard logische Objekte, die Bezug nehmen zu den definierten Datentypen innerhalb einer Datenbank.

2. Modifizierte logische Objekte, die Regeln von den standardisierten logischen Objekten übernehmen und einige benutzerdefinierte Regeln besitzen.

3. Benutzerdefinierte logische Objekte, die komplett die standardisierten logischen Objekte überschreiben.

Logische Objekte können beispielsweise Kataloge, Packages, katalogisierte Prozeduren, Bildvereinbarungen und Ähnliches sein (besondere logische Objekte werden ausführlicher in späteren Kapiteln erklärt). Derzeit existieren etwa 45 logische Objekte, die in Klassen

zusammengefasst werden.[5]

Logische Objekte werden normalerweise innerhalb einer *.mod Datei definiert.

```
item name=text;usage=text;overwrite=yes
signs text=yes
exit name=text
```

Bei diesem Beispiel eines logischen Objekts wird als Datentyp text definiert. Durch die Anweisung signs text=yes wird veranlasst, dass Sonderzeichen wie alle anderen Zeichen verwendet werden sollen.

Falls man jedoch mit zwei Datenbanken arbeitet, die die gleichen Definitionen der Datenstruktur verwenden, eignet es sich eine zusätzliche Datei anzufertigen, auf die beide Datenbanken bezüglich der logischen Objekte zugreifen können.

Die logischen Objekte kann man in diesem Fall am Beispiel der Datei „startup.env“ für die Datenbanken „maler.xml“ und „abstraktion.xml“ beschreiben. Diese Datei enthält nähere Vereinbarungen zu logischen Objekten, Abkürzungen und Vereinbarungen der numerischen Angaben:

```
item name=geschlecht;usage=category
sign sign=m;write="männlich"
sign sign=w;write="weiblich"
exit name=geschlecht

item name=abbrev;usage=category
sign sign=m;write="männlich"
sign sign=w;write="weiblich"
sign sign=ü;write="&uuml;"
exit name=abbrev

item name=jahreinteilung;usage=number
text name="m";number=0.0833333
text name="mo";number=0.0833333
text name="monate";number=0.083333
exit name=jahreinteilung
```

[5] Vgl. Thaller, Manfred: Texts, databases, Kleio: a note on the architecture of computer systems for the humanities, S. 49-76; ferner Thaller, Manfred: Kleio 4. Ein Datenbanksystem; Woollard/ Denley: Source-Oriented Data Processing for Historians, S. 116-120; ebenso die Beschreibung der quellenorientierten Datenverarbeitung in: Grotum, Thomas: Das digitale Archiv, S. 22-65.

Logische Objekte werden durch den item Befehl eingeführt und durch exit beendet. Durch den Parameter usage wird ihr Datentyp vereinbart.

```
item name=geschlecht;usage=category
sign sign=m;write="männlich"
sign sign=w;write="weiblich"
exit name=geschlecht
```

Das Geschlecht wird durch den Datentyp category festgelegt. category wird immer dann verwendet, wenn bestimmten Abkürzungen eindeutige Sachverhalte zugewiesen werden sollen. Einem Kürzel, eingeleitet mit der sign Direktive und dem sign Parameter, wird hierbei eine Konstante in Hochkommata zugewiesen, die durch den write Parameter der sign Direktive ausgeschrieben wird. Bei den Datenbanken „maler.xml“ und „abstraktion.xml“ steht also beim Element-Tag <geschlecht> jeweils als Inhalt nur ein „m“ oder „w“. Bei konkreten Abfragen werden die Einzelbuchstaben nun ersetzt durch die Konstanten „männlich“ oder „weiblich“.

3. Kleio Webserver

Um Datenbanken im Internet zu präsentieren, kann Kleio selbst als CGI-Server eingesetzt werden. Grundlegend ist zu beachten, dass allen Aktivitäten im Internet die Client/Server-Architektur zugrunde liegt. So werden Informationen zunächst auf Computern gespeichert, die als Web-Server eingerichtet wurden. Die Dateien auf den Web-Servern sind öffentlich zugänglich und meist in HTML kodiert. In diese Web-Seiten kann man beliebige Multimedia-Inhalte einfügen und per Hyperlinks miteinander verbinden. Organisiert werden die Dateien anhand einer URL (Uniform Resource Locator), einer Art Adresse, die den physikalischen Objektidentifikatoren entsprechen, anders ausgedrückt, die den physikalischen Speicherort des Dokuments kodiert. Jede URL besitzt einen speziellen Pfadnamen, der aus einer Zeichenkette besteht. In dieser Zeichenkette werden Domain-Namen, Verzeichnis- und Dateinamen angeführt. Jede URL beginnt mit dem Präfixkürzel des HyperText-Transport-Protokolls (HTTP), dass für den Zugriff auf Web-Seiten entwickelt wurde.

Das Format für eine absolute URL im HTTP-Protokoll wäre demnach:

1. http
2. Rechnername oder eine IP-Adresse des Rechners
3. Port (bzw. die Portnummer). Der Port dient dem Rechner zur Unterscheidung verschiedener Dienste. Der voreingestellte Dienst ist 80 für HTTP und 443 für HTTPS.
4. ein Pfad zu einer Ressource auf dem Rechner.

   Ein Beispiel: `http://rechner[:port][pfad][?anfrage].`

Ein Nachteil einer URL-Adressierung ist, dass die URLs nicht dauerhaft gegen die Verschiebung von Dokumenten gefeit sind. Wenn beispielsweise ein Dokument in ein anderes Unterverzeichnis verschoben wird, oder der Name geändert wird, so ist die ursprüngliche URL ungültig. Teilweise wird dieses Problem durch die Einführung der URNs (Uniform Resource Names) gelöst. Diese entsprechen logischen Objektidentifikatoren, die unabhängig vom physischen Speicherort der Quelle sind. Speziell in digitalen Informationssystemen für geisteswissenschaftliche Zwecke eignen sich besonders die „Permanent Universal Resource Identifiers“, die unveränderlichen eindeutigen Referenzen. In diesem Zusammenhang wäre dediziert für die Geisteswissenschaften ebenso das DACO-Konzept für flexible Beschreibungsmechanismen für Objekte in Kulturerbe-Servern anzuführen.[6]

Der Zugriff auf Datenbanken im WWW wird immer häufiger über dynamische Web-Seiten realisiert. Der Web-Server benutzt dafür eine Standardschnittstelle, die als Common Gateway Interface (CGI) bezeichnet wird. CGI ist somit eine Middleware, eine zusätzliche Softwareschicht, die zwischen der Benutzeroberfläche und dem DBMS den Zugriff auf Datenbanken vereinfacht. Die CGI-Schnittstelle führt externe Programme oder Scripts aus, um auf die dynamischen Informationen zuzugreifen. An den Server zurück werden die Informationen über HTML und letztlich vom Server an den Browser zurückgegeben.

Dieses CGI-Script kann in jeder beliebigen Programmiersprache realisiert werden. Häufig genutzte Sprachen sind hierbei PERL, Tcl, Unix Shell oder C/C++. In unserem Fall wird das CGI-Skript durch Kleio erzeugt. D.h., wir verwenden ein in Kleio implementiertes

[6] Vgl. Thaller: Reproduktion, Erschließung, Edition, Interpretation: Ihre Beziehungen in einer digitalen Welt; ferner: Tanenbaum/van Steen: Verteilte Systeme, S. 746-748; ebenso Hochstätter: Die Rolle digitaler „Cultural Heritage Systeme“ als Bestandteil des kulturellen Gedächtnisses, weiter Loebbecke/Thaller: Preserving Europe’s Cultural Heritage in the Digital World.

„Kommunikationsprotokoll“, das sich das CGI zu Nutze macht. Kleio kann also als eine CGI-Engine zur Realisierung dynamischer Webserver verstanden werden.[7]

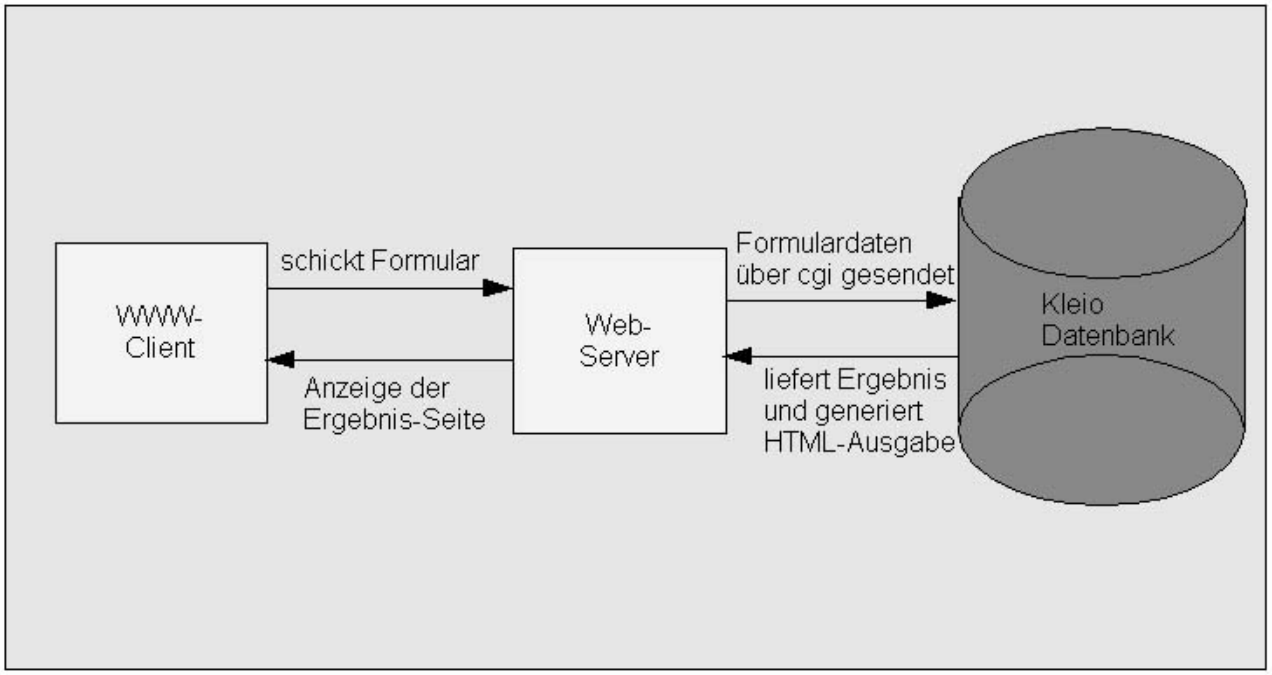


Abbildung 4: Prinzips eines CGI-Servers mit Kleio (Common Gateway Interface)

Beschreibung zur Grafik: der Nutzer sendet ein Formular mit dem Web-Browser. Der Browser codiert die Daten und sendet sie wie jede andere Dokumentenanfrage zum WWW-Server. Der Server wiederum übermittelt die Eingabedaten als Parameter an ein CGI-Programm/-Script und startet dieses Script. Das Script verarbeitet die Eingabedaten und stellt Datenbank-Abfragen an die Kleio-Datenbank, wertet den Abfrage-Report aus und gibt das Ergebnis wieder an den Server zurück. Der Server sendet die dynamisch generierten Ergebnisse als HTML-Dokument zum Nutzer bzw. Client zurück, wo die Ergebnisse der Anfrage im Web-Browser dargestellt werden.

3.1. Befehle zur Kommunikation in HTTP-Protokollen

Das HTTP-Protokoll ist ein Client/Server-Anwendungsprotokoll. Es basiert auf dem Internetprotokoll TCP/IP (Transmission Control Protocol/Internet Protokol) und wurde entwickelt, um Dokumente in verschiedene Richtungen übertragen zu können.

[7] Vgl. Elmasri/Navathe: Grundlagen von Datenbanksystemen, S. 937-950; ferner Kemper/Eickler: Datenbanksysteme, S. 515-519; weiterhin: Heuer/Saake: Datenbanken: Konzepte und Sprachen, S. 592-594; ebenso: Vorländer: CGI – kurz&gut.

Wichtige Befehle zur Kommunikation in HTTP-Protokollen sind:[8]

HEAD	zum Abfragen der Informationen über eine Web-Seite (Anforderung, den Header eines Dokuments zurückzugeben)
GET	zum Anfordern einer Web-Seite (ein Dokument soll an den Client zurückgegeben werden)
PUT	zum Speichern einer Seite vom Web-Browser an den WWW-Server (Anforderung, ein Dokument zu speichern)
POST	zum Senden von Daten (Formularen) vom Web-Browser an den WWW-Server (Bereitstellung von Daten, die dem Dokument hinzugefügt werden sollen)
DELETE	zum Löschen einer vorher gespeicherten Seite auf dem WWW-Server (Anforderung, ein Dokument zu lösen)

3.2. URL-Encoding

Bei einer GET-Anforderung werden die Befehle, die vom Benutzer erfragt werden, zusammengefasst an den Web-Server übergeben. Der Browser des Anwenders realisiert das Zusammenfassen. Diese Parameter werden dann getrennt durch ein „?“ an die URL angehängt. Wenn mehrere Parameter/Werte-Paare übergeben werden, müssen sie durch ein „&“-Zeichen voneinander getrennt werden. Erfragt werden diese Parameter in einem HTML-Formular. Jeder Eingabewert wird an den Variablennamen aus dem Formular gebunden. Diesen Vorgang bezeichnet man auch als das URL-Encoding. Kommen die Daten in dieser Form am Server an, so werden sie durch Parsen wieder entschlüsselt und dem CGI-Programm als Variablen zur Verfügung gestellt.

[8] Vgl. Abeck/Lockemann/Schiller et al.: Verteilte Informationssysteme, S. 200-202.

Beispiel für einen Daten-String, wie es an den WWW-Server gesandt wird:

```
"http://www.server.domain/test/test.html?name1=wert1&name2=wert2&name3=wert3"
```

Hinter jedem Variablennamen wird ein "=" gesetzt. Darauf folgt dann der Wert der Variablen. Hinter jedem Variablenwert wird ein "&" gesetzt und definiert somit das Ende des Dateninhalts der Variablen. Ausnahme: kein & hinter der letzten Variablen.
Beinhaltet der Wert einer Variablen ein Leerzeichen, so wird dieses durch ein "+"-Zeichen ersetzt.
Sonderzeichen:
Da das '+' Zeichen (und andere) eine spezielle Bedeutung haben, muss das kodieren eines echten '+' hexadezimal von statten gehen. '+' bedeutet %2b. Das %xy (Prozentzeichen gefolgt von zwei anderen Zeichen) spezifiziert den hexadezimalen Wert eines ASCII-Zeichens.

Beim Command-Line Parameter können auch Parameter an das Skript-Programm übergeben werden. Wird nämlich an die URL ein "?" gefolgt vom Skriptaufruf und den Parametern angehängt, so wird die URL wie eine normale Dokumentenanfrage an den WWW-Server gesandt.
Beispiel:

```
   <a href="test/cmdline.cgi?xyz">
```

Auf dem Server wird das Argument "xyz" an das Skript "cmdline.cgi" im "test" Unterverzeichnis übergeben. Es wurden also 2 Parameter übergeben:
arg 0 = "cmdline.cgi"
arg 1 = "xyz"
Beim einem String, der URL-encoded wurde, sind die Parameter durch '+' getrennt:

```
   <a href="test/cmdline.cgi?Eins+Zwei+Drei+Vier">
```

Es wurden 5 Argumente übergeben:
arg 0 = "cmdline.cgi"
arg 1 = "Eins"
arg 2 = "Zwei"
arg 3 = "Drei"
arg 4 = "Vier"
Auch in diesem Fall, hat das '+' Zeichen (und andere) eine spezielle Bedeutung. Deshalb

muss das dekodieren eines echten '+' hexadezimal vor sich gehen. '+' bedeutet %2b.
Folglich:

```
<a href="test/cmdline.cgi?dies%2bist%2bnur%2bein%2bargument">
```

Es wurden 2 Parameter übergeben:
arg 0 = "cmdline.cgi"
arg 1 = "dies+ist+nur+ein+argument"

Allgemeines Beispiel für die Verwendung einer GET-Anforderung in einem HTML-Formular:

```
<html>
<head>
      <title>Formular</title>
</head>
<body>
     <form action=“http://xyz/cgi“ method=“get“>
          <input type=text name=“name“ value=”Emil Muster”>
           <textarea name=”Zusatz”>Hier ein Texteingabefeld.
           </textarea>
      <input type=submit value=”Suche starten”></input>
</form>
</body>
</html>
```

Daraus folgt die Anfrage-URL:

http://xyz/cgi?name=Emil+Muster&Zusatz=Hier+ein+Texteingabefeld.

3.3. URL-Codierung bei Kleio

Die CGI-Komponenten von Kleio stellen zunächst fest, welche Aufforderungen aus dem CGI-Proktokoll verwendet wurden, und sorgen anschließend dafür, dass in den einzelnen Kleio “Prozeduren“, die übergebenen Argumente nach einer einheitlichen Logik ankommen. Dabei gilt bei allen GET und POST Aufrufen werden stets Namen/Werte-Paare übergeben. An die URL durch Schrägstriche angefügte Argumente werden dagegen gezählt. Um jedoch mit beiden Formen von Interfaces sowohl Namen/Werte-

Paare als auch gezählte Argumente verwenden zu können, gelten folgende Regeln:

1. <input type=“hidden“ name=“Feld“ value=“Wert“> führt dazu, dass dem Namen „Feld“ der Wert „Wert“ zugewiesen wird.
2. <input type=“hidden“ name=“_2“ value=“Wert“> führt dazu, dass das erste (! obwohl Unterstrich _2 dort steht) gezählte Argument den Wert „Wert“ hat.
3. ... /kleioc/0010/exec/prozedur/“Wert“ führt ebenfalls dazu, dass das erste gezählte Argument den „Wert“ hat.
4. ... /kleioc/0010/exec/prozedur/Feld/“Wert“ führt dazu, dass das Feld „Feld“ den Wert „Wert“ hat. Namen können in weiterer Folge als so genannte Ersetzungsmarker innerhalb von Kleio Programmen verwendet werden. In unserem Beispiel würde etwa @=feld@ durch die Zeichenkette „Wert“ ersetzt. @#1@ würde ebenfalls durch „Wert“ ersetzt, wenn ein erstes gezähltes Argument vorhanden ist. Sobald ein Argument benannt wird, darf man es nicht mehr zählen.

Die einzelnen Befehle in HTTP-Protokollen werden später noch eingehender in den jeweiligen Projektdateien erläutert.

3.4. Die Basisstruktur eines logischen Kleioservers

```
Software:

cgi-bin
cgi-bin/kleiob
cgi-bin/kleioc
cgi-bin/kl__.*
cgi-bin/*.ini

Datenbank(en)
database
database/fauzk.dat
database/fauzk.mod

Eigentliche Programme:
```

```
server
server/build.db
```

4. Übungsdateien

Im Folgenden werden anhand der zwei XML Datenbanken „maler.xml“ und „abstraktion.xml“ die grundlegenden Schritte der Kleio Kommandosprache näher am Beispiel von Übungsdateien erläutert. Grundsätzlich gehen diese Übungsdateien bzw. „Steps“ auf eine wissenschaftliche Übung der Historisch-Kulturwissenschaftlichen Informationsverarbeitung, die von Prof. Manfred Thaller im Sommersemester 2004 an der Universität zu Köln geleitet wurde, zurück. Ziel dieser Dateien ist die schrittweise Hinführung zu einem unter Kleio betriebenen Webserver.

Zu Beginn werden die Anfangsschritte sehr genau erläutert. Mit fortgeschrittener Erkenntnis werden auch die Erläuterungstexte kürzer gehalten.

4.0. Installation der Übungsdateien

Zunächst beschreiben wir die Installation unter Windows. Die aktuelle Kleio Version v10 können Sie sich vorab einmal unter dieser Adresse downloaden.: http://www.hki.uni-koeln.de/teach/ss04/U_Kleio/tag2/index.html

Legen Sie für die Dateien am Besten ein Verzeichnis unter C:\kleio an. Wenn die Dateien entpackt wurden, öffnen Sie die startup.bat mit einem beliebigen Texteditor. In dieser Datei werden die Pfadinformationen vereinbart. Natürlich können Sie die Path-Statements durch Modifikation noch Ihren Wünschen entsprechend anpassen.

Es empfiehlt sich die Übungsdateien nicht in demselben Verzeichnis wie die Programmdateien aufzubewahren.

Beispiel startup.bat:

```
@echo off
doskey
set PATH=%PATH%;c:\kleio\v10
```

```
set KLEIO=c:\kleio\v10
c:
cd \daten\tag2
```

Nachdem Sie die startup.bat Ihren Wünschen entsprechend verändert haben, rufen Sie Kleio am Besten mit Hilfe der DOS-Konsole auf. Wechseln Sie zunächst in das Verzeichnis, in dem das Programm installiert wurde (im Beispiel C:\kleio). Geben Sie startup.bat ein. Durch diesen Aufruf werden Sie automatisch in Ihr gewähltes Directory gewiesen, in dem sich Ihre Übungsdateien befinden. Nachdem Sie unter C:\daten\tag2\ sind, rufen Sie bitte nacheinander zum Kompilieren folgende Dateien auf: a) kleiob startup.env, b) kleiob maler.mod und c) kleiob maler.dat.

4.0.1. Aufruf der Aufgabe auf der Konsole

Kompiliert werden die Aufgaben durch a) den Aufruf des Befehles kleiob, b) den Namen der jeweiligen Aufgabe und c) einer selbst benannten Ergebnisdatei, der vorzugsweise eine *.txt Extention angehängt werden sollte.

Aufruf-Schema:

kleiob bsp.1 ergebnis.txt

4.1. Step 1: Gesamter Inhalt der Datenbank ausgeben

```
query name=maler;part=/maler
write
stop
```

Erläuterung:

- Abfrage in der Datenbank maler. Durch den part Parameter wird auf das Gruppenelement maler zugegriffen, wobei das Gruppenelement über ein Slash-Zeichen (/) angesprochen wird. Es wird der gesamte Inhalt der Künstlerdatenbank ausgegeben.
- Die Ausgabe wird durch den write Befehl eingeleitet.
- Die Ausgabe umfasst alle Angaben zu den Künstlern in der Datenbank. Einer Künstlergruppe können dabei beliebig viele Maler angehören.

Ausgabe ergebnis1.txt: Es wird hier und in den folgenden Dateien aus Platzgründen jeweils nur der erste Teil der Ergebnisdatei angezeigt.

```
Kleio Version 10.0.0 --- 31.03.2005, 05:00:40
Historical Workstation Project / UNIX version
query name=maler;part=/maler
write
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.


maler (1 = "m01")
          geboren      31.7.1883
          gestorben    27.1.1970
          geschlecht   männlich
          id           m01
          vorname      Erich
          nachname     Heckel
          stilrichtung Expressionismus

   gruppe (1 = "gru-1")
             gruppenname  Die Brücke
```

```
            gruendung    1905
            aufloesung   1913
            ort          Dresden und Berlin
            mitgliedschaft 1905-1913

maler (2 = "m02")
          geboren      6.5.1880
          gestorben    15.6.1938
          geschlecht   männlich
          id           m02
          vorname      Ernst Ludwig
          nachname     Kirchner
          stilrichtung Expressionismus

   gruppe (1 = "gru-1")
            gruppenname  Die Brücke
            gruendung    1905
            aufloesung   1913
            ort          Dresden und Berlin
            mitgliedschaft 1905-1913

maler (3 = "m03")
          geboren      31.12.1869
          gestorben    3.11.1954
          geschlecht   männlich
          id           m03
          vorname      Henri
          nachname     Matisse
          stilrichtung Fauvismus

   gruppe (1 = "gru-1")
            gruppenname  Fauves
            gruendung    1905
            aufloesung   1909
            ort          Paris
            mitgliedschaft 1905-1909

maler (4 = "m04")
          geboren      16.10.1874
          gestorben    24.9.1930
          geschlecht   männlich
          id           m04
          vorname      Otto
          nachname     Mueller
          stilrichtung Expressionismus
```

```
   gruppe (1 = "gru-1")
             gruppenname  Die Brücke
             gruendung    1905
             aufloesung   1913
             ort          Dresden und Berlin
             mitgliedschaft 1910-1913
. . .
Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.2. Step 2: Eine Gruppenfunktion ausgeben

```
query name=maler;part=/gruppe
write
stop
```

Erläuterung:

- Durch den part Parameter wird auf das Gruppenelement gruppe zugegriffen.
- Die Ausgabe wurde eingeschränkt und umfasst jetzt nur noch alle Angaben zu den Künstlergruppen in der Datenbank.

Ausgabe ergebnis2.txt:

```
Kleio Version 10.0.0 --- 13.04.2005, 17:36:41
Historical Workstation Project / UNIX version
query name=maler;part=/gruppe
write
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.


maler (1 = "m01") : gruppe (1 = "gru-1")
          gruppenname  Die Brücke
          gruendung    1905
```

```
          aufloesung   1913
          ort          Dresden und Berlin
          mitgliedschaft 1905-1913

maler (2 = "m02") : gruppe (1 = "gru-1")
          gruppenname  Die Brücke
          gruendung    1905
          aufloesung   1913
          ort          Dresden und Berlin
          mitgliedschaft 1905-1913

maler (3 = "m03") : gruppe (1 = "gru-1")
          gruppenname  Fauves
          gruendung    1905
          aufloesung   1909
          ort          Paris
          mitgliedschaft 1905-1909

maler (4 = "m04") : gruppe (1 = "gru-1")
          gruppenname  Die Brücke
          gruendung    1905
          aufloesung   1913
          ort          Dresden und Berlin
          mitgliedschaft 1910-1913
. . .
```

4.3. Step 3: Ausgabe in alphabetischer Reihenfolge

```
query name=maler;part=/maler
index part=:nachname;limit=", ";
      part=:vorname;
      type=list
stop
```

Erläuterung:

- Durch den part Parameter wird auf das Gruppenelement maler zugegriffen.
- Durch den index Befehl wird das Element nachname in alphabetischer Reihenfolge ausgeschrieben. Der Parameter limit begrenzt die Ausgabe mit einem Komma. Ebenfalls wird das Element vorname ausgeschrieben. Durch type=list werden die

Identifikationsnummern der Dokumente unterdrückt.

- Bei der Ausgabe werden alle Maler als alphabetisches Register angezeigt, jeweils mit dem Vor- und Nachnamen. Die Nachnamen sind in alphabetischer Reihenfolge.

Ausgabe ergebnis3.txt:

```
Kleio Version 10.0.0 --- 31.03.2005, 06:33:48
Historical Workstation Project / UNIX version
query name=maler;part=/maler
index part=:nachname;limit=", ";
      part=:vorname;
      type=list
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
am Ende, Hans

Barlach, Ernst

Beckmann, Max

Bock, Marie

Braque, Georges

Brusselmans, Jean

Campendonk, Heinrich

Cezanne, Paul

de Kooning, Willem

de Vlaminck, Maurice

Derain, André

Dix, Otto

Dufy, Raoul
```

```
Ensor, James

Feininger, Lyonel

Fenneker, Josef

Friesz, Emile-Othon

. . .
```

4.4. Step 4: Ausgabe in „HTML“-Datei leiten

```
query name=maler;part=/maler
index part=:nachname;limit=", ";
      part=:vorname;
      type=list
stop target="register.html";overwrite=yes
```

Erläuterung:

- Durch den part Partmeter wird auf das Gruppenelement maler zugegriffen.
- Der index Befehl wird weiter durch einen part Parameter spezifiziert. Nach part wird auf das Element nachname zugegriffen (nach Doppelpunkt Ausgabe von Elementen). Das Element soll durch ein Komma begrenzt werden. Ebenfalls soll das Element vorname ausgegeben werden. type=list unterdrückt die Identifikationsnummern.
- Durch das stop target Kommando wird die Ausgabe in eine HTML-Datei umgelenkt. overwrite=yes bewirkt, dass der bestehende Inhalt stets durch einen neu erstellten Inhalt (bzw. durch neue Abfragen) überschrieben werden darf.
- Bei der Ausgabe in die register.html werden alle Maler angezeigt, jeweils mit dem Vor- und Nachnamen. Die Nachnamen sind in alphabetischer Reihenfolge geordnet.

Ausgabe register.html:

```
am Ende, Hans

Barlach, Ernst

Beckmann, Max

Bock, Marie

Braque, Georges

Brusselmans, Jean

Campendonk, Heinrich

Cezanne, Paul

de Kooning, Willem

de Vlaminck, Maurice

Derain, André

Dix, Otto

Dufy, Raoul

Ensor, James

Feininger, Lyonel

Fenneker, Josef

Friesz, Emile-Othon
. . .
```

4.5. Step 5: HTML-Datei definieren durch Package

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei
```

```
query name=maler;part=/maler
index part=:nachname;limit=", ";
      part=:vorname;
      type=list
stop target="register.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Das Package htmldatei wird definiert. Es enthält die Grundformatierungsangaben einer einfachen HTML-Seite.
- Danach beginnt die eigentliche Datenbankabfrage, die sich nicht von Step 4 unterscheidet. Durch den part Partmeter wird auf das Gruppenelement maler zugegriffen.
- Der index Befehl wird weiter durch einen part Parameter spezifiziert. Nach part wird auf das Element nachname zugegriffen. Das Element soll durch ein Komma begrenzt werden. Ebenfalls soll das Element vorname ausgegeben werden. type=list unterdrückt die Identifikationsnummern.
- Durch das stop Kommando wird die Ausgabe in eine HTML-Datei umgelenkt.
- Das Package htmldatei wird aufgerufen und durch form=html wird vereinbart, dass der Inhalt als Character Entities ausgegeben wird.
- Bei der Ausgabe werden der Vor- und Nachname der Maler angeführt.

Quellcode der register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>
am Ende, Hans

Barlach, Ernst

Beckmann, Max
```

```
Bock, Marie

Braque, Georges

Brusselmans, Jean
. . .
Westhoff, Clara

Wouters, Rik

</body>
</html>
```

Ausgabe der register.html im Webbrowser:

```
am Ende, Hans Barlach, Ernst Beckmann, Max Bock, Marie Braque, Georges Brusselmans, Jean Campendonk, Heinrich Cezanne, Paul de Kooning, Willem de Vlaminck, Maurice Derain, AndrÃ© Dix, Otto Dufy, Raoul Ensor, James Feininger, Lyonel Fenneker, Josef Friesz, Emile-Othon Gauguin, Paul Gerstl, Richard Gromaire, Marcel Grosz, George Guttuso, Renato Hansen, Svend Wiig Heckel, Erich Iskowitz, Gershon Kandinsky, Wassily Kirchner, Ernst Ludwig Klee, Paul Kokoschka, Oskar Kollwitz, KÃ¤the Kubin, Alfred Macke, August Mackensen, Fritz Manguin, Henri Charles Marc, Franz Marquet, Albert Matisse, Henri MÃ¼nter, Gabriele Meidner, Ludwig Modersohn, Otto Modersohn-Becker, Paula Mueller, Otto Munch, Edvard Nauen, Heinrich Nolde, Emil Overbeck, Fritz Paerels, Willem Pascin, Jules Pechstein, Max Picasso, Pablo Puy, Jean Rohlfs, Christian Rouault, Georges Schiele, Egon Schirren, Ferdinand Schmidt-Rottluff, Karl ThÃ©venet, Louis Thorn-Prikker, Jan van Dongen, Kees van Gogh, Vincent Vogeler, Heinrich von Jawlensky, Alexej von Werefkin, Marianne Westhoff, Clara Wouters, Rik
```

4.6. Step 6: Ausgabe als Liste gestalten

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="<li>";
exit name=registeranfang
```

```
item name=registerende;usage=package;overwrite=yes
fields limit="</li>"
exit name=registerende

query name=maler;part=/maler
index part=:nachname;limit=", ";package=registeranfang;
      part=:vorname;package=registerende;
      type=list
stop target="register.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Zunächst werden drei Packages definiert. Das Package htmldatei enthält wie gehabt die Grundformatierungsangaben einer einfachen HTML-Seite. Hinzu gekommen sind die -Tags zur Gestaltung einer ungeordneten Liste.
- Package registeranfang enthält öffnendes li-Tag.
- Package registerende enthält schließendes li-Tag.
- Danach beginnt die Datenbankabfrage mit Zugriff auf Gruppenelement.
- Durch den index Befehl werden nacheinander die Elemente nachname und vorname ausgegeben. Nach part wird auf das Element nachname zugegriffen. Danach wird das Package registeranfang aufgerufen. Ebenfalls soll das Element vorname ausgegeben werden, mit Aufruf des Packages registerende.
- Durch das stop Kommando wird die Ausgabe in eine HTML-Datei umgelenkt.
- Zum Schluss wird das Package htmldatei aufgerufen und definiert.
- Bei der Ausgabe werden der Vor- und Nachname der Maler angezeigt.

Quellcode der register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body><ul>
<li>am Ende, Hans</li>
```

```
<li>Barlach, Ernst</li>

<li>Beckmann, Max</li>

<li>Bock, Marie</li>

<li>Braque, Georges</li>

<li>Brusselmans, Jean</li>
. . .
<li>Westhoff, Clara</li>

<li>Wouters, Rik</li>

</ul></body>
</html>
```

Ausgabe der register.html im Webbrowser:

- am Ende, Hans
- Barlach, Ernst
- Beckmann, Max
- Bock, Marie
- Braque, Georges
- Brusselmans, Jean
- Campendonk, Heinrich
- Cezanne, Paul . . .

4.7. Step 7: Ausgabe als Liste mit Hilfe der Einbaufunktion package [ ]

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="<li>";
exit name=registeranfang
```

```
item name=registerende;usage=package;overwrite=yes
fields limit="</li>"
exit name=registerende

query name=maler;part=/maler
index part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
      type=list
stop target="register.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Wie Step 6 bis zur Datenbankabfrage.
- Per index Befehl wird nun auf die Einbaufunktion :package[ ] zugegriffen. Innerhalb dieser Einbaufunktion wird das Element nachname und das Package registeranfang vereinbart. In der zweiten Einbaufunktion wird das Element vorname und das Package registerende vereinbart.
- Zum Schluss wie Step 6.
- Bei der Ausgabe werden der Vor- und Nachname der Maler angezeigt.

Quellcode der register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body><ul>
<li>am Ende, Hans</li>

<li>Barlach, Ernst</li>

<li>Beckmann, Max</li>

<li>Bock, Marie</li>

<li>Braque, Georges</li>

<li>Brusselmans, Jean</li>
```

```
. . .
<li>Westhoff, Clara</li>

<li>Wouters, Rik</li>

</ul></body>
</html>
```

Ausgabe der register.html im Webbrowser:

- am Ende, Hans
- Barlach, Ernst
- Beckmann, Max
- Bock, Marie
- Braque, Georges
- Brusselmans, Jean
- Campendonk, Heinrich
- Cezanne, Paul . . .

4.8. Step 8: Unbeschränkte Ausgabe

```
query name=maler;part=/maler
write
stop
```

Erläuterung:

- Abfrage in der Datenbank maler. Durch den part Parameter wird das Gruppenelement maler zugegriffen. Es wird er gesamte Inhalt der Künstlerdatenbank ausgegeben.
- Die Ausgabe wird durch den write Befehl eingeleitet.
- Die Ausgabe umfasst alle Angaben zu den Künstlern in der Datenbank.

Ausgabe ergebnis8.txt:

```
Kleio Version 10.0.0 --- 31.03.2005, 08:03:47
Historical Workstation Project / UNIX version
query name=maler;part=/maler
write
stop

Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.


maler (1 = "m01")
          geboren      31.7.1883
          gestorben    27.1.1970
          geschlecht   männlich
          id           m01
          vorname      Erich
          nachname     Heckel
          stilrichtung Expressionismus

   gruppe (1 = "gru-1")
             gruppenname  Die BrÃ¼cke
             gruendung    1905
             aufloesung   1913
             ort          Dresden und Berlin
             mitgliedschaft 1905-1913

maler (2 = "m02")
          geboren      6.5.1880
          gestorben    15.6.1938
          geschlecht   männlich
          id           m02
          vorname      Ernst Ludwig
          nachname     Kirchner
          stilrichtung Expressionismus

   gruppe (1 = "gru-1")
             gruppenname  Die BrÃ¼cke
             gruendung    1905
             aufloesung   1913
             ort          Dresden und Berlin
             mitgliedschaft 1905-1913

maler (3 = "m03")
          geboren      31.12.1869
```

```
          gestorben    3.11.1954
          geschlecht   männlich
          id           m03
          vorname      Henri
          nachname     Matisse
          stilrichtung Fauvismus

   gruppe (1 = "gru-1")
             gruppenname  Fauves
             gruendung    1905
             aufloesung   1909
             ort          Paris
             mitgliedschaft 1905-1909

maler (4 = "m04")
          geboren      16.10.1874
          gestorben    24.9.1930
          geschlecht   männlich
          id           m04
          vorname      Otto
          nachname     Mueller
          stilrichtung Expressionismus

   gruppe (1 = "gru-1")
             gruppenname  Die BrÃ¼cke
             gruendung    1905
             aufloesung   1913
             ort          Dresden und Berlin
             mitgliedschaft 1910-1913
. . .
Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.9. Step 9: Ausgabe einschränken

```
query name=maler;part=/maler
write part=:nachname,:vorname,:gestorben,:stilrichtung
stop
```

Erläuterung:

- Durch den part Partmeter wird auf die Gruppe maler zugegriffen.
- Der write Befehl gibt mit Hilfe des part Parameters weiter die Elemente nachname, vorname, gestorben und stilrichtung aus.
- Bei der Ausgabe werden Angaben zu den Künstlern angezeigt (in dem Fall Vor- und Nachname, Todesdatum und Stilrichtung).

Ausgabe ergebnis9.txt:

```
Kleio Version 10.0.0 --- 31.03.2005, 08:07:37
Historical Workstation Project / UNIX version
query name=maler;part=/maler
write part=:nachname,:vorname,:gestorben,:stilrichtung
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.

maler (1 = "m01") : nachname     Heckel
maler (1 = "m01") : vorname      Erich
maler (1 = "m01") : gestorben    27.1.1970
maler (1 = "m01") : stilrichtung Expressionismus
maler (2 = "m02") : nachname     Kirchner
maler (2 = "m02") : vorname      Ernst Ludwig
maler (2 = "m02") : gestorben    15.6.1938
maler (2 = "m02") : stilrichtung Expressionismus
maler (3 = "m03") : nachname     Matisse
maler (3 = "m03") : vorname      Henri
maler (3 = "m03") : gestorben    3.11.1954
maler (3 = "m03") : stilrichtung Fauvismus
maler (4 = "m04") : nachname     Mueller
maler (4 = "m04") : vorname      Otto
maler (4 = "m04") : gestorben    24.9.1930
maler (4 = "m04") : stilrichtung Expressionismus
maler (5 = "m05") : nachname     Nolde
maler (5 = "m05") : vorname      Emil
maler (5 = "m05") : gestorben    15.4.1956
maler (5 = "m05") : stilrichtung Expressionismus
maler (6 = "m06") : nachname     Schmidt-Rottluff
```

```
maler (6 = "m06") : vorname      Karl
maler (6 = "m06") : gestorben    10.8.1976
. . .
Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.10. Step 10: Ausgabe mit Hilfe von Packages verpacken

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=famname;usage=package;overwrite=yes
fields start="<br><b>Nachname</b><i> ";
       limit="</i>"
exit name=famname
item name=vorname;usage=package;overwrite=yes
fields start="<br><b>Vorname</b><i> ";
       limit="</i>"
exit name=vorname
item name=todesdatum;usage=package;overwrite=yes
fields start="<br><b>Todesdatum</b><i> ";
       limit="</i>"
exit name=todesdatum
item name=stil;usage=package;overwrite=yes
fields start="<br><b>Stilrichtung</b><i> ";
       limit="</i>"
exit name=stil

query name=maler;part=/maler
write part=:package[:nachname,famname],
           :package[:vorname,vorname],
           :package[:gestorben,todesdatum],
           :package[:stilrichtung,stil]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Es werden vier neue Packages definiert, um die Anzeige im Webbrowser etwas

eleganter zu gestalten (Package famname, vorname, todesdatum und stil).

- Per index Befehl wird auf die jeweilige Package Einbaufunktion zugegriffen. Innerhalb dieser Einbaufunktionen wird dann jedes Element der Datenbank mit seinem zugehörigen Package in Beziehung gesetzt.
- Die Ausgabe wird in die inhalt.html geschrieben.
- Bei der Ausgabe werden Angaben zu den Künstlern angezeigt (in dem Fall Vor- und Nachname, Todesdatum und Stilrichtung).

Quellcode der inhalt.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>

<br><b>Nachname</b><i> Heckel</i>
<br><b>Vorname</b><i> Erich</i>
<br><b>Todesdatum</b><i> 27.1.1970</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<br><b>Nachname</b><i> Kirchner</i>
<br><b>Vorname</b><i> Ernst Ludwig</i>
<br><b>Todesdatum</b><i> 15.6.1938</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<br><b>Nachname</b><i> Matisse</i>
<br><b>Vorname</b><i> Henri</i>
<br><b>Todesdatum</b><i> 3.11.1954</i>
<br><b>Stilrichtung</b><i> Fauvismus</i>
<br><b>Nachname</b><i> Mueller</i>
<br><b>Vorname</b><i> Otto</i>
<br><b>Todesdatum</b><i> 24.9.1930</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<br><b>Nachname</b><i> Nolde</i>
<br><b>Vorname</b><i> Emil</i>
. . .
<br><b>Nachname</b><i> Meidner</i>
<br><b>Vorname</b><i> Ludwig</i>
<br><b>Todesdatum</b><i> 14.5.1966</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<br><b>Nachname</b><i> Pascin</i>
```

```
<br><b>Vorname</b><i> Jules</i>
<br><b>Todesdatum</b><i> 20.6.1930</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
</body>
</html>
```

Ausgabe der inhalt.html im Webbrowser:

```
Nachname Heckel
Vorname Erich
Todesdatum 27.1.1970
Stilrichtung Expressionismus
Nachname Kirchner
Vorname Ernst Ludwig
Todesdatum 15.6.1938
Stilrichtung Expressionismus
Nachname Matisse
Vorname Henri
Todesdatum 3.11.1954
Stilrichtung Fauvismus
Nachname Mueller
Vorname Otto
Todesdatum 24.9.1930
Stilrichtung Expressionismus
Nachname Nolde
Vorname Emil
Todesdatum 15.4.1956
Stilrichtung Expressionismus
. . .
```

4.11. Step 11: Zusätzliches verändern der Ausgabe mit der Einbaufunktion form [ ]

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=famname;usage=package;overwrite=yes
fields start="<br><b>Nachname</b><i> ";
       limit="</i>"
```

```
exit name=famname
item name=vorname;usage=package;overwrite=yes
fields start="<br><b>Vorname</b><i> ";
       limit="</i>"
exit name=vorname
item name=todesdatum;usage=package;overwrite=yes
fields start="<br><b>Todesdatum</b><i> ";
       limit="</i>"
exit name=todesdatum
item name=stil;usage=package;overwrite=yes
fields start="<br><b>Stilrichtung</b><i> ";
       limit="</i>"
exit name=stil

query name=maler;part=/maler
write part=:form["<p>MalerInnen"],
           :package[:nachname,famname],
           :package[:vorname,vorname],
           :package[:gestorben,todesdatum],
           :package[:stilrichtung,stil]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Wie Step 10 bis zur Datenbankabfrage.
- Der write Befehl wird nun weiter durch die Einbaufunktion :form [ ] gegliedert. Mit form können nicht komplexe Zeichenketten ausgeschrieben werden.
- Bei der Ausgabe werden Angaben zu den Künstlern angezeigt (in dem Fall Vor- und Nachname, Todesdatum und Stilrichtung).

Quellcode der inhalt.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>

<p>MalerInnen
<br><b>Nachname</b><i> Heckel</i>
```

```
<br><b>Vorname</b><i> Erich</i>
<br><b>Todesdatum</b><i> 27.1.1970</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<p>MalerInnen
<br><b>Nachname</b><i> Kirchner</i>
<br><b>Vorname</b><i> Ernst Ludwig</i>
<br><b>Todesdatum</b><i> 15.6.1938</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
<p>MalerInnen
<br><b>Nachname</b><i> Matisse</i>
<br><b>Vorname</b><i> Henri</i>
<br><b>Todesdatum</b><i> 3.11.1954</i>
<br><b>Stilrichtung</b><i> Fauvismus</i>
<p>MalerInnen
<br><b>Nachname</b><i> Mueller</i>
<br><b>Vorname</b><i> Otto</i>
<br><b>Todesdatum</b><i> 24.9.1930</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
. . .
<p>MalerInnen
<br><b>Nachname</b><i> Pascin</i>
<br><b>Vorname</b><i> Jules</i>
<br><b>Todesdatum</b><i> 20.6.1930</i>
<br><b>Stilrichtung</b><i> Expressionismus</i>
</body>
</html>
```

Ausgabe der inhalt.html im Webbrowser:

MalerInnen
Nachname *Heckel*
Vorname *Erich*
Todesdatum *27.1.1970*
Stilrichtung *Expressionismus*
MalerInnen
Nachname *Kirchner*
Vorname *Ernst Ludwig*
Todesdatum *15.6.1938*
Stilrichtung *Expressionismus*
MalerInnen
Nachname *Matisse*
Vorname *Henri*
Todesdatum *3.11.1954*
Stilrichtung *Fauvismus*

```
. . .
```

4.11.1. Erweiterung zu Step 11:

```
query name=maler;part=/gruppe
index part=:form["   "];
   part=:ort;
   part=:gruppenname
stop
```

Erläuterung:

- Durch den part Partmeter wird auf das Gruppenelement gruppe zugegriffen.
- Nach dem index Befehl wird die Einbaufunktion form [ ] angeführt, mit deren Hilfe man hier die Ausgabe im Aussehen beeinflusst. Die Elemente ort und gruppenname werden in alphabetischer Reihenfolge ausgeschrieben.
- Bei der Ausgabe werden alle Orte, an denen die Künstlergruppen ansässig waren, aufgeführt. Nach den Ortsnamen folgen die Gruppennamen. Die Anzeige beginnt mit einem Zeilenvorschub.

Ausgabe ergebnis11_1.txt:

```
Kleio Version 10.0.0 --- 28.03.2005, 19:59:59
Historical Workstation Project / UNIX version
query name=maler;part=/gruppe
index part=:form["   "];
   part=:ort;
   part=:gruppenname
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
    Brüssel Brabanter Fauvisten m

    Brüssel Brabanter Fauvisten m

    Brüssel Brabanter Fauvisten m
```

```
    Brüssel Brabanter Fauvisten m

    Brüssel Brabanter Fauvisten m

    Dresden und Berlin Die Brücke m

    Dresden und Berlin Die Brücke m

    Dresden und Berlin Die Brücke m

    Dresden und Berlin Die Brücke m

    Dresden und Berlin Die Brücke m

    Dresden und Berlin Die Brücke m

    Murnau Der blaue Reiter m

    Murnau Der blaue Reiter m

    Murnau Der blaue Reiter m

    Murnau Der blaue Reiter m
. . .
```

4.11.2. Erweiterung zu Step 11:

```
query name=maler;part=/gruppe
index part=:gruppenname;form=right;
   part=:gruendung;form=right;
   part=:ort;form=right
stop
```

Erläuterung:

- Durch den part Partmeter wird auf das Gruppenelement gruppe zugegriffen.
- Durch den index Befehl werden die Elemente gruppenname, gruendung und ort ausgeschrieben. Durch form=right wird angeordnet, dass die Ausgabe rechtsbündig dargestellt werden soll.

- Bei der Ausgabe wird die Gruppenzugehörigkeit der Maler untersucht. Es werden Gruppenname, Gründungsjahr und Ort der Künstlervereinigung ausgeschrieben.

Ausgabe ergebnis11_2.txt:

```
Kleio Version 10.0.0 --- 30.03.2005, 09:59:07
Historical Workstation Project / UNIX version
query name=maler;part=/gruppe
index part=:gruppenname;form=right;
   part=:gruendung;form=right;
   part=:ort;form=right
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
            Fauves                                  Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

            Fauves               1905               Paris m

       Die Brücke              1905 Dresden und Berlin m

       Die Brücke              1905 Dresden und Berlin m
```

```
       Die Brücke               1905 Dresden und Berlin m
. . .
Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.12. Step 12: Ausgabe einschränken mit der Einbaufunktion each [ ]

```
query name=maler;part=/maler
write part=:each[]
stop
```

Erläuterung:

- Durch den part Partmeter wird auf die Gruppe maler zugegriffen.
- Der write Befehl wird weiter durch die Einbaufunktion each [ ] spezifiziert. Durch each [ ] werden alle Elemente, aber keine durch eine Hierarchie abhängigen Gruppen oder Elemente ausgegeben. Man kann also nur auf die zuletzt angesprochene Informationsgruppe Bezug nehmen.
- Bei der Ausgabe werden alle Maler angezeigt, ohne ihre Zugehörigkeit zu einer Künstlergruppe.

Ausgabe ergebnis12.txt:

```
Kleio Version 10.0.0 --- 31.03.2005, 09:05:37
Historical Workstation Project / UNIX version
query name=maler;part=/maler
write part=:each[]
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.


maler (1 = "m01")
          geboren      31.7.1883
          gestorben    27.1.1970
```

```
          geschlecht   männlich
          id           m01
          vorname      Erich
          nachname     Heckel
          stilrichtung Expressionismus

maler (2 = "m02")
          geboren      6.5.1880
          gestorben    15.6.1938
          geschlecht   männlich
          id           m02
          vorname      Ernst Ludwig
          nachname     Kirchner
          stilrichtung Expressionismus

maler (3 = "m03")
          geboren      31.12.1869
          gestorben    3.11.1954
          geschlecht   männlich
          id           m03
          vorname      Henri
          nachname     Matisse
          stilrichtung Fauvismus

maler (4 = "m04")
          geboren      16.10.1874
          gestorben    24.9.1930
          geschlecht   männlich
          id           m04
          vorname      Otto
          nachname     Mueller
          stilrichtung Expressionismus
. . .
```

4.13. Step 13: Ausgabe verpacken mit each [ ]

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei
```

```
item name=famname;usage=package;overwrite=yes
fields start="<br><b>Nachname</b><i> ";
       limit="</i>"
exit name=famname
item name=vorname;usage=package;overwrite=yes
fields start="<br><b>Vorname</b><i> ";
       limit="</i>"
exit name=vorname
item name=todesdatum;usage=package;overwrite=yes
fields start="<br><b>Todesdatum</b><i> ";
       limit="</i>"
exit name=todesdatum
item name=stil;usage=package;overwrite=yes
fields start="<br><b>Stilrichtung</b><i> ";
       limit="</i>"
exit name=stil

query name=maler;part=/maler
write part=:each[=packaged,
     :nachname,famname,
     :vorname,vorname,
     :gestorben,todesdatum,
     :stilrichtung,stil
     ]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Wie Step 10 bis zur Datenbankabfrage.
- Der write Befehl wird nun weiter durch die Einbaufunktion :each […] spezifiziert. **each**[=packaged,:element1,package1,:element2,package2] → Verpackt jeden Mehrfacheintrag extra.
- Bei der Ausgabe werden Angaben zu den Künstlern angezeigt (in dem Fall Vor- und Nachname, Todesdatum und Stilrichtung).

Quellcode der inhalt.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>


maler (1 = "m01")
          <br><b>Todesdatum</b><i> 27.1.1970</i>
          <br><b>Vorname</b><i> Erich</i>
          <br><b>Nachname</b><i> Heckel</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i>

maler (2 = "m02")
          <br><b>Todesdatum</b><i> 15.6.1938</i>
          <br><b>Vorname</b><i> Ernst Ludwig</i>
          <br><b>Nachname</b><i> Kirchner</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i>

maler (3 = "m03")
          <br><b>Todesdatum</b><i> 3.11.1954</i>
          <br><b>Vorname</b><i> Henri</i>
          <br><b>Nachname</b><i> Matisse</i>
          <br><b>Stilrichtung</b><i> Fauvismus</i>

maler (4 = "m04")
          <br><b>Todesdatum</b><i> 24.9.1930</i>
          <br><b>Vorname</b><i> Otto</i>
          <br><b>Nachname</b><i> Mueller</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i>
. . .
maler (65 = "m65")
          <br><b>Todesdatum</b><i> 20.6.1930</i>
          <br><b>Vorname</b><i> Jules</i>
          <br><b>Nachname</b><i> Pascin</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i>
</body>
</html>
```

Ausgabe der inhalt.html im Webbrowser:

maler (1 = "m01")
Todesdatum *27.1.1970*

```
Vorname Erich
Nachname Heckel
Stilrichtung Expressionismus maler (2 = "m02")
Todesdatum 15.6.1938
Vorname Ernst Ludwig
Nachname Kirchner
Stilrichtung Expressionismus maler (3 = "m03")
Todesdatum 3.11.1954
Vorname Henri
Nachname Matisse
Stilrichtung Fauvismus maler (4 = "m04")
Todesdatum 24.9.1930
Vorname Otto
Nachname Mueller
. . .
```

4.14. Step 14: Ausgabe differenzierter formatieren mit each [ ]

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei


item name=famname;usage=package;overwrite=yes
fields start="<br><b>Nachname</b><i> ";
       limit="</i>"
exit name=famname
item name=vorname;usage=package;overwrite=yes
fields start="<br><b>Vorname</b><i> ";
       limit="</i>"
exit name=vorname
item name=todesdatum;usage=package;overwrite=yes
fields start="<br><b>Todesdatum</b><i> ";
       limit="</i>"
exit name=todesdatum
item name=stil;usage=package;overwrite=yes
fields start="<br><b>Stilrichtung</b><i> ";
       limit="</i>"
exit name=stil

item name=gruppe;usage=package;overwrite=yes
```

```
fields start="MalerInnen";
       limit="<p><hr><p>"
exit name=gruppe

query name=maler;part=/maler
write part=:each[=header,gruppe,
     =packaged,
     :nachname,famname,
     :vorname,vorname,
     :gestorben,todesdatum,
     :stilrichtung,stil
     ]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Wie Step 10 bis zur Datenbankabfrage.
- Durch den write Befehl wird die Einbaufunktion each [ ] aufgerufen. Dabei gilt: **each** [=header,package] → Verpackt eine ganze Gruppe. Die im Package definierten Zeichenketten umschließen also alles, was durch :each [ ] ausgegeben wird.
- **each** [=packaged,:element1,package1,:element2,package2] → Verpackt jeden Mehrfacheintrag extra.
- Bei der Ausgabe werden Angaben zu den Künstlern angezeigt (in dem Fall Vor- und Nachname, Todesdatum und Stilrichtung).

Quellcode der inhalt.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>

MalerInnen
          <br><b>Todesdatum</b><i> 27.1.1970</i>
          <br><b>Vorname</b><i> Erich</i>
          <br><b>Nachname</b><i> Heckel</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
```

```
MalerInnen
          <br><b>Todesdatum</b><i> 15.6.1938</i>
          <br><b>Vorname</b><i> Ernst Ludwig</i>
          <br><b>Nachname</b><i> Kirchner</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
MalerInnen
          <br><b>Todesdatum</b><i> 3.11.1954</i>
          <br><b>Vorname</b><i> Henri</i>
          <br><b>Nachname</b><i> Matisse</i>
          <br><b>Stilrichtung</b><i> Fauvismus</i><p><hr><p>
MalerInnen
          <br><b>Todesdatum</b><i> 24.9.1930</i>
          <br><b>Vorname</b><i> Otto</i>
          <br><b>Nachname</b><i> Mueller</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
. . .
MalerInnen
          <br><b>Todesdatum</b><i> 20.6.1930</i>
          <br><b>Vorname</b><i> Jules</i>
          <br><b>Nachname</b><i> Pascin</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
</body>
</html>
```

Ausgabe der inhalt.html im Webbrowser:

MalerInnen
Todesdatum *27.1.1970*
Vorname *Erich*
Nachname *Heckel*
Stilrichtung *Expressionismus*

MalerInnen
Todesdatum *15.6.1938*
Vorname *Ernst Ludwig*
Nachname *Kirchner*
Stilrichtung *Expressionismus*

MalerInnen
Todesdatum *3.11.1954*
Vorname *Henri*
Nachname *Matisse*
Stilrichtung *Fauvismus*

```
MalerInnen
      Todesdatum 24.9.1930
      Vorname Otto
      Nachname Mueller
      Stilrichtung Expressionismus
. . .
```

4.15. Step 15: Ausgabe mit each [ ] / Schlüsselwort default

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei


item name=famname;usage=package;overwrite=yes
fields start="<br><b>Nachname</b><i> ";
       limit="</i>"
exit name=famname
item name=vorname;usage=package;overwrite=yes
fields start="<br><b>Vorname</b><i> ";
       limit="</i>"
exit name=vorname
item name=todesdatum;usage=package;overwrite=yes
fields start="<br><b>Todesdatum</b><i> ";
       limit="</i>"
exit name=todesdatum
item name=stil;usage=package;overwrite=yes
fields start="<br><b>Stilrichtung</b><i> ";
       limit="</i>"
exit name=stil
item name=standard;usage=package;overwrite=yes
fields start="<br><b>Sonstiges</b><i> ";
       limit="</i>"
exit name=standard

item name=gruppe;usage=package;overwrite=yes
fields start="MalerInnen";
       limit="<p><hr><p>"
exit name=gruppe

query name=maler;part=/maler
```

```
write part=:each[=header,gruppe,
     =packaged,
        :nachname,famname,
        :vorname,vorname,
        :gestorben,todesdatum,
        :stilrichtung,stil,
     =default,standard
     ]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Wie Step 10 bis zur Datenbankabfrage.
- Durch den write Befehl wird die Einbaufunktion each [ ] aufgerufen. Dabei gilt: **each** [=default,package] → Es werden alle Einträge in eigenen Packages verpackt, bis auf alle übrigen Elemente der Gruppe, die dann durch das Package standard verpackt werden. Das Schlüsselwort =default kann auch ohne explizit angeführtes Element verwendet werden.
- Bei der Ausgabe werden nun alle Angaben zu den Künstlern angezeigt.

Quellcode der inhalt.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body>

MalerInnen
          <br><b>Sonstiges</b><i> 31.7.1883</i>
          <br><b>Todesdatum</b><i> 27.1.1970</i>
          <br><b>Sonstiges</b><i> m&auml;nnlich</i>
          <br><b>Sonstiges</b><i> m01</i>
          <br><b>Vorname</b><i> Erich</i>
          <br><b>Nachname</b><i> Heckel</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
MalerInnen
          <br><b>Sonstiges</b><i> 6.5.1880</i>
          <br><b>Todesdatum</b><i> 15.6.1938</i>
```

```
          <br><b>Sonstiges</b><i> m&auml;nnlich</i>
          <br><b>Sonstiges</b><i> m02</i>
          <br><b>Vorname</b><i> Ernst Ludwig</i>
          <br><b>Nachname</b><i> Kirchner</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
MalerInnen
          <br><b>Sonstiges</b><i> 31.12.1869</i>
          <br><b>Todesdatum</b><i> 3.11.1954</i>
          <br><b>Sonstiges</b><i> m&auml;nnlich</i>
          <br><b>Sonstiges</b><i> m03</i>
          <br><b>Vorname</b><i> Henri</i>
          <br><b>Nachname</b><i> Matisse</i>
          <br><b>Stilrichtung</b><i> Fauvismus</i><p><hr><p>
. . .
MalerInnen
          <br><b>Sonstiges</b><i> 31.3.1885</i>
          <br><b>Todesdatum</b><i> 20.6.1930</i>
          <br><b>Sonstiges</b><i> m&auml;nnlich</i>
          <br><b>Sonstiges</b><i> m65</i>
          <br><b>Vorname</b><i> Jules</i>
          <br><b>Nachname</b><i> Pascin</i>
          <br><b>Stilrichtung</b><i> Expressionismus</i><p><hr><p>
</body>
</html>
```

Anzeige der inhalt.html im Webbrowser:

MalerInnen
Sonstiges *31.7.1883*
Todesdatum *27.1.1970*
Sonstiges *männlich*
Sonstiges *m01*
Vorname *Erich*
Nachname *Heckel*
Stilrichtung *Expressionismus*

MalerInnen
Sonstiges *6.5.1880*
Todesdatum *15.6.1938*
Sonstiges *männlich*
Sonstiges *m02*
Vorname *Ernst Ludwig*
Nachname *Kirchner*
Stilrichtung *Expressionismus*

```
MalerInnen
      Sonstiges 31.12.1869
      Todesdatum 3.11.1954
      Sonstiges männlich
      Sonstiges m03
      Vorname Henri
      Nachname Matisse
      Stilrichtung Fauvismus
. . .
```

4.16. Step 16: Zusätzliche Angaben der Fields Direktive

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

query name=maler;part=/maler
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop target="inhalt.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Die Packages Gruppen und Elemente sind neu. In ihnen befinden sich Formatierungsanweisungen für diverse HTML-Code Sequenzen, die eine Tabelle aufbauen.

- Bisher wurden bei den Packages in der Fields Direktive nur die Parameter start und limit verwendet. Durch die neuen Parameter write, connect, origin und complete lässt sich die Formatierung noch eingehender steuern.

- Ausgabe von Elementen durch die Fieldsparameter start, limit und write.

  ➔ Bei start wird eine Zeichenkette bzw. Konstante vor einem Element ausgegeben.

  ➔ Bei limit wird die Zeichenkette/Konstante nach einem Element ausgegeben.

  ➔ Bei write wird der Wert vor dem Tagnamen eines Elements ausgegeben.

- Ausgabe von Gruppen durch die Fieldsparameter origin, connect und complete.

  ➔ Bei origin wird der Inhalt einer Zeichenkette/Konstante vor dem ersten Element einer Gruppe ausgegeben. origin bezieht sich somit auf die Ausgabe sämtlicher in dieser Gruppe enthaltenen Elemente und Gruppen.

  ➔ Bei connect wird der Wert zwischen dem Tagnamen eines Elements und dem Elementinhalt ausgegeben.

  ➔ Bei complete wird der Inhalt einer Zeichenkette/Konstante nach dem Element einer Gruppe ausgegeben. complete umfasst die Ausgabe sämtlicher in dieser Gruppe ausgewählten Elemente.

- Bei der Ausgabe durch den write Befehl werden die Packages Gruppen und Elemente innerhalb der Einbaufunktion each [ ] aufgerufen. Hierbei ist die Groß- und Kleinschreibung der Packagenamen jedoch gleichgültig, da alle Kleio Namen nicht case sensitiv sind.

- Bei der Browseranzeige werden nun alle Angaben zu den Künstlern aufgeführt. Weil die Packages per Standard verpackt werden, wird als Bezeichnung für die Inhalte der Elemente nur die Tagbezeichnung der XML Daten ausgeschrieben.

Quellcode der inhalt.html:

```
<html>
<html>
<title>Liste der Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
 <td width="80%"><i>31.7.1883</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
 <td width="80%"><i>27.1.1970</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
 <td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m01</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Erich</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Heckel</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>

<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>6.5.1880</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>15.6.1938</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m02</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Ernst Ludwig</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Kirchner</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>

<table>
<h3>maler</h3>
```

```
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>31.12.1869</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>3.11.1954</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m03</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Henri</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Matisse</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Fauvismus</i></td></tr></table>
. . .
<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>31.3.1885</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>20.6.1930</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m65</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Jules</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Pascin</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der inhalt.html im Webbrowser:

maler

geboren *31.7.1883*
gestorben *27.1.1970*
geschlecht *männlich*
id *m01*
vorname *Erich*
nachname *Heckel*
stilrichtung *Expressionismus*

```
. . .
```

4.17. Step 17: Ausgabe zerlegen in Einzeldateien durch die Einbaufunktion output [ ]

```
item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

item name=personendatei;usage=package;overwrite=yes
fields start="personen/";
       limit=".html"
exit name=personendatei

item name=personenmarkup;usage=package;overwrite=yes
fields start="<html>\n<title>Ein Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=personenmarkup

query name=maler;part=/maler
write part=/output[:package[:id,personendatei],overwrite,personenmarkup,html]
     :each[=header,gruppen,
        =packaged,
        =default,elemente
        ]
stop
```

Erläuterung:

- Ziel der Aufgabe ist es, alle Gruppenelemente in einzelne Dateien zu zerlegen.
- Das Ziel für die Ausgabe wird dabei näher im Package personendatei definiert. Für die

Ausgabe wird durch die Fields Direktive mit dem start Parameter ein neues Verzeichnis mit dem Namen „personen“ angelegt. Alle Dateien im Verzeichnis personen erhalten eine „.html“ Endung.

- Das Package personenmarkup definiert das Grundgerüst einer HTML-Seite.
- Durch den write Befehl wird die Einbaufunktion /output […] aufgerufen. Dabei gilt die Form:

  … / output [<Namensquelle>, <Verhalten> , <Dateiverpackung> , <Zeichensatz>]

  <Namensquelle> definiert den neuen Dateinamen, in unserem Fall das Package personendatei zusammen mit dem Element id.

  <Verhalten> definiert den Modus, wenn die Datei bereits existiert (preserve, overwrite, append, cumulate). In unserem Fall overwrite.

  <Dateiverpackung> definiert das Package, das alle Daten umschließt. In unserem Fall personenmarkup.

  <Zeichensatz> definiert die Behandlung von Sonderzeichen. In unserem Fall html.

- Bei der Ausgabe werden nun alle Angaben zu einem Künstler in eine einzelne HTML-Datei im Verzeichnis „personen“ geleitet.

Quellcode m01.html:

```
<html>
<title>Ein Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>31.7.1883</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>27.1.1970</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
```

```
<tr><td width="20%"><b>id</b></td> <td width="80%"><i>m01</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Erich</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Heckel</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der m01.html im Webbrowser:

maler
- **geboren** *31.7.1883*
- **gestorben** *27.1.1970*
- **geschlecht** *männlich*
- **id** *m01*
- **vorname** *Erich*
- **nachname** *Heckel*
- **stilrichtung** *Expressionismus*

4.18. Step 18: Verweise auf Daten

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="<li>";
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</li>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
fields start="<br>Die Angaben zu dieser Person finden sich in der Datei:
personen/";
       limit=".html"
exit name=verweis
```

```
query name=maler;part=/maler
index part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
      part=:package[:id,verweis];
      type=list
stop target="register.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Ziel der Aufgabe ist es, eine Bezugnahme des Verzeichnisses auf die Daten aufzuzeigen.
- Durch den index Befehl wird das Package verweis zusammen mit dem Element id verarbeitet.
- Bei der Ausgabe wird nun eine Liste der Künstlernamen erstellt, die ebenso das Verzeichnis anzeigt, indem die kompletten Angaben zu dem betreffenden Künstler zu finden sind.

Quellcode register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body><ul>
<li>am Ende, Hans</li> <br>Die Angaben zu dieser Person finden sich in der
Datei: personen/m44.html

<li>Barlach, Ernst</li> <br>Die Angaben zu dieser Person finden sich in der
Datei: personen/m52.html

<li>Beckmann, Max</li> <br>Die Angaben zu dieser Person finden sich in der
Datei: personen/m41.html

<li>Bock, Marie</li> <br>Die Angaben zu dieser Person finden sich in der Datei:
personen/m50.html
. . .
<li>Wouters, Rik</li> <br>Die Angaben zu dieser Person finden sich in der Datei:
personen/m30.html
```

```
</ul></body>
</html>
```

Anzeige der register.html im Webbrowser:

```
• am Ende, Hans
      Die Angaben zu dieser Person finden sich in der Datei: personen/m44.html
• Barlach, Ernst
      Die Angaben zu dieser Person finden sich in der Datei: personen/m52.html
• Beckmann, Max
      Die Angaben zu dieser Person finden sich in der Datei: personen/m41.html
• Bock, Marie
      Die Angaben zu dieser Person finden sich in der Datei: personen/m50.html
. . .
```

4.19. Step 19: Verlinkung vom Verzeichnis auf Einzeldateien

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="<li>";
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</li>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
fields start='<a href="personen/';
       first='.html">';
       limit="</a>"
exit name=verweis

query name=maler;part=/maler
index part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
```

```
      part=:package[:id,verweis];
      type=list
stop target="register.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Ziel der Aufgabe ist es, eine Bezugnahme des Verzeichnisses auf die Daten aufzuzeigen. Diesmal mit einer Linkgenerierung von dem Verzeichnis zu den Daten.
- Das Package verweis definiert durch das href-Attribut einen Verweis auf die jeweiligen Künstlerdaten. Beim Package verweis wird die Ausgabe des Elements id zweimal angezeigt. Durch den Fieldsparameter start wird der Wert der Zeichenkette, in dem Fall , nach dem Element id ausgegeben. Der Inhalt des Fieldsparameter limit, hier , wird einzeln zum Abschluss des Package verweis angezeigt.
- Bei der Ausgabe wird nun eine Liste der Künstlernamen erstellt, die ebenso das gelinkte Verzeichnis anzeigt, indem die kompletten Angaben zu dem betreffenden Künstler zu finden sind.

Quellcode register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body><ul>
<li>am Ende, Hans</li> <a href="personen/m44.html">m44</a>

<li>Barlach, Ernst</li> <a href="personen/m52.html">m52</a>

<li>Beckmann, Max</li> <a href="personen/m41.html">m41</a>

<li>Bock, Marie</li> <a href="personen/m50.html">m50</a>
. . .
<li>Wouters, Rik</li> <a href="personen/m30.html">m30</a>



</ul></body>
```

```
</html>
```

Anzeige der register.html im Webbrowser:

- am Ende, Hans
  m44
- Barlach, Ernst
  m52
- Beckmann, Max
  m41
- Bock, Marie
  m50
- Braque, Georges
  m12

4.20. Step 20: Variante zu Step 19

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="";
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</a>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
fields start='<li><a href="personen/';
       limit='.html">';
exit name=verweis

query name=maler;part=/maler
index part=:package[:id,verweis];without=yes;
      part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
      type=list
stop target="register.html";overwrite=yes;
```

```
    package=htmldatei;form=html;
```

Erläuterung:

- Ziel der Aufgabe ist es, eine Bezugnahme des Verzeichnisses auf die Daten aufzuzeigen. Diesmal mit einer Linkgenerierung von dem Verzeichnis zu den Daten.
- Das Package verweis definiert die HTML-Anzeige einer Personenanzeige. Durch das href-Attribut wird ein Verweis auf die Künstlernamen gelegt.
- Bei der Ausgabe wird nun eine Liste der gelinkten Künstlernamen erstellt. Der Link führt in das Verzeichnis, indem die kompletten Angaben zu dem betreffenden Künstler zu finden sind.

Quellcode register.html:

```
<html>
<title>Liste der Maler</title>
</head>
<body><ul>
<li><a href="personen/m44.html"> am Ende, Hans</a>

<li><a href="personen/m52.html"> Barlach, Ernst</a>

<li><a href="personen/m41.html"> Beckmann, Max</a>

<li><a href="personen/m50.html"> Bock, Marie</a>
. . .
<li><a href="personen/m30.html"> Wouters, Rik</a>


</ul></body>
</html>
```

Anzeige der register.html im Webbrowser:

- am Ende, Hans
- Barlach, Ernst
- Beckmann, Max
- Bock, Marie

```
• Braque, Georges
```

4.21. Step 21: Einfache Suche nach einer Zeichenkette

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

query name=maler;part=/maler:nachname="Modersohn"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop target="ergebnis.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Ziel der Aufgabe ist es, eine Person oder bestimmte Personen in den Einträgen der Datenbank zu suchen.
- Die Packages bleiben unverändert, erst die Datenbankabfrage weicht geringfügig ab. Nach query name=maler; wird durch den part Parameter auf das Gruppenelement maler zugegriffen. Von diesem Gruppenelement wird noch mal speziell das Element nachname abgefragt. Bei diesem Element wird nach der Zeichenkette „Modersohn“

verlangt.

- Bei der Ausgabe wird nun eine Liste der Künstlernamen erstellt, die die Zeichenkette „Modersohn“ im Nachnamen tragen.

Quellcode ergebnis.html:

```
<html>
<title>Bericht &uuml;ber einen Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>8.2.1876</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>20.11.1907</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>weiblich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m38</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Paula</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Modersohn-Becker</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>

<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>22.2.1865</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>10.3.1943</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m39</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Otto</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
```

```
<td width="80%"><i>Modersohn</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der ergebnis.html im Webbrowser:

maler

geboren *8.2.1876*
gestorben *20.11.1907*
geschlecht *weiblich*
id *m38*
vorname *Paula*
nachname *Modersohn-Becker*
stilrichtung *Expressionismus*

maler

geboren *22.2.1865*
gestorben *10.3.1943*
geschlecht *männlich*
id *m39*
vorname *Otto*
nachname *Modersohn*
stilrichtung *Expressionismus*

4.22. Step 22: Einfache Suche mit Bedingungen (and)

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
```

```
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

query name=maler;part=/maler:nachname="Modersohn" and :vorname="Paula"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop target="ergebnis.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Ziel der Aufgabe ist es, eine Person in den Einträgen der Datenbank zu suchen.
- Die Packages bleiben unverändert, erst die Datenbankabfrage wandelt sich ab. Nach query name=maler; wird durch den part Parameter auf das Gruppenelement maler zugegriffen. Von diesem Gruppenelement wird noch mal speziell das Element nachname und vorname abgefragt. Bei nachname wird nach der Zeichenkette „Modersohn“ verlangt und bei vorname wird nach der Zeichenkette „Paula“. Verbunden sind beide Bedingungen durch den Operator and.
- Werden zwei Teilbedingungen durch den Operator *and* miteinander verbunden, müssen beide wahr sein, um die zusammengesetzte Bedingung zu erfüllen (Verknüpfungsoperator and). Steht jedoch vor einer Teilbedingung der Operator *not*, so darf diese Bedingung nicht erfüllt werden, damit sie als wahr behandelt wird (Negation von Bedingungen).
- Bei der Ausgabe wird nun eine Liste der Künstlernamen erstellt, die die Zeichenkette „Modersohn“ im Nachnamen und „Paula“ im Vornamen tragen.

Quellcode ergebnis.html:

```
<html>
<title>Bericht &uuml;ber einen Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>8.2.1876</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>20.11.1907</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>weiblich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m38</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Paula</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Modersohn-Becker</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der ergebnis.html im Webbrowser:

maler

geboren *8.2.1876*
gestorben *20.11.1907*
geschlecht *weiblich*
id *m38*
vorname *Paula*
nachname *Modersohn-Becker*
stilrichtung *Expressionismus*

4.22.1. Erweiterung zu Step 22 (and / not)

```
query    name=maler;part=:stilrichtung="Expressionismus"    and    :nachname=not
"Moderson"
write part=:vorname,:nachname,:stilrichtung
stop
```

Erläuterung:

- Durch den part Partmeter wird auf das Element stilrichtung zugegriffen. Eingegrenzt wird das Element stilrichtung mit der Zeichenkette “Expressionismus“, dem Verknüpfungsoperator *and* und der Bedingung, dass das Element nachname nicht die Zeichenkette “Moderson“ enthalten darf. Werden zwei Teilbedingungen durch das Schlüsselwort *and* miteinander verbunden, müssen beide wahr sein, um die zusammengesetzte Bedingung zu erfüllen (Verknüpfungsoperator and). Steht jedoch vor einer Teilbedingung das Schlüsselwort *no* oder *not*, so darf diese Bedingung nicht erfüllt werden, damit sie als wahr behandelt wird (Negation von Bedingungen).
- Der write Befehl wird weiter durch den Zugriff auf die Elemente vorname, nachname und stilrichtung spezifiziert.
- Bei der Ausgabe werden die Angaben vorname, nachname und stilrichtung der Maler angezeigt. Bedingungen: diese Maler gehören alle dem Expressionismus an und heißen nicht mit Nachnamen “Moderson“.

Ausgabe ergebnis22_1.txt:

```
Kleio Version 10.0.0 --- 23.03.2005, 09:18:13
Historical Workstation Project / UNIX version
query    name=maler;part=:stilrichtung="Expressionismus"    and    :nachname=not
"Moderson"
write part=:vorname,:nachname,:stilrichtung
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.

maler (1 = "m") : vorname      Erich
maler (1 = "m") : nachname     Heckel
```

```
maler (1 = "m") : stilrichtung Expressionismus
maler (2 = "m") : vorname      Ernst Ludwig
maler (2 = "m") : nachname     Kirchner
maler (2 = "m") : stilrichtung Expressionismus
maler (4 = "m") : vorname      Otto
maler (4 = "m") : nachname     Mueller
maler (4 = "m") : stilrichtung Expressionismus
maler (5 = "m") : vorname      Emil
maler (5 = "m") : nachname     Nolde
maler (5 = "m") : stilrichtung Expressionismus
maler (6 = "m") : vorname      Karl
maler (6 = "m") : nachname     Schmidt-Rottluff
maler (6 = "m") : stilrichtung Expressionismus
maler (16 = "m") : vorname      Wassily
maler (16 = "m") : nachname     Kandinsky
maler (16 = "m") : stilrichtung Expressionismus
maler (17 = "w") : vorname      Gabriele
maler (17 = "w") : nachname     MÃ¼nter
maler (17 = "w") : stilrichtung Expressionismus
maler (18 = "w") : vorname      Marianne
maler (18 = "w") : nachname     von Werefkin
maler (18 = "w") : stilrichtung Expressionismus
. . .
```

4.22.2. Erweiterung zu Step 22 (null)

```
query name=maler;part=/maler:stilrichtung=null
write part=:each[]
stop
```

Erläuterung:

- Durch den part Partmeter wird auf die Gruppe maler zugegriffen. Eingegrenzt wird die Gruppe durch das Element stilrichtung und das Vergleichsmuster *null*. Es existiert eine durch die Pfadbezeichnung spezifizierte Menge von Elementarinformationen. Beim Vergleichsmuster null darf kein Element aus dieser Menge existieren, damit die Bedingung als erfüllt für ein Element der Menge von enthaltenen Informationsgruppen angesehen wird.

- Der write Befehl wird weiter durch die Einbaufunktion each [ ] spezifiziert.
- Bei der Ausgabe werden nur die Maler angezeigt, bei denen keine weiteren Angaben zur stilrichtung gemacht wurden.

Ausgabe ergebnis22_2.txt:

```
Kleio Version 10.0.0 --- 27.03.2005, 17:21:56
Historical Workstation Project / UNIX version
query name=maler;part=/maler:stilrichtung=null
write part=:each[]
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.


maler (35 = "m")
          geboren      7.6.1848
          gestorben    8.5.1903
          geschlecht   männlich
          id           m35
          vorname      Paul
          nachname     Gauguin

maler (58 = "m")
          geboren      14.9.1883
          gestorben    4.11.1908
          geschlecht   männlich
          id           m58
          vorname      Richard
          nachname     Gerstl

maler (60 = "m")
          geboren      24.7.1892
          gestorben    11.4.1971
          geschlecht   männlich
          id           m60
          vorname      Marcel
          nachname     Gromaire

maler (65 = "m")
          geboren      31.3.1885
```

```
          gestorben    20.6.1930
          geschlecht   männlich
          id           m65
          vorname      Jules
          nachname     Pascin
. . .
```

4.22.3. Erweiterung zu Step 22 (not null)

```
query name=maler;part=/maler:stilrichtung=not null
write part=:each[]
stop
```

Erläuterung:

- Durch den part Partmeter wird auf die Gruppe maler zugegriffen. Eingegrenzt wird die Gruppe durch das Element stilrichtung und das Vergleichsmuster *not null*. Wird das Vergleichsmuster *null* negiert, muss mindestens ein Element der durch den letzten Bestandteil der Pfadbezeichnung definierten Menge von Elementarinformationen existieren.
- Der write Befehl wird weiter durch die Einbaufunktion each [ ] spezifiziert.
- Bei der Ausgabe werden alle Maler angezeigt, bei denen weitere Angaben zur stilrichtung gemacht wurden.

Ausgabe ergebnis22_3.txt:

```
Kleio Version 10.0.0 --- 27.03.2005, 18:04:28
Historical Workstation Project / UNIX version
query name=maler;part=/maler:stilrichtung=not null
write part=:each[]
stop


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
```

```
maler (1 = "m")
          geboren      31.7.1883
          gestorben    27.1.1970
          geschlecht   männlich
          id           m01
          vorname      Erich
          nachname     Heckel
          stilrichtung Expressionismus

maler (2 = "m")
          geboren      6.5.1880
          gestorben    15.6.1938
          geschlecht   männlich
          id           m02
          vorname      Ernst Ludwig
          nachname     Kirchner
          stilrichtung Expressionismus

maler (3 = "m")
          geboren      31.12.1869
          gestorben    3.11.1954
          geschlecht   männlich
          id           m03
          vorname      Henri
          nachname     Matisse
          stilrichtung Fauvismus

maler (4 = "m")
          geboren      16.10.1874
          gestorben    24.9.1930
          geschlecht   männlich
          id           m04
          vorname      Otto
          nachname     Mueller
          stilrichtung Expressionismus
. . .
```

4.23. Step 23: Einführung von Katalogen

```
query name=maler;part=/maler
catalogue name=personen;part=:nachname;
          type=term;overwrite=yes
stop
```

Erläuterung:

- Im Step 23 wird erstmals ein Katalog angelegt. Durch den Einsatz von Katalogen wird die Suchzeit bei einer Datenbankabfrage verkürzt, weil nur ein spezieller vorgefertigter Teil der Datenbank abgefragt wird. Ein Katalog funktioniert wie ein Index, ist jedoch sehr viel mächtiger.

- Es wird bei der Query Abfrage durch part ein bestimmter Teil der Datenbank festgelegt, den der Katalog umfasst. Der Katalog zeigt hier an, worauf sich der aktuelle Pfad bezieht. Später kann anhand eines Pfades bei einer write Abfrage auf diese Hierarchieebene verwiesen werden. Per Einbaufunktion back [ ] kann man jeweils zwischen den Pfaden springen. Die Pfade verweisen auf die unterschiedlichen Hierarchieebenen in der *.mod Datei.

- Der Katalogname ist in unserem Fall personen. Durch den part Parameter wird auf das Element nachname zugegriffen.

- type=term bedeutet, dass ein Eintrag als Ganzes in den Katalog eingehen soll, also keine Zerlegung in Einzelworte. Ein Eintrag darf dabei 255 Zeichen lang sein. overwrite=yes bewirkt, dass der bestehende Inhalt stets durch einen neu erstellten Inhalt (bzw. durch neue Abfragen) überschrieben werden darf.

4.24. Step 24: Abfrage mit Hilfe eines Katalogs

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen
```

```
item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

query name=maler;part=/catalogue[personen,complete,"Modersohn"]
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop target="ergebnis.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Die Packages htmldatei, Gruppen und Elemente sind bereits bekannt. Bei der Datenbank Query wird diesmal mit Hilfe der Gruppenfunktion catalogue [ ] im zuvor in Step 23 definierten Katalog personen gesucht. Es wird durch complete, dem zweiten Argument in der Katalogvereinbarung, definiert, dass genau nach dieser Zeichenkette gesucht wird. In der Gruppenfunktion catalogue [ ] wird schließlich noch nach der Zeichenkette bzw. Konstante „Modersohn“ gesucht.

- Das zweite Argument gibt jeweils an, welcher Zugriffsweg im Katalog gewählt werden soll. So soll bei:

  ➔ complete ein Term als Ganzes gesucht werden,

  ➔ start ein Term gesucht werden, der mit einer bekannten Zeichenkette anfängt (Suche nach dem Wortanfang),

  ➔ limit ein Term gesucht werden, der mit einer bekannten Zeichenkette endet (Suche nach dem Wortende),

  ➔ algorithm ein Term gesucht werden, der vorher in eine abgeleitete Form (Lemma, Soundex- oder Skeletierungscode) modifiziert wurde.

- Bei der Ausgabe werden alle Maler angezeigt, die in ihrem Datensatz eine Zeichenkette „Modersohn“ enthalten.

Quellcode ergebnis.html:

```
<html>
<title>Bericht &uuml;ber einen Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>22.2.1865</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>10.3.1943</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m39</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Otto</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Modersohn</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der ergebnis.html im Webbrowser:

maler

geboren *22.2.1865*
gestorben *10.3.1943*
geschlecht *männlich*
id *m39*
vorname *Otto*
nachname *Modersohn*
stilrichtung *Expressionismus*

4.25. Step 25: Katalogabfragen

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

query                   name=maler;part=/catalogue[personen,complete,"Modersohn-
Becker"]:vorname="Paula"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop target="ergebnis.html";overwrite=yes;
     package=htmldatei;form=html
```

Erläuterung:

- Bearbeitung des Katalogs personen. Im Katalog personen wird die Zeichenkette „Modersohn-Becker“ gesucht. Es wird durch complete vereinbart, dass genau nach dieser Zeichenkette bzw. diesem Terminus gesucht wird. Ferner soll das Element vorname noch die Zeichenkette „Paula“ enthalten.
- Bei der Ausgabe werden alle Maler angezeigt, die in ihrem Datensatz die Zeichenketten „Modersohn-Becker“ und „Paula“ enthalten.

Quellcode ergebnis.html:

```
<html>
<title>Bericht &uuml;ber einen Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>8.2.1876</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>20.11.1907</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>weiblich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m38</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Paula</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Modersohn-Becker</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>
```

Anzeige der ergebnis.html im Webbrowser:

maler

geboren *8.2.1876*
gestorben *20.11.1907*
geschlecht *weiblich*
id *m38*
vorname *Paula*
nachname *Modersohn-Becker*
stilrichtung *Expressionismus*

4.26. Step 26: Vorübungen für echte Server

```
query name=maler;part=/maler:nachname="Mackensen"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html
```

Erläuterung:

- Step 26 ist das erste Programm, das auf dem Webserver abläuft. Mit Hilfe der Secure Shell und ihrer Anwendungen (z.B. SSH, SFTP, SCP u. a.), wird vom lokalen Rechner zum jeweiligen Webserver eine Verbindung aufgebaut. Auf dem Webserver sollte man sich zunächst noch einmal die Basisstruktur eines logischen Kleioservers vergegenwärtigen, in der der Webserver in Software-, Datenbank- und die eigentlichen Programmverzeichnisse untergliedert wird. Um die Datenbank anfangs zu kompilieren, rufen Sie das Programm mit dem sh-Befehl *sh build.db > build.lis* aus dem Verzeichnis *server* auf. In der neu erstellten build.lis Datei befinden sich Informationen zum Aufbau der Datenbank(en). Danach geht es mit dem gewohnten Kleio Aufruf-Schema weiter. In diesem Fall: *kleiob step26.txt*.

- Durch den part Partmeter wird auf das Gruppenelement maler zugegriffen.

- Ferner soll das Element nachname ausgeschrieben werden. In dem Element soll die Zeichenkette „Mackensen“ gesucht werden.

- Besonderheit bei Step 26: Die Packages Gruppen und Elemente wurden bisher in den vorherigen Steps mehrfach definiert und ausgeführt. Hier verkürzt sich nun der write Befehl und die Einbaufunktion each [ ], die darauf zugreifen, ohne dass diese Packages noch einmal genau angeführt werden. Deshalb muss auf die Datei pack.1 hingewiesen werden. Das System greift auf die dort definierten Packages Gruppen und Elemente zu.

Datei pack.1:

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
      Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=groups

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=elemente
```

Darüber hinaus von Bedeutung ist die Datei readonly.on, die durch den Aufruf der build.db ebenfalls kompiliert wird. Diese Datei bewirkt, dass die Datenbank nur lesend verarbeitet werden soll.

Datei readonly.on:

```
environment usage=readonly
database name=maler;usage=readonly
database name=kleio;usage=readonly
```

- Bei der Ausgabe von Step 26 werden alle Künstler mit dem Nachnamen Mackensen ausgegeben.

Ausgabe step26.txt:

```
Kleio Version 10.0.0 --- 01.04.2005, 08:57:44
Historical Workstation Project / UNIX version
query name=maler;part=/maler:nachname="Mackensen"
write part=:each[=header,gruppen,
```

```
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html



Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
<html>
<title>Bericht &uuml;ber einen Maler</title>
</head>
<body>


<table>
<h3>maler</h3>
<tr><td width="20%"><b>geboren</b></td>
<td width="80%"><i>8.4.1866</i></td></tr>
<tr><td width="20%"><b>gestorben</b></td>
<td width="80%"><i>12.5.1953</i></td></tr>
<tr><td width="20%"><b>geschlecht</b></td>
<td width="80%"><i>m&auml;nnlich</i></td></tr>
<tr><td width="20%"><b>id</b></td>
<td width="80%"><i>m40</i></td></tr>
<tr><td width="20%"><b>vorname</b></td>
<td width="80%"><i>Fritz</i></td></tr>
<tr><td width="20%"><b>nachname</b></td>
<td width="80%"><i>Mackensen</i></td></tr>
<tr><td width="20%"><b>stilrichtung</b></td>
<td width="80%"><i>Expressionismus</i></td></tr></table>
</body>
</html>

Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.27. Step 27: Katalogisierte Prozeduren

```
item name=step26;usage=procedure;overwrite=yes
catalogue source="step26.txt";type=file
exit name=step26;
```

Erläuterung:

- In Step 27 wird erstmals eine Prozedur aufgeführt. Prozeduren vereinbaren Kleio Programme, die in Textdateien gespeichert sind und über das Web unter zugewiesenen Namen aktiviert werden können. Sie enthalten also eine Folge von Anweisungen, in denen verschiedene Programmelemente aufgerufen werden. Unter dem Begriff katalogisierte Prozeduren versteht man, dass die Prozeduren selbst unter dem Namen verzeichnet sind. Werden Prozeduren aktiviert, wird der Programmtext neu kompiliert.

- Die Prozedur wird mit dem Befehl item name=step26 in die logische Kleio Umwelt eingeführt. Ihr Typ wird durch den usage Parameter festgelegt. overwrite=yes bedeutet, dass der bestehende Inhalt durch einen neu erstellten Inhalt überschrieben werden darf. Als Quelle für den Katalog (= die Textdatei mit Anweisungen), wird die Datei step26.txt angeführt. type=file bedeutet, dass der Inhalt der Prozedur in einer ASCII-Datei gespeichert werden soll. Die Vereinbarung der Prozedur schließt mit exit name=step26 ab.

- Der Zugriff auf die Prozedur erfolgt über diesen Link: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurz/exec/step26

- Weitergehende Informationen zu Prozeduren sind in Abschnitt 5.1. verzeichnet.

4.28. Step 28: Einfache Suche mit Ersetzungsmarkern

```
query name=maler;part=/maler:nachname="@#1@"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html
```

Erläuterung:

- In Step 28 kommt es erstmals zum Einsatz von Ersetzungsmarkern. Wir greifen wie

gehabt (vgl. Step 26) durch den part Parameter auf das Element nachname zu. Anstatt beim Element nachname nach einer konkreten Zeichenkette zu suchen, wird nun über Ersetzungsmarker gesucht. Das hat den Vorteil, dass man per Aufruf der Adressliste des HTTP-Protokolls auf irgendeinen beliebigen Namen zugreifen kann. %22 ist dabei ein Escapecode für das Hochkommata. Dies wird so verschlüsselt, weil es problematisch wäre Sonderzeichen in URLs darzustellen.

- Ausführlichere Informationen zum Einsatz von Ersetzungsmarkern stehen in Abschnitt 5.1.

4.29. Step 29: Aufruf einer Prozedur

```
item name=step28;usage=procedure;overwrite=yes
catalogue source="step28.txt";type=file
exit name=step28;
```

Erläuterung:

- In Step 29 wird wiederum die Prozedur, die in „step28.txt“ vereinbart ist, aufgerufen. Step 29 ist parametrisiert durch einen Ersetzungsmarker und gibt alle Künstler der Datenbank „maler.xml“ aus. Der Zugriff darauf erfolgt über die HTTP-Adresse: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurs/exec/step28/%22%22. Falls man nach einem bestimmten Namen suchen möchte, beispielsweise nach Modersohn, würde man folgenden Link aufrufen: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KlKurs/exec/step28/%22Modersohn%22.

4.30. Step 30: Statische Liste auf dem Server

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
fields start="";
```

```
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</a>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
fields          start='<li><a          href="http://lehre.hki.uni-koeln.de/kurs-
cgi/kleioc/0010KlKurs/exec/step31/%22';
       limit='%22">';
exit name=verweis

query name=maler;part=/maler
index part=:package[:id,verweis];without=yes;
      part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
      type=list
stop target="../tag5.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- In Step 30 werden uns bereits bekannte Packages definiert. Nur im Package verweis wird mit dem Parameter start der Fields Direktive auf den Wert, der vor das Element id ausgeschrieben werden soll, in diesem Fall die HTTP-Adresse http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KlKurs/exec/step31/%22, verwiesen. Durch den Parameter limit der Fields Direktive wird der Wert, der nach dem Element id ausgeschrieben werden soll, hier %22“>, angeführt. Durch den index Befehl wird das ganze als Liste dargestellt. Die Ausgabe der Datenbankabfrage soll später in die HTML Datei tag5.html geschrieben werden.

4.31. Step 31: Query für Step 30

```
query name=maler;part=/maler:id="@#1@"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html
```

Erläuterung:

- In Step 31 findet sich die Aufgabenstellung für Step 30. So wird über die Gruppe maler auf das Element id zugegriffen. Das Element id wird ebenfalls über Ersetzungsmarker parametrisiert.

4.32. Step 32: Prozedur zu Step 31

```
item name=step31;usage=procedure;overwrite=yes
catalogue source="step31.txt";type=file
exit name=step31;
```

Erläuterung:

- Damit Step 31 aktiviert wird, muss die Prozedur, die in „step31.txt“ vereinbart ist, aufgerufen werden.
- Erst jetzt kann man die HTML-Seite tag5.html über den Link http://lehre.hki.uni-koeln.de/kleiokurs/tag5.html aufrufen.

Anzeige der tag5.html im Webbrowser:

```
• am Ende, Hans
• Barlach, Ernst
• Beckmann, Max
• Bock, Marie
• Braque, Georges
. . .
```

4.33. Step 33: Katalogausgaben modifizieren

```
query name=maler;part=:nachname
catalogue name=personen;part=:nachname;
          type=term;symbol=*
```

Erläuterung:

- In Step 33 wird der bereits existierende Katalog personen leicht modifiziert.
- Durch den part Parameter wird beim Query auf das Element nachname zugegriffen. Als Katalogname wird personen festgelegt und durch den part Parameter wird erneut nach dem Element nachname gesucht. type=term bedeutet, dass ein Eintrag als Ganzes in den Katalog eingehen soll, also keine Zerlegung in Einzelworte. symbol=* stellt eine Möglichkeit dar, um Wildcard-Symbole festzulegen. Der Parameter symbol beim catalogue Befehl erwartet ein einzelnes Zeichen bzw. Trunkierungssymbol, das als Wildcard verwendet werden soll. Als häufig genutztes Trunkierungssymbol in Katalogabfragen fungiert das Sternchen (*). Dies ist ein Platzhalter, der dann für beliebige nach dem Wortanfang stehende Zeichen gelten kann. Bsp.: Suche nach „Modersohn“ mit „Mod*“.

4.34. Step 34: Einfache Query

```
query name=maler;part=/catalogue[personen,complete,"Modersohn"]
write part=:each[]
```

Erläuterung:

- In Step 34 wird eine Datenbankabfrage definiert, wobei durch den part Parameter auf den Katalog personen zugegriffen wird. Im Katalog personen soll nach der Zeichenkette „Modersohn“ gesucht werden. Durch die Anweisung complete muss ganz genau nach diesem Wort gesucht werden.

Ausgabe ergebnis34.txt:

```
Kleio Version 10.0.0 --- 27.04.2005, 16:37:10
Historical Workstation Project / UNIX version
query name=maler;part=/catalogue[personen,complete,"Modersohn"]
write part=:each[]


Die Ausfuehrung der von Ihnen gestellten Aufgabe beginnt.
```

```
maler (39 = "m39")
          geboren      22.2.1865
          gestorben    10.3.1943
          geschlecht   männlich
          id           m39
          vorname      Otto
          nachname     Modersohn
          stilrichtung Expressionismus
Die Ausfuehrung der von Ihnen gestellten Aufgabe wurde beendet.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

4.35. Step 35: Einfache Abfrage

```
query name=maler;part=/catalogue[personen,complete,"@#1@"]
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html
```

Erläuterung:

- Durch den part Parameter wird auf den Katalog personen zugegriffen. Im Katalog personen soll nach der Zeichenkette gesucht werden, die durch Ersetzungsmarker definiert wird. Durch die Anweisung complete muss ganz genau nach diesem Wort gesucht werden.
- Nach dem write Befehl finden sich die gleichen Anweisungen wie in Step 31.

4.36. Step 36: Aufruf einer Prozedur

```
item name=step35;usage=procedure;overwrite=yes
catalogue source="step35.txt";type=file
exit name=step35;
```

Erläuterung:

- In Step 36 wird eine Prozedur aufgerufen. Damit Step 35 aktiviert wird, muss die Prozedur, die in „step35.txt“ vereinbart wurde, hier aufgerufen werden.

4.37. Step 37: Aufruf einer Abfragemaske

Quellcodes step37.html:

```
<html>
<head><title>Form Beispiel 1</title></head>
<body>
<form action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc" type="POST">
<input type="text" name="_2">
<input type="submit" value="Suchen">
<input type="hidden" name="!_kleioprot" value="0010KlKurs">
<input type="hidden" name="_1" value="step35">
</form>
</body>
```

Erläuterung:

- In Step 37 wird die erste einfache Suchmaske für die Datenbank „maler.xml“ generiert.

```
<form action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc" type="POST">
```

- Einleitendes Formularfeld mit Adresse des Webservers, auf dem die Anwendung läuft.

```
<input type="text" name="_2">
```

- Bezieht sich auf das 1. gezählte Element nach dem Schrägstrich.

```
<input type="submit" value="Suchen">
<input type="hidden" name="!_kleioprot" value="0010KlKurs">
```

- Der submit Button soll mit der Zeichenkette „Suchen“ beschriftet werden. Unter input type=hidden fallen alle zusätzlichen Argumente, die für den Benutzer nicht sichtbar sind.

```
<input type="hidden" name="_1" value="step35">
```

- Der Code bezieht sich auf die Stelle, von der die Prozedur aus aufgerufen wird. Das Argument _1 ist der Name der mit exec aufgerufenen Prozedur.

- Die einfache Suchmaske wird unter folgendem Link im Webbrowser aufgerufen: http://lehre.hki.uni-koeln.de/kleiokurs/server/step37.html

Anzeige im Webbrowser:

Abbildung 5: Anzeige Webbrowser

4.38. Step 38: Generierung einer Suchmaske

Datei pack.2:

```
note packages zu step38
note
item name=wrapsuche;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head><title>Ein ganz wichtiger Server</title>\n
     <body>\n
     <h1>Zugang zu einem Server</h1>';
       limit='</body></html>'
exit name=wrapsuche

item name=basicquery1;usage=package;overwrite=yes
```

```
fields    origin='<form    action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"
      type="POST">\n
   <input type="text" name="_2">\n
   <input type="submit" value="Suchen">\n
   <input type="hidden" name="!_kleioprot" value="0010KlKurs">\n
   <input type="hidden" name="_1" value="step35">\n
   </form>'
exit name=basicquery1

note ========================================================================

note packages zu step40

item name=psuche1;usage=package;overwrite=yes
fields    origin='<form    action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"
      method="POST"
    name="selector">\n
 <input type="hidden" name="_kleioprot" value="0010KlKurs">\n
 <input type="hidden" name="_1" value="step35">\n
 <input type="hidden" name="richtung" value="middle">\n';
       first='<select name="_2" size="4">';
       second='</select><br>\n';
       complete='<input type="submit" value="Zeige Personen" name="doexec">\n
 </form></center>';
       start="<option";
       write=' value=';       connect='>';
       limit="</option>";
exit name=psuche1
```

- Zu Beginn von Step 38 sollte man die Datei pack.2 kompilieren. Sie enthält verschiedene Packages, die zur Darstellung der Suchmaske von Bedeutung sind.

Eigentliche Query:

```
query name=maler;part=/nooperation[]
write part=:package[:form[""],basicquery1]
stop package=wrapsuche;form=html
```

Erläuterung:

- Die Einbaufunktion nooperation [ ] verhindert zunächst einmal eine Datenbankausgabe. Jedoch wird durch den write Befehl ein Package vereinbart, das

den Parameter form [ ] verwendet und das Package basicquery1 aufruft. Das Package wrapsuche hält das Grundgerüst einer HTML-Seite bereit.

4.39. Step 39: Aufruf der Prozedur von Step 38

```
item name=step38;usage=procedure;overwrite=yes
catalogue source="step38.txt";type=file
exit name=step38;
```

Erläuterung:

- In Step 39 wird die Prozedur step38 aufgerufen. Die Textdatei mit den Programmanweisungen befindet sich in step38.txt.
- Man kann die Prozedur unter diesem Link aufrufen: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KlKurs/exec/step38.

Anzeige im Webbrowser:

Abbildung 6: Anzeige im Webbrowser

4.40. Step 40: Text zur Generierung des Webinterfaces mit pick [ ]

```
query name=maler;part=/nooperation[]
write part=:pick[personen,"@=person@",@=richtung@,1000,psuche1],
     :package[:form[""],basicquery1]
stop package=wrapsuche;form=html
```

Erläuterung:

- Die Einbaufunktion nooperation [ ] erreicht, dass der Query Befehl die Voraussetzungen für die Bearbeitung des write Befehls herstellt, ohne das auf die Datenbank zunächst zugegriffen wird.

- Durch den write Befehl wird mit Hilfe der Einbaufunktion pick [ ] auf den Katalog personen zugegriffen. Durch die Einbaufunktion pick [ ] wird ein Stück Webinterface generiert.

- Diese Einbaufunktion erwartet fünf Argumente, von denen die ersten zwei vorgeschrieben sind.

  Das erste Argument ist der Katalogname.

  Das zweite Argument enthält meist einen Charakter String.[9]

  Beim dritten Argument innerhalb von pick [ ] kann jeweils eines der folgenden Schlüsselwörter angegeben werden: self, before, after oder middle.

  Das vierte Argument ist eine Nummer.

  Beim fünften Argument wird ein Packagename erwartet.

- Als Argumente innerhalb von pick [ ] wird in unserem Beispiel als erstes Argument auf den Katalog personen zugegriffen. Als zweites Argument wird der benannte Ersetzungsmarker @=person@ angeführt. Als drittes Argument wird ein zweiter benannter Ersetzungsmarker @=richtung@ eingeführt. Das vierte Argument ist eine Nummer, in dem Fall die Zahl 1000, die bedeutet, dass bis zu 1000 Werte dargestellt werden können. Das fünfte Argument ist ein Objektname, hier das Package psuche1, das in der Datei pack.2 näher definiert wurde.

[9] Weiterhin kann das zweite Argument aber ebenso einen Suchausdruck als Charakter String oder einen Pfad mit einem Element als letztem Bestandteil beinhalten.

- Wenn man versucht diese Query von der Kommandozeile zu kompilieren, erhält man jedoch eine Fehlermeldung dieser Art:

```
Kleio Version 10.0.0 --- 29.04.2005, 14:14:12
Historical Workstation Project / UNIX version
query name=maler;part=/nooperation[]
write part=:pick[personen,"@=person@",@=richtung@,1000,psuche1]
*****   Fehler:   Sie   haben   einen   unzulaessigen   Parameterwert   angegeben:
      "@=richtung@",1000,psuche1
Erlaubt: self, before, after, middle
,
     :package[:form[""],basicquery1]
stop package=wrapsuche;form=html


Die letzte Aufgabe wurde auf Grund der ausgewiesenen Fehler ignoriert.

Auf Wiedersehen bei Ihrer naechsten Kleio Sitzung!
```

Dies liegt daran, dass beim Kompilieren die Ersetzungsmarker nicht als solche behandelt, sondern genauso verwendet werden, wie sie dastehen. Um eine katalogisierte Prozedur zu testen, brauchen wir einen exec Aufruf. Dieser aktiviert die Prozedur genauso, als ob sie am Webserver aufgerufen wäre. Die entsprechende Datei heißt in unserem Beispiel call.40 (die Datei kann selbstverständlich jeden anderen beliebigen Namen führen).

Datei call.40:

```
exec name=step40;supply=person,"modersohn"
```

Die Datei erwartet immer einen, meist zwei Parameter. Unter name wird der Name der katalogisierten Prozedur, die ausgeführt werden soll, angeführt. Supply bedeutet, dass die angegebenen Werte für die Ersetzungsmarker benutzt werden, verwendet werden sollen. In dem Fall wird also für den benannten Ersetzungsmarker @=person@ der Wert „modersohn“ ersetzt.

Zum tieferen Verständnis folgendes Beispiel:

```
 exec name=step40;supply="yyyy","xxxx",name1,"qqqq",name2,"wwwww"

@#1@ == yyyy
@#2@ == xxxx
@=name1@ == qqqq
```

```
@=name2@ == wwww
```

Hierbei werden jeweils durch den supply Parameter gezählten und benannten Ersetzungsmarkern verschiedene Werte zugewiesen.

Aber:

```
exec name=step40;supply="yyyy","xxxx",name1,"qqqq","wwwww"
```

ist FALSCH. Wird einmal mit Namen begonnen, muss das bis zum Ende des Befehls fortgesetzt werden.

4.41. Step 41: Aufruf der Prozedur step40.txt

```
item name=step40;usage=procedure;overwrite=yes
catalogue source="step40.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=step40;
```

Erläuterung:

- In Step 41 wird wiederum eine Prozedur aufgerufen. In diesem Fall step40.
- Mit Hilfe des init Parameters kann man den benannten Ersetzungsmarkern „person“ und „richtung“ nun diverse Anfangswerte zuweisen. person bekommt das Wildcard Zeichen „m*“ zugewiesen. Durch diesen Buchstaben wird die Liste der Malernamen genauer positioniert, dabei wird sich am Buchstaben m orientiert, weil dieser in der Mitte des Alphabets steht. Der Ersetzungsmarker richtung bekommt den Wert middle zugeschrieben. Er zeigt an, in welche Richtung die Auswahlliste aufgebaut wird.
- Unter folgender Adresse kann man im Webbrowser den Link aufrufen: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurs/exec/step40.

4.42. Step 42: Errichtung eines ausgebauten Servers I

Ziel von Step 42 ist es, die zweite Datenbank „abstraktion.xml“ in die bereits bestehende einzubinden. Zunächst werden die Dateien „abstraktion.xml“, „abstraktion.mod“ und „abstraktion.dat“ in das Verzeichnis database hineinkopiert. Das Verzeichnis server erhält nun als neue Dateien die build2.db, proc.2, cat.1 und readonly2.on Datei. Im Folgenden werden jeweils die einzelnen Dateien genauer erläutert.

Sobald alle neuen Projektdateien installiert bzw. kompiliert sind, kann man sich über diesen Link eine einfache Suchmaske anzeigen lassen, in der jetzt der Inhalt von beiden Maler Datenbanken abgefragt werden kann:

http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KlKurs/exec/step40 .

Quellcode step40:

```
<html><head><title>Ein ganz wichtiger Server</title>
      <body>
      <h1>Zugang zu einem Server</h1>

   <form   action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"   method="POST"
name="selector">
     <input type="hidden" name="_kleioprot" value="0010KlKurs">
     <input type="hidden" name="_1" value="step35">
     <input type="hidden" name="richtung" value="middle">

   <select name="_2" size="4">
   <option>albers</option>
   <option>alechinsky</option>
   <option>altman</option>
   <option>am ende</option>
   <option>appel</option>
   <option>arp</option>
   <option>barlach</option>
   <option>beckmann</option>
   <option>bock</option>
   <option>braque</option>
   . . .
   <option>vordemberge-gildewart</option>
   <option>westhoff</option>
   <option>wouters</option>
   </select><br>
```

```
   <input type="submit" value="Zeige Personen" name="doexec">
     </form></center>

<form action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc" type="POST">
    <input type="text" name="_2">
    <input type="submit" value="Suchen">
    <input type="hidden" name="!_kleioprot" value="0010KlKurs">
    <input type="hidden" name="_1" value="step35">
    </form>
</body></html>
```

Anzeige von step40 im Webbrowser:

Abbildung 7: Suchmaske mit dem Inhalt zweier Datenbanken.

4.43. Step 43: Aufruf der build2.db

```
rm *.?_?
kleiob ../database/startup.env
kleiob ../database/maler.mod
kleiob ../database/maler.dat
kleiob ../database/abstraktion.mod
kleiob ../database/abstraktion.dat
########### Compilieren der erforderlichen Packages ####
kleiob pack.1
kleiob pack.2
########### Katalogisieren  der erforderlichen Prozeduren ####
kleiob proc.1
kleiob proc.2
########### Erzeugen der Kataloge
```

```
kleiob cat.1
########### Einmaliger Aufruf aller Programme die semi-statische Webseiten
########### generieren
kleiob write.personen
########### Umschalten aller DBs in den "read only" Status ####
kleiob readonly2.on
```

Erläuterung:

- Nachdem die neuen Projektdateien installiert wurden, sollte man zunächst die Batch-Datei build2.db ausführen. Sie dient vornehmlich dem einmaligen Aufruf aller Projektdateien, d.h. in ihr werden die jeweiligen Kommandos automatisch nacheinander ausgeführt.
- Weiterführende Informationen zur build.db Datei finden sich in Abschnitt 5.2.

4.44. Step 44: Datei readonly2.on

```
environment usage=readonly
database name=kleio;usage=readonly
database name=maler;usage=readonly
database name=abstraktion;usage=readonly
```

Erläuterung:

- Umschalten aller Datenbanken in den READ ONLY Status. Das bedeutet, dass die Datenbank durch einen schreibgeschützten Modus nur lesend verarbeitet werden soll. Es kann weder der Inhalt der Datenbank noch die Datenbank selbst geändert werden.

4.45. Step 45: Datei pack.2

Die schon aus Step 38 bekannte Datei pack.2 wird weiterhin verwendet. In dieser Datei werden spezielle Packages definiert, die zur Darstellung der einfachen Maskensuche benötigt werden.

4.46. Step 46: Datei proc.2

```
item name=step35;usage=procedure;overwrite=yes
catalogue source="step35.txt";type=file
exit name=step35;

item name=step40;usage=procedure;overwrite=yes
catalogue source="step40.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=step40;
```

Erläuterung:

- In der Datei proc.2 werden nun im Gegensatz zu den bisher verwendeten Dateien dieser Art, mehrere Prozeduren gleichzeitig aufgerufen. Durch den Aufruf der Prozeduren step35 und step40 werden die jeweiligen Textdateien aktiviert, die zusammen die Anzeige für die Suchmaske generieren.

4.47. Step 47: Datei cat.1

```
query name=maler;part=:nachname
catalogue name=personen;part=:nachname;
          type=term;symbol=*;
          suspend="dummy"
continue
query name=abstraktion;part=:nachname
catalogue name=personen;part=:nachname;
          complete="dummy"
```

Erläuterung:

- In der Datei cat.1 werden nacheinander verschiedene Kataloge der beiden Datenbanken aufgerufen.
- Die Parameter suspend, augment und complete beim Befehl catalogue steuern die Verbindung zwischen den einzelnen Katalogen in verschiedenen Datenbanken. Der Parameter suspend legt zunächst die Datei dummy an, in der die ausgewählten

Informationen (hier vom Element nachname) gespeichert werden. Durch den complete Parameter wird der Katalog erstellt und die dummy Datei zum Schluss wieder gelöscht. Weil wir in unseren Übungsdateien nur über zwei XML Datenbanken verfügen, kommt der Parameter augment, der darüber hinaus Kataloge aus weiteren XML Datenbanken einfließen lassen könnte, nicht zum Einsatz.

- Der continue Befehl trennt die beiden Aufgaben und ermöglicht so, dass die Katalogbefehle sich auf unterschiedliche Datenbanken beziehen.

4.48. Step 48: Errichtung eines ausgebauten Servers II / Datei pack.1

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
      Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"
exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen

item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente
```

Erläuterung:

- Weiterhin kann man noch die Datenbankausgabe mit verschiedenen Querys und Prozeduren variieren. Hier bei wird die Thematik durch vorangegangene Steps wiederholt, jedoch leicht modifiziert und variiert. So ist die Datei pack.1 bereits aus Step 26 bekannt.

4.49. Step 49: Datei proc.1

```
item name=result1;usage=procedure;overwrite=yes
catalogue source="result1.txt";type=file
exit name=result1;

item name=suche1;usage=procedure;overwrite=yes
catalogue source="suche1.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche1;
```

Erläuterung:

- In der Datei proc.1 werden die Prozeduren result1 und suche1 vereinbart. Den Ersetzungsmarkern @=person@ und @=richtung@, die in der Datei suche1.txt verwendet werden, werden in der Prozedur suche1 verschiedene Initialisierungswerte zugewiesen.

4.50. Step 50: Datei cat.1

```
query name=maler;part=:nachname
catalogue name=personen;part=:nachname;
          type=term;symbol=*;
          overwrite=yes;
          suspend="dummy"
continue
query name=abstraktion;part=:nachname
catalogue name=personen;part=:nachname;
          complete="dummy"
```

Erläuterung:

In die Datei cat.1 wurde beim Katalog personen noch ein overwrite=yes eingefügt. Ansonsten wie Step 47.

4.51. Step 51: Datei suche1.txt

```
query name=maler;part=/nooperation[]
write part=:pick[personen,"@=person@",@=richtung@,1000,psuche1],
     :package[:form[""],basicquery1]
stop package=wrapsuche;form=html
```

Erläuterung:

- Die Datei suche1.txt wird dann kompiliert, wenn sie aus dem Web aufgerufen wird, also durch die Datei proc.1.

4.52. Step 52: Datei result1.txt

```
query name=maler;part=/catalogue[personen,complete,"@#1@"]
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
stop package=htmldatei;form=html
```

Erläuterung:

- In der Datei result1.txt wird der Katalog personen abgefragt. Mit Hilfe eines gezählten Ersetzungsmarkers soll auf die verschiedenen Elemente zugegriffen werden. In Step 35 wurde bereits die gleiche Abfrage mit Hilfe des Katalogs personen und einem gezählten Ersetzungsmarker vorgenommen.

4.53. Step 53: Datei write.personen

```
item name=htmldatei;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmldatei

item name=registeranfang;usage=package;overwrite=yes
```

```
fields start="<li>";
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</li>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
fields start="<br>Die Angaben zu dieser Person finden sich in der Datei:
      personen/";
       limit=".html"
exit name=verweis
query name=maler;part=/maler
index part=:package[:id,verweis];without=yes;
      part=:package[:nachname,registeranfang];limit=", ";
      part=:package[:vorname,registerende];
      type=list
stop target="../tag7.html";overwrite=yes;
     package=htmldatei;form=html;
```

Erläuterung:

- Durch die Datei write.personen wird mit Hilfe von Packages eine listenartige Aufzählung der Künstlereinträge generiert. Diese einfache Listendarstellung findet sich auch schon in Step 30.

4.54. Step 54: Datei readonly2.on

```
environment usage=readonly
database name=kleio;usage=readonly
database name=maler;usage=readonly
database name=abstraktion;usage=readonly
```

Erläuterung:

- Umschalten aller Datenbanken in den READ ONLY Status. Das bedeutet, dass die Datenbank durch einen schreibgeschützten Modus nur lesend verarbeitet werden soll. Es kann weder der Inhalt der Datenbank noch die Datenbank selbst geändert werden.

- Unter folgendem Link kann man die beiden Malerdatenbanken auf dem Server http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurs/exec/suche1 ansehen.

4.55. Step 55: Weiterführende Suche / Datei suche2.txt

```
query name=maler;part=/nooperation[]
write part=:pick[personen,"@=person@",@=richtung@,5,psuche2],
     :package[:form[""],basicquery1]
stop package=wrapsuche;form=htmldatabase
```

Erläuterung:

- In der Datei suche2.txt wird durch das erste Argument der Einbaufunktion pick [ ] auf den Katalog personen zugegriffen. Mit den Ersetzungsmarkern @=person@ und @=richtung@ werden spezielle Zugriffe definiert. Das vierte Argument ist eine Zahlenangabe, in dem Fall die Zahl 5, die besagt, dass bis zu 5 Werte in der Klappliste der Suchmaske dargestellt werden können. Das fünfte Argument greift auf das Package psuche2 zu. Alles Weitere ist genauso wie in suche1.txt.
- Die Datei suche2.txt wird durch die Kompilierung der Prozedur proc.2 aktiviert.

4.56. Step 56: Datei pack.2

```
item name=htmldatei;usage=package;overwrite=yes
fields          start="<html>\n<title>Bericht          &uuml;ber          einen
      Maler</title>\n</head>\n<body>";
       limit="</body>\n</html>"

exit name=htmldatei

item name=Gruppen;usage=package;overwrite=yes
fields write="<h3>";
       connect="</h3>";
       origin='<table>';
       limit="</table>"
exit name=Gruppen
```

```
item name=Elemente;usage=package;overwrite=yes
fields write='<tr><td width="20%"><b>';
       start='</b></td> <td width="80%"><i>';
       limit="</i></td></tr>"
exit name=Elemente

note ##############################################################################

item name=htmlliste;usage=package;overwrite=yes
fields start="<html>\n<title>Liste der Maler</title>\n</head>\n<body><ul>";
       limit="</ul></body>\n</html>"
exit name=htmlliste

item name=registeranfang;usage=package;overwrite=yes
fields start="";
exit name=registeranfang

item name=registerende;usage=package;overwrite=yes
fields limit="</a>"
exit name=registerende

item name=verweis;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<li><a href="@=MyServer@suche1/%22';
       limit='%22">';
exit name=verweis

note ##############################################################################
item name=wrapsuche;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head><title>Ein ganz wichtiger Server</title>\n
     <script language="JavaScript" type="text/javascript">\n
       function getSet(richtung)\n
          {\n
          document.forms["selector"].elements["_1"].value="suche2";\n
          if (richtung==0)\n
             {\n
             document.forms["selector"].elements["person"].value=\n
                document.forms["selector"].elements["prevname"].value;\n
             document.forms["selector"].elements["richtung"].value="before";\n
             }\n
          else\n
             {\n
             document.forms["selector"].elements["person"].value=\n
                document.forms["selector"].elements["nextname"].value;\n
```

```
             document.forms["selector"].elements["richtung"].value="after";\n
             }\n
          }\n
     </script>\n
     </head>\n
     <body>\n
     <h1>Zugang zu einem Server <i>by Pages</i></h1>';
       limit='</body></html>'
exit name=wrapsuche

item name=basicquery1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<form action="@=MyCGI@" type="POST">\n
   <input type="text" name="_2">\n
   <input type="submit" value="Suchen">\n
   <input type="hidden" name="!_kleioprot" value="@=MyProtocol@">\n
   <input type="hidden" name="_1" value="result1">\n
   </form>'
exit name=basicquery1

note ##############################################################################

item name=psuche2;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<form action="@=MyCGI@" method="POST"
    name="selector">\n
 <input type="hidden" name="_kleioprot" value="@=MyProtocol@">\n
 <input type="hidden" name="_1" value="result1">\n
 <input type="hidden" name="richtung" value="middle">\n
 <input type="hidden" name="person" value="">\n';
       first='<br><select name="_2" size="4">';
       second='</select><br>\n';
       complete='<input type="submit" value="Zeige Personen" name="doexec">\n
 </form></center>';
       start="<option";
       write=' value='; connect='>';
       limit="</option>";
       before='<input type="submit" value="Voriger Name" name="doprev"
  onClick="return getSet(0);">\n
               <input type="hidden" name="prevname" value="';
       more='">\n';';
       after='<input type="submit" value="N&auml;chster Name" name="donext"
  onClick="return getSet(1);">\n
             <input type="hidden" name="nextname" value="';
exit name=psuche2
```

Erläuterung:

- Die Datei pack.2 enthält nun einige neue Packages. Bestimmte Packages wurden bereits in Step 38 eingeführt, wobei diese Packages nun um diverse Komponenten erweitert worden sind. Das Package wrapsuche beispielsweise enthält nun eine JavaScript-Funktion zum Selektieren der Ersetzungsmarker @=person@ und @=richtung@. Durch das Package psuche2 werden in der HTML-Seite die kompletten Fomulareigenschaften, inklusive der Vor- und Zurück-Buttons, generiert.

4.57. Step 57: Datei proc.2

```
item name=result1;usage=procedure;overwrite=yes
catalogue source="result1.txt";type=file
exit name=result1;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.2 werden die Prozeduren result1 und suche2 aufgerufen.
- Man kann die Prozedur unter diesem Link aufrufen: http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurs/exec/suche2.

Quellcode von suche2.html:

```
<html><head><title>Ein ganz wichtiger Server</title>
      <script language="JavaScript" type="text/javascript">
        function getSet(richtung)
           {
           document.forms["selector"].elements["_1"].value="suche2";
```

```
          if (richtung==0)
             {
             document.forms["selector"].elements["person"].value=
                document.forms["selector"].elements["prevname"].value;
             document.forms["selector"].elements["richtung"].value="before";
             }
          else
             {
             document.forms["selector"].elements["person"].value=
                document.forms["selector"].elements["nextname"].value;
             document.forms["selector"].elements["richtung"].value="after";
             }
          }
      </script>
      </head>
      <body>
      <h1>Zugang zu einem Server <i>by Pages</i></h1>

   <form   action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"   method="POST"
      name="selector">
     <input type="hidden" name="_kleioprot" value="0010KlKurs">
     <input type="hidden" name="_1" value="result1">
     <input type="hidden" name="richtung" value="middle">
     <input type="hidden" name="person" value="">

   <input  type="submit"  value="Voriger  Name"  name="doprev"     onClick="return
      getSet(0);">
                  <input type="hidden" name="prevname" value="larionow">

   <br><select name="_2" size="4">
   <option>lissitzky</option>
   <option>louis</option>
   <option SELECTED>macke</option>
   <option>mackensen</option>
   <option>majakowski</option>
   </select><br>

   <input     type="submit"     value="N&auml;chster     Name"     name="donext"
      onClick="return getSet(1);">
                  <input type="hidden" name="nextname" value="malevich">

   <input type="submit" value="Zeige Personen" name="doexec">
     </form></center>

<form action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc" type="POST">
```

```
    <input type="text" name="_2">
    <input type="submit" value="Suchen">
    <input type="hidden" name="!_kleioprot" value="0010KlKurs">
    <input type="hidden" name="_1" value="result1">
    </form>
</body></html>
```

Anzeige im Webbrowser:

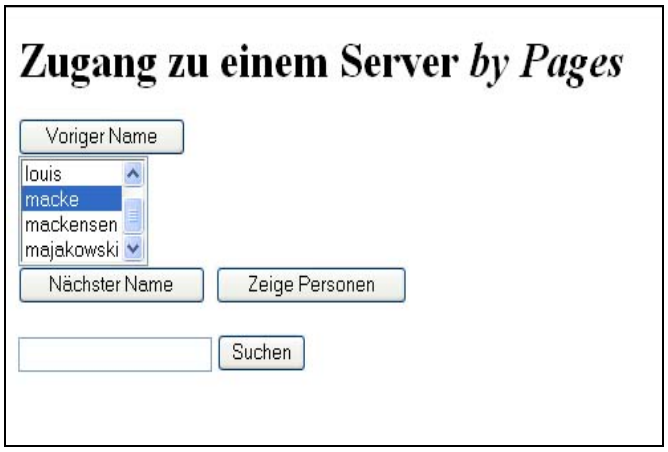


Abbildung 8: Erweiterte Suchfunktionen beim Webserver.

4.58. Step 58: Suchfunktionen zum „Blättern“ / Datei result2.txt

```
@+term@=#1@
@>term@supply@
@>pocketin@supply@
@>pocketout@supply@

query name=maler;part=/catalogue[personen,complete,"@#1@"]
prologue part=:package[:dump[pocket],lastpocket],
              :form['<form name="searchresult">']
write part=/pocket[pocket,@=pocketin@,@=pocketout@,mute]:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
epilogue part=:package[:dump[pocket],nextpocket]
stop package=htmldatei;form=html
```

Erläuterung:

- In der Datei result2.txt werden zunächst die benannten Ersetzungsmarker definiert.

```
@+term@=#1@
@>term@supply@
@>pocketin@supply@
@>pocketout@supply@
```

- So wird dem Ersetzungsmarker @+term@ der Inhalt des ersten nicht benannten, also gezählten Arguments zugewiesen. Bedeutung des Pluszeichens vor term: Falls der Name bzw. Marker term noch nicht existiert, so wird dieser eingeführt. Er erhält dann den Inhalt des 1. Ersetzungsmarkers.

- Die Ersetzungsmarker @>term@supply@, @>pocketin@supply@ und @>pocketout@supply@ besagen jeweils, dass in dem Fall damit in „ersetzungssensitive“ Packages (gekennzeichnet durch den Befehl: brackets sensitive=yes) Ersetzungsmarker exportiert werden können.

  -> Ersetzungssensitive Packages: Ein Package bei dem brackets sensitive=yes angegeben ist, arbeitet ähnlich wie eine katalogisierte Prozedur. Wenn das Package für die Ausführung vorbereitet wird, dann werden die Ersetzungsmarker zunächst ersetzt.

```
prologue part=:package[:dump[pocket],lastpocket],
```

- Bei dem Query werden die Befehle prologue und epilogue neu eingeführt. Durch diese Befehle werden die Vor- und Zurück-Buttons erzeugt. Bei prologue wird auf eine Packagevereinbarung zugegriffen. In diesem Package wird durch die Elementarfunktion dump [ ] der Inhalt der lokalen Variablen pocket ausgegeben und auf das Package lastpocket zugegriffen. prologue und epilogue wirken wie write Befehle. Aber prologue wird nur genau einmal ausgeführt und zwar vor dem ersten anderen Befehl. Analog dazu wird epilogue genau einmal ausgeführt und zwar nachdem alle anderen Befehle abgearbeitet sind.

```
write part=/pocket[pocket,@=pocketin@,@=pocketout@,mute]
```

- Die Einbaufunktion pocket [ ] verweist auf das erste Argument mit dem Namen pocket. Das Argument pocket stellt hier eine lokale Variable dar, also ein Name unter dem etwas gespeichert wird. Das zweite und dritte Argument sind die Ersetzungsmarker @=pocketin@ und @=pocketout@. Es sind Ersetzungsmarker die grundsätzlich durch Zahlen ersetzt werden. Nehmen wir 11-20 an. So werden alle auf die Einbaufunktion pocket folgenden Pfadbestandteile dann ausgeführt, wenn es sich um den 11. bis 20.

Aufruf von pocket handelt. Die als erste angegebene lokale Variable enthält jeweils eine Zahl, die angibt, um den wievielten Aufruf es sich handelt. Diese Zahl muss > oder = der lokalen Variable sein. Mute sorgt dafür, dass diagnostische Mitteilungen unterdrückt werden. Wenn beispielsweise 11-20 bearbeitet wird, dann wird beim 21. mitgeteilt, dass dieses nicht mehr bearbeitet wird.

```
epilogue part=:package[:dump[pocket],nextpocket]
```

Bei epilogue wird ein Package vereinbart, das wiederum mit dump den Inhalt der lokalen Variable ausgeben kann. Der Inhalt von pocket wird in nextpocket gespeichert.

4.59. Step 59: Datei pack.3

```
item name=htmldatei;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html>\n<title>Bericht &uuml;ber einen Maler</title>\n
     <script             language="JavaScript"             type="text/javascript"
      src="@=MyStatic@scripts/pocket.js">\n
     </script>\n
       </head>\n<body>';
       limit="</body>\n</html>"

exit name=htmldatei

. . .

item name=wrapsuche;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head><title>Ein ganz wichtiger Server</title>\n
     <script language="JavaScript" type="text/javascript">\n
       function getSet(richtung)\n
          {\n
          document.forms["selector"].elements["_1"].value="suche2";\n
          if (richtung==0)\n
             {\n
             document.forms["selector"].elements["person"].value=\n
                document.forms["selector"].elements["prevname"].value;\n
             document.forms["selector"].elements["richtung"].value="before";\n
             }\n
          else\n
             {\n
```

```
            document.forms["selector"].elements["person"].value=\n
               document.forms["selector"].elements["nextname"].value;\n
            document.forms["selector"].elements["richtung"].value="after";\n
            }\n
          }\n
     </script>\n
     </head>\n
     <body>\n
     <h1>Zugang zu einem Server <i>by Pages and Pockets</i></h1>';
       limit='</body></html>'
exit name=wrapsuche

. . .

item name=basicquery1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<form action="@=MyCGI@" type="POST">\n
   <input type="text" name="_2">\n
   <input type="submit" value="Suchen">\n
   <input type="hidden" name="!_kleioprot" value="@=MyProtocol@">\n
   <input type="hidden" name="_1" value="result2">\n
   </form>'
exit name=basicquery1

. . .

item name=psuche2;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<form action="@=MyCGI@" method="POST"
    name="selector">\n
 <input type="hidden" name="_kleioprot" value="@=MyProtocol@">\n
 <input type="hidden" name="_1" value="result2">\n
 <input type="hidden" name="richtung" value="middle">\n
 <input type="hidden" name="person" value="">\n';
       first='<br><select name="_2" size="4">';
       second='</select><br>\n';
       complete='<input type="submit" value="Zeige Personen" name="doexec">\n
 </form></center>';
       start="<option";
       write=' value='; connect='>';
       limit="</option>";
       before='<input type="submit" value="Voriger Name" name="doprev"
  onClick="return getSet(0);">\n
               <input type="hidden" name="prevname" value="';
       more='">\n';';
```

```
       after='<input type="submit" value="N&auml;chster Name" name="donext"
  onClick="return getSet(1);">\n
              <input type="hidden" name="nextname" value="';
exit name=psuche2

. . .

item name=lastpocket;usage=package;overwrite=yes
bracket sensitive=yes
fields start='<script language="JavaScript">\n
              writePreviousPock("@=term@",@=pocketin@,@=pocketout@,';
       limit=');\n</script>'
exit name=lastpocket

item name=nextpocket;usage=package;overwrite=yes
bracket sensitive=yes
fields start='<script language="JavaScript">\n
              writeNextPock("@=term@",@=pocketin@,@=pocketout@,';
       limit=');\n</script>'
exit name=nextpocket
```

Erläuterung:

- Die Datei pack.3 besitzt größtenteils die gleichen Packages wie die Datei pack.2 in Step 56. Deswegen wird an dieser Stelle auf die vollständige Aufführung von pack.3 verzichtet, so dass nur jeweils die Packages aufgezeigt werden, die neu hinzugekommen sind oder sich verändert haben.

- Im ersten Package htmldatei ist ein Verweis auf die JavaScript Funktion pocket.js im Verzeichnis scripts hinzugekommen.

- Beim Package wrapsuche wird nur eine neue Überschrift formuliert („Zugang zu einem Server *by Pages and Pockets*“).

- Im Package basicquery1 wird innerhalb der Formularfelder beim Verweis auf das erste, gezählte Argument ein neues value (result2) angegeben.

- Im Package psuche2 wird gleichfalls innerhalb der Formularfelder beim Verweis auf das erste, gezählte Argument ein neues value (result2) angegeben.

- Als neue Packages werden in der Datei pack.3 lastpocket und nextpocket vereinbart, die den Inhalt der Funktionen writePreviousPock und writeNextPock ausschreiben. Die

JavaScript Funktion writeNextPock wird beispielsweise mit diesem Parametern aufgerufen “b*“,1,6,45. Dabei wird b* dem Ersetzungsmarker term, 1 pocketin, 6 pocketout und 45 der Variable current zugeordnet.

4.60. Step 60: Datei pocket.js

```
function writePreviousPock(term,pocketin,pocketout,current)
{
if (pocketin>1)
   {
   if (pocketin>=6) pocketin=pocketin-5;
   else pocketin=1;
   document.writeln('<form    name="lastpocket"    action="http://lehre.hki.uni-
      koeln.de/kurs-cgi/kleioc">');
   document.writeln('<input           type="hidden"           name="!_kleioprot"
      value="0010KlKurs">');
   document.writeln('<input type="hidden" name="_1" value="result2">');
   document.writeln('<input name="_2" type="hidden" value="'+term+'">');
   document.writeln('<input            name="pocketin"            type="hidden"
      value="'+pocketin+'">');
   document.writeln('<input           name="pocketout"           type="hidden"
      value="'+(pocketin+5)+'">');
   document.writeln('<input      type="submit"      name="previousPocketButton"
      value="Vorherige Personen">');
   document.writeln('</form>');
   }
}

function writeNextPock(term,pocketin,pocketout,current)
{
if (current>pocketout)
   {
   document.writeln('<form    name="nextpocket"    action="http://lehre.hki.uni-
      koeln.de/kurs-cgi/kleioc">');
   document.writeln('<input           type="hidden"           name="!_kleioprot"
      value="0010KlKurs">');
   document.writeln('<input type="hidden" name="_1" value="result2">');
   document.writeln('<input name="_2" type="hidden" value="'+term+'">');
   document.writeln('<input            name="pocketin"            type="hidden"
      value="'+pocketout+'">');
```

```
   document.writeln('<input            name="pocketout"            type="hidden"
      value="'+(pocketout+5)+'">');
   document.writeln('<input type="submit" name="nextPocketButton" value="Weitere
      Personen">');
   document.writeln('</form>');
   }
}
```

Erläuterung:

- In der JavaScript Datei pocket.js befinden sich die beiden Funktionen writePreviousPock und writeNextPock.

4.61. Step 61: Datei proc.3

```
item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.3 werden die Prozeduren result2 und suche2 aufgerufen.
- Wenn result2 das erste Mal aufgerufen wird, ohne das pocketin und pocketout angegeben werden, dann wird 1 und 6 dafür verwendet.

Quellcode der erweiterten suche2.html:

```
<html><head><title>Ein ganz wichtiger Server</title>
      <script language="JavaScript" type="text/javascript">
```

```
      function getSet(richtung)
         {
         document.forms["selector"].elements["_1"].value="suche2";
         if (richtung==0)
            {
            document.forms["selector"].elements["person"].value=
               document.forms["selector"].elements["prevname"].value;
            document.forms["selector"].elements["richtung"].value="before";
            }
         else
            {
            document.forms["selector"].elements["person"].value=
               document.forms["selector"].elements["nextname"].value;
            document.forms["selector"].elements["richtung"].value="after";
            }
         }
   </script>
   </head>
   <body>
   <h1>Zugang zu einem Server <i>by Pages and Pockets</i></h1>

<form   action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"   method="POST"
   name="selector">
  <input type="hidden" name="_kleioprot" value="0010KlKurs">
  <input type="hidden" name="_1" value="result2">
  <input type="hidden" name="richtung" value="middle">
  <input type="hidden" name="person" value="">

<input  type="submit"  value="Voriger  Name"  name="doprev"     onClick="return
   getSet(0);">
               <input type="hidden" name="prevname" value="larionow">

<br><select name="_2" size="4">
<option>lissitzky</option>
<option>louis</option>
<option SELECTED>macke</option>
<option>mackensen</option>
<option>majakowski</option>
</select><br>

<input     type="submit"     value="N&auml;chster     Name"     name="donext"
   onClick="return getSet(1);">
              <input type="hidden" name="nextname" value="malevich">

<input type="submit" value="Zeige Personen" name="doexec">
```

```
     </form></center>

<form action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc" type="POST">
    <input type="text" name="_2">
    <input type="submit" value="Suchen">
    <input type="hidden" name="!_kleioprot" value="0010KlKurs">
    <input type="hidden" name="_1" value="result2">
    </form>
</body></html>
```

Anzeige im Webbrowser:

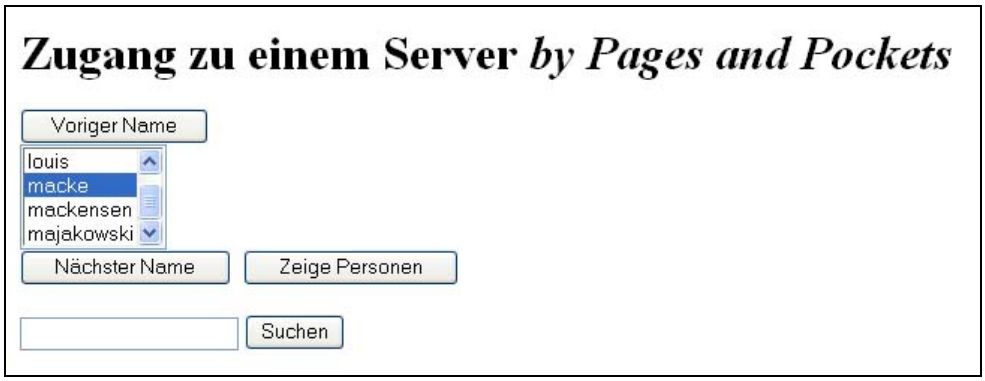


Abbildung 9: Suchmaske.

Abbildung 10: Beispielhafte Darstellung der „Blätterfunktion“ mit Hilfe von pocket.js.

4.62. Step 62: Maskensuche / Datei maske1.txt

```
query name=maler;part=/nooperation[]
write part=:mask[maske1,
      "Familienname",40,nachname,"",
      "Vorname",40,vorname,"",
      "Jahr",=double,
        "von", 4, von, "1800",
        "bis", 4, bis, "2000"]
stop form=html
```

Erläuterung:

- In der Datei maske1.txt wird durch die Einbaufunktion mask [ ] die Beschriftungen und die Textfelder einer einfachen Suchmaske festgelegt.
- Die Layoutkomponenten für die Suchmaskendarstellung der HTML-Seite werden in der Datei pack.4 bereitgestellt.

4.63. Step 63: Datei verarbeite1.txt

```
query name=maler;part=/nooperation[ ]
write part=:form["Maske 1 erfolgreich aufrufen."]
```

Erläuterung:

- In diesem Beispiel soll lediglich die einfachste Version der Maskensuche angezeigt werden, deshalb wird an dieser Stelle auf kompliziertere Querys verzichtet.

4.64. Step 64: Datei pack.4

```
item name=maske1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<html>\n
    <head>\n
   <title>Maskensuche</title>';
   before='</head>\n
    <body>\n
    <center><h1>Suchmaske</h1></center>\n
    <form method="POST" action="@=MyCgi@" name="maske1">\n
    <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyProtocol@">\n
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite1">\n
       <table border="0" width="100%" cellspacing="0">';
   after='</table>';
   start='<center>';
   limit='</center></form>';
       complete='</body>\n
      </html>\n';
   first='Suchen';
   second='Standard';
exit name=maske1

. . .
```

Erläuterung:

- In der Datei pack.4 wird als neues Package nur maske1 definiert, die in den vorherigen pack.* Dateien erzeugten Packages bleiben alle gleich. In maske1 wird mit Hilfe der

Fieldsparameter zunächst eine HTML-Seite ausgeschrieben. Mit origin und complete wird das Layout für das Gesamtmaterial definiert. Im Body werden die einzelnen <form> Tags des Formulars mit before festgelegt. Eine gewisse Positionierung des Ganzen wird durch die Fieldsparameter after, start und limit erzeugt, die eine blinde Tabelle um die Ausgabe legen und die Buttons mittig darstellen. Durch die Fieldsparameter first und second werden die Eingabetypen der Buttons - submit für den Absende-Button und reset für den Abbrechen-Button - mit den Werten „Suchen“ und „Standard“ beschriftet.

4.65. Step 65: Datei proc.4

```
item name=maske1;usage=procedure;overwrite=yes
catalogue source="maske1.txt";type=file
exit name=maske1

item name=verarbeite1;usage=procedure;overwrite=yes
catalogue source="verarbeite1.txt";type=file
exit name=verarbeite1

item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.4 werden neben den bisher bekannten Prozeduren result2 und suche2 noch die neuen maske1 und verarbeite1 aufgerufen.

4.66. Step 66: Fehlermeldungen integrieren / Datei verarbeite2.txt

```
@(Arg_Nachname@
error number=264;package=errorwrap;
      text="Der von Ihnen gewaehlte Suchbegriff wird nicht verwendet.<p>";
query name=maler;part=/catalogue[personen,complete,"@=Arg_nachname@"]
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
@)Arg_Nachname@
@[Arg_Nachname@
error number=0;package=errorwrap;
      text="Mindestens der Familienname muss angegeben sein.<p>";
      signal=yes
@]Arg_Nachname@
stop package=htmldatei;form=html
```

Erläuterung:

- In der Datei verarbeite2.txt werden speziell für die Katalogabfrage der Nachnamen Fehlermeldungen entwickelt. Der jeweiligen Fehlerart wird eine Nummer zugewiesen. So wird die abstrakte Fehlermeldung mit der Nummer 264 durch eine benutzerfreundlichere Sprache ersetzt.

Abbildung 11: Fehlerbericht für error number=264.

- signal=yes bewirkt, dass die Meldung sofort ausgegeben wird. Es ist keine der Standardfehlermeldungen, sondern falls der error Befehl vorgefunden wird, dann wird diese im errortext angegebene Fehlermeldung sofort generiert.
- Bei der Datei verarbeite2.txt testen die Ersetzungsmarker, ob Arg_Nachname vorhanden ist.

4.67. Step 67: Datei pack.5

```
item name=maske1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<html>\n
    <head>\n
   <title>Maskensuche</title>';
   before='</head>\n
    <body>\n
    <center><h1>Suchmaske</h1></center>\n
    <form method="POST" action="@=MyCgi@" name="maske1">\n
    <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyProtocol@">\n
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite2">\n
       <table border="0" width="100%" cellspacing="0">';
   after='</table>';
   start='<center>';
   limit='</center></form>';
       complete='</body>\n
      </html>\n';
   first='Suchen';
   second='Standard';
exit name=maske1

item name=errorwrap;usage=package;overwrite=yes
fields start='<html><head><title>Fehlerbericht</title></head>\n
  <body><font size=5">\n
  <center><H1>Fehlerbericht</H1>\n';
       limit='</center></font></body>\n</html>'
exit name=errorwrap

. . .
```

Erläuterung:

- In der Datei pack.5 wird beim Package maske1 im Formularfeld des ersten gezählten Arguments auf die Prozedur verarbeite2 zugegriffen.
- Als neues Package wird errorwrap definiert. In errorwrap wird das Grundgerüst für die Darstellung einer einfachen HTML-Seite bereitgestellt, die später die jeweiligen Fehlermeldungen anzeigen soll.

4.68. Step 68: Datei proc.5

```
item name=maske1;usage=procedure;overwrite=yes
catalogue source="maske1.txt";type=file
exit name=maske1

item name=verarbeite2;usage=procedure;overwrite=yes
catalogue source="verarbeite2.txt";type=file
exit name=verarbeite2

item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.5 werden zu den bisher bekannten noch die Prozedur verarbeite2.txt hinzugefügt.
- Die Suchmaske kann man unter dem Link http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc/0010KIkurs/exec/maske1 aufrufen.

Quellcode der maske1.html:

```
<html>
     <head>
    <title>Maskensuche</title></head>
     <body>
     <center><h1>Suchmaske</h1></center>
     <form  method="POST"  action="http://lehre.hki.uni-koeln.de/kurs-cgi/kleioc"
      name="maske1">
     <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="0010KlKurs">
     <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite2">
```

```
        <table border="0" width="100%" cellspacing="0">
<tr><td>Familienname</td><td><input      type="text"      size="40"      value=""
      name="Arg_nachname">
</td><td><input type="hidden" value="and" name="Con_nachname">
</td></tr><tr><td>Vorname</td><td><input    type="text"    size="40"    value=""
      name="Arg_vorname">
</td><td><input type="hidden" value="and" name="Con_vorname">
</td></tr><tr><td>Jahr</td><td>von<input   type="text"   size="4"   value="1800"
      name="Arg_von">
<input type="hidden" value="and" name="Con_von">
bis<input type="text" size="4" value="2000" name="Arg_bis">
</td><td> </td></tr></table>
<center><input type="submit" value="Suchen" name="Sub_Main"><input type="reset"
      value="Standard" name="Sub_Clean"></center></form></body>
       </html>
```

Anzeige im Webbrowser:

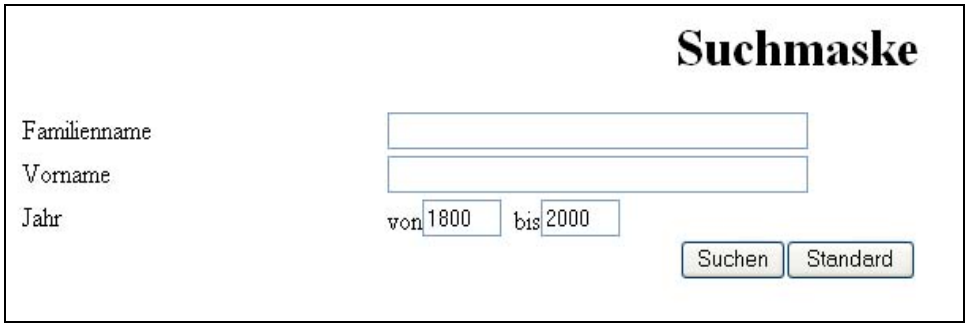


Abbildung 12: Maskensuche.

4.69. Step 69: Zusätzliche Ersetzungsmarker und Debugging-Modus / Datei verarbeite3.txt

```
@(Arg_Nachname@
error number=264;package=errorwrap;
      text="Der von Ihnen gewaehlte Suchbegriff wird nicht verwendet.<p>";
query
      name=maler;part=/catalogue[personen,complete,"@=Arg_nachname@"]@(Arg_Vorna
      me@:vorname="@=Arg_Vorname@"@)Arg_Vorname@
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
```

```
@)Arg_Nachname@
@[Arg_Nachname@
error number=0;package=errorwrap;
      text="Mindestens der Familienname muss angegeben sein.<p>";
      signal=yes
@]Arg_Nachname@
stop package=htmldatei;form=html
```

Erläuterung:

- In der Datei verarbeite3.txt werden zu den bisher bereits definierten Ersetzungsmarkern noch der Marker zum Element vorname festgelegt.

```
. . . @(Arg_Vorname@:vorname="@=Arg_Vorname@"@)Arg_Vorname@ . . .
```

- Bedingung: das Element :vorname wird nur verwendet, wenn der Ersetzungsmarker @=Arg_Vorname@ bekannt ist.

4.70. Step 70: Datei pack.6

```
item name=maske1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<html>\n
    <head>\n
   <title>Maskensuche</title>';
   before='</head>\n
    <body>\n
    <center><h1>Suchmaske</h1></center>\n
    <form method="POST" action="@=MyCGI@" name="maske1">\n
    <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyDebug@">\n
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite3">\n
       <table border="0" width="100%" cellspacing="0">';
   after='</table>';
   start='<center>';
   limit='</center></form>';
       complete='</body>\n
      </html>\n';
   first='Suchen';
   second='Standard';
exit name=maske1
```

. . .

Erläuterung:

- In der Datei pack.6 werden Veränderungen letztlich nur am ersten Package maske1 vorgenommen. So wird durch den Ersetzungsmarker @=MyDebug@ der Debugging-Modus gestartet und es werden dem Benutzer verschiedene Debugging-Informationen mitgeteilt. Das Debugging dient zunächst dazu, Programmierfehler in einer Anwendung zu finden und entfernen zu können, aber in unserem Fall soll man nur besser über die Vorgänge bei einer Datenbankabfrage informiert werden. Schritt für Schritt kann das Verhalten einer Anwendung somit analysiert werden. In der supply.ini wird der Marker MyDebug mit dem Wert "0111KlKurs" ersetzt. Wenn beispielsweise als Nachname die Zeichenkette „Hallo“ eingegeben würde, so wird im Debugging-Modus folgende Fehlermeldung generiert:

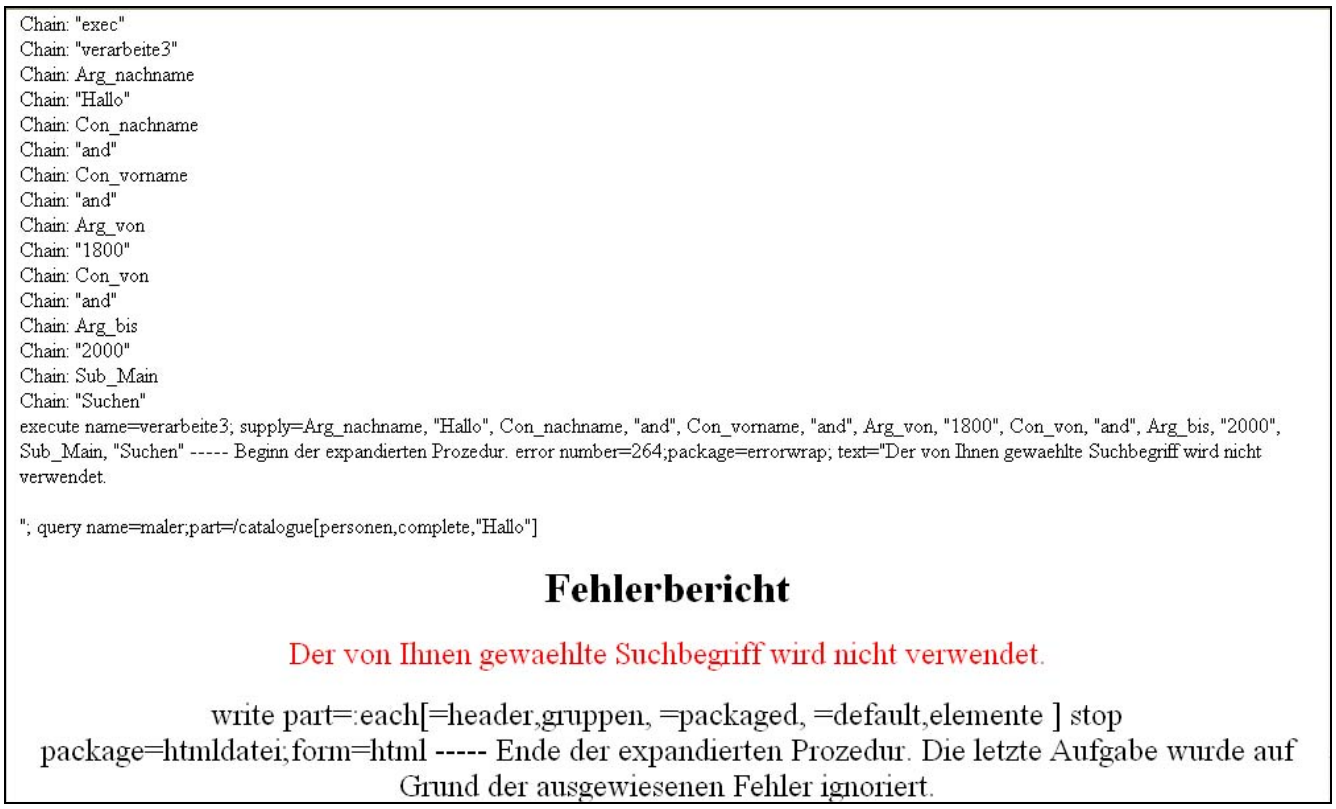


Abbildung 13: Debugging-Informationen.

- Dabei gilt: Der Ersetzungsmarker arg bezieht sich auf das Verhalten von mask [ ] und zeigt an, was übergeben wird. Der Ersetzungsmarker con zeigt an, wie etwas

verbunden wird. Chain bezieht sich auf den eigentlichen Debugging-Modus. Dabei spezifiziert chain, welche Argumente über das Web angekommen sind, nachdem sämtliche Zeichenersetzungen abgearbeitet sind.

- Die Prozedur testet der Reihe nach, ob die Ersetzungsmarker vorhanden sind, die den einzelnen Maskenzeilen entsprechen. Leere Formfelder werden per cgi nicht übertragen.

4.71. Step 71: Datei proc.6

```
item name=maske1;usage=procedure;overwrite=yes
catalogue source="maske1.txt";type=file
exit name=maske1

item name=verarbeite3;usage=procedure;overwrite=yes
catalogue source="verarbeite3.txt";type=file
exit name=verarbeite3

item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.6 wird als neue Prozedur nur verarbeite3 aufgerufen.
- An dem Quellcode der Suchmaske ändert sich nur diese Zeile:

```
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite3">\n
```

Deshalb wird die HTML-Seite der Maskensuche nicht noch einmal explizit angezeigt.

4.72. Step 72: Zeitangaben mit Ersetzungsmarkern / Datei verarbeite4.txt

```
@(Arg_Nachname@
error number=264;package=errorwrap;
      text="Der von Ihnen gewaehlte Suchbegriff wird nicht verwendet.<p>";
query
      name=maler;part=/catalogue[personen,complete,"@=Arg_nachname@"]@(Arg_Vorna
      me@:vorname="@=Arg_Vorname@"@)Arg_Vorname@
@_Arg_Von@1800@@+zeit@:yes@@]Arg_Von@
@_Arg_Bis@2005@@+zeit@:yes@@]Arg_Bis@
@(zeit@
switch part=:database[];name=selectDB
block name=maler;part="maler"
query part=:year[:geboren]=greater equal "@=Arg_Von@" and
           :year[:gestorben]=less equal "@=Arg_Bis@"
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]
exit name=maler;

block name=abstraktion;part="abstraktion"
query part=:year[:lebensdaten]=greater equal "@=Arg_Von@" and
           :year[:lebensdaten]=less equal "@=Arg_Bis@"
@)zeit@
write part=:each[=header,gruppen,
     =packaged,
     =default,elemente
     ]

@(zeit@
exit name=abstraktion
exit name=selectDB
@)zeit@

@)Arg_Nachname@

@[Arg_Nachname@
error number=0;package=errorwrap;
      text="Mindestens der Familienname muss angegeben sein.<p>";
      signal=yes
@]Arg_Nachname@
stop package=htmldatei;form=html
```

Erläuterung:

- In der Datei verarbeite4.txt werden Familien von Ersetzungsmarkern definiert („nachname“, „vorname“, „von“ und „bis“).

```
@_Arg_Von@1800@@+zeit@:yes@@]Arg_Von@
@_Arg_Bis@2005@@+zeit@:yes@@]Arg_Bis@
```

- Diese Konstruktion vereinfacht eine Abfrage, weil untersucht wird, ob eine zeitliche Einschränkung vorgenommen wird. @_Arg_Von@1800@@+zeit@:yes@@ -> Der Ersetzungsmarker zeit wird definiert und auf den Wert yes gesetzt, wenn der Ersetzungsmarker Arg_Von nicht den Wert 1800 hat. Damit wird geprüft, ob der Benutzer eine zeitliche Einschränkung vorgenommen hat.

- @+zeit@:yes@ -> Das Pluszeichen bewirkt, dass wenn der Ersetzungsmarker zeit noch nicht existiert, er eingeführt werden soll. Er erhält dann als Wert die Zeichenkette yes.

- Anders ausgedrückt, der Ersetzungsmarker Arg_Von wird angesprochen, wenn ein Zahlenwert = oder > 1800 abgefragt wird. Dasselbe vice versa zu @Arg_Bis@.

```
query part=:year[:gestorben]=greater equal "@=Arg_Von@" and
           :year[:gestorben]=less equal "@=Arg_Bis@"
```

- In der Datei verarbeite4.txt werden die Ersetzungsmarker Arg_Von und Arg_Bis definiert. Mit Hilfe der Vergleichsmodifikatoren greater than und less equal kann man beim Datentyp number noch genauer das Jahr definieren.

```
switch part=:database[];name=selectDB
      block name=maler;part="maler"
        . . . (Ausgabe für die 1. Situation.)
      exit name=maler;
      block name=abstraktion;part="abstraktion"
        . . . (Ausgabe für die 2. Situation.)
      exit name=abstraktion
exit name=selectDB
```

- Will man in mehreren Datenbanken nach gleichen Elementen suchen, so kann man mit dem switch Befehl diese Abfrage erleichtern. Durch den switch Befehl wird auf die Elementarfunktion :database [ ] zugegriffen. Jede switch Anweisung wird durch den Parameter name, in unserem Fall selectDB, identifiziert und durch die exit Direktive, mit demselben Namen, abgeschlossen. Innerhalb des switch Befehls befinden sich die

block Anweisungen, die auf die jeweiligen Datenbanken („maler.xml“ und „abstraktion.xml“) zugreifen. Der switch Befehl wird in Kleio ähnlich verwendet wie in anderen Programmiersprachen, der Unterschied ist hier nur das die Direktive block eine case Anweisung darstellt. Es wird das Wort case nicht verwendet, weil es Kleio intern schon für etwas anderes reserviert ist. In unserem Fall werden in den beiden Datenbanken die Jahresangaben verglichen und jeweils durch einen write Befehl ausgegeben.

- Der switch Befehl wird weiterhin von dem Ersetzungsmarker zeit geklammert.

```
@(zeit@
. . . (nur wenn zeit relevant ist, dann brauchen wir diesen Teil,
. . .
@)zeit@
. . .
@(zeit@
. . . ansonsten keine zeitliche Eingrenzung. Prinzip der bedingten Kompilation.)
@)zeit@
```

4.73. Step 73: Datei pack.7

```
item name=maske1;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<html>\n
    <head>\n
   <title>Maskensuche</title>';
   before='</head>\n
    <body>\n
    <center><h1>Suchmaske</h1></center>\n
    <form method="POST" action="@=MyCgi@" name="maske1">\n
    <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyProtocol@">\n
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite4">\n
       <table border="0" width="100%" cellspacing="0">';
   after='</table>';
   start='<center>';
   limit='</center></form>';
       complete='</body>\n
      </html>\n';
   first='Suchen';
   second='Standard';
exit name=maske1
```

```
. . .
```

Erläuterung:

- Die Datei pack.7 wird hier verkürzt aufgeführt, weil sich in ihr lediglich der Wert (verarbeite4) des ersten gezählten Arguments verändert hat.

4.74. Step 74: Datei proc.7

```
item name=maske1;usage=procedure;overwrite=yes
catalogue source="maske1.txt";type=file
exit name=maske1

item name=verarbeite4;usage=procedure;overwrite=yes
catalogue source="verarbeite4.txt";type=file
exit name=verarbeite4

item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- Auch in der Datei proc.7 wird lediglich die Prozedur verarbeite4 neu definiert. An der Darstellung der Maskensuche im Internet hat sich ebenso nichts geändert, weswegen hier auf die Abbildung verzichtet wird.

4.75. Step 75: Modifizierung der Suchmaske / Datei maske2.txt

```
query name=maler;part=/nooperation[]
write part=:mask[maske2,
      "Familienname",40,nachname,"",
         =catalogue,personen,
      "Vorname",40,vorname,"",
         =catalogue,vornamen,
      "Jahr",=double,
        "von", 4, von, "1800",
        "bis", 4, bis, "2005"]
stop form=html
```

Erläuterung:

- In der Datei maske2.txt kann man mit Hilfe von Index Buttons einen Suchbegriff auswählen.

4.76. Step 76: Datei index.txt

```
@+cat@=#1@
@+form@=#2@
@>form@supply@
@+arg@=#3@
@>arg@supply@
query name=maler;part=/nooperation[]
write part=:pick[@=cat@,"@=term@",@=dir@,1000,index]
stop form=html;package=indexwrap
```

Erläuterung:

- In der Datei index.txt werden mit Hilfe von Ersetzungsmarkern bestimmte Argumente festgelegt. So wird, falls der der Ersetzungsmarker cat noch nicht existiert dieser eingeführt (durch das +). Der Marker cat erhält den Inhalt des ersten gezählten (namenlosen) Ersetzungsmarkers zugewiesen. Der Ersetzungsmarker form, wird ebenso eingeführt, falls er noch nicht existiert, und erhält als Wert den Inhalt des zweiten gezählten Ersetzungsmarkers zugewiesen. Der Ersetzungsmarker form bekommt weiterhin durch das > Zeichen und supply mitgeteilt, dass er in

ersetzungssensitiven Packages Ersetzungsmarker exportieren kann. Der Ersetzungsmarker arg, wird ebenso eingeführt, falls er noch nicht existiert, und erhält als Wert den Inhalt des dritten gezählten Ersetzungsmarkers zugewiesen. Weiterhin kann arg in ersetzungssensitiven Packages Ersetzungsmarker exportieren.

- Durch den write Befehl wird mit der Einbaufunktion pick [ ] ein Webinterface generiert, indem auf einige Ersetzungsmarker zugegriffen wird.

4.77. Step 77: Datei pack.8

```
item name=maske2;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<html>\n
    <head>\n
   <title>Maskensuche</title>';
   before='</head>\n
    <body>\n
    <center><h1>Suchmaske</h1></center>\n
    <form method="POST" action="@=MyCgi@" name="maske2">\n
    <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyProtocol@">\n
    <INPUT TYPE="hidden" NAME="_1" VALUE="verarbeite4">\n
       <table border="0" width="100%" cellspacing="0">';
   after='</table>';
   start='<center>';
   limit='</center></form>';
       complete='</body>\n
      </html>\n';
   first='Suchen';
   second='Standard';
   more='Index';
   connect='@=MyServer@index';
   write='maske2';
   prologue='document.';
exit name=maske2

. . .

item name=indexwrap;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head><title>Verf&uuml;gbare Suchbegriffe</title></head>\n
    <script language="JavaScript">\n
```

```
    function installTerm()\n
       {\n
       opener.document.@=form@.elements["@=arg@"].value=\n

      document.catlist.execcat.options[document.catlist.execcat.selectedIndex].t
      ext;\n
       window.close();\n
       return true;\n
       }\n
  </script>\n
    <body>  <center><h1>Verf&uuml;gbare Suchbegriffe</h1></center>';
       limit='</body>\n
              </html>'
exit name=indexwrap

item name=index;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<center><FORM ACTION="@=MyCGI@" METHOD="POST" name="catlist">
      <INPUT TYPE="hidden" NAME="_1" VALUE="searchcat">
      <INPUT TYPE="hidden" NAME="!_kleioprot" VALUE="@=MyProtocol@">';
   first='<SELECT NAME="execcat" SIZE="12">';
   second='</SELECT><BR>';
   complete='<P><INPUT     TYPE="button"     VALUE="Verwenden"     NAME="doexec"
      onClick="return installTerm()">\n
      <INPUT TYPE="button" VALUE="Abbrechen" onClick="return window.close()">\n
      </FORM></center>';
   start="<OPTION";
   write=' VALUE=';
   connect='>';
   limit="</OPTION>";
convert prepare=prepforweb;
   usage=primary
convert prepare=prepforweb;
   usage=first
exit name=index

. . .
```

Erläuterung:

- In der Datei pack.8 wird das neue Package maske2 vereinbart. Als neue Parameter der Fields Direktive kommen more, durch den der Index-Button seine Beschriftung erhält, und connect, durch den der dynamische Index eingebunden wird bzw. die

Verbindung zur Kleio Prozedur hergestellt wird, hinzu.

- Im Package indexwrap wird der einfache Seitenwrapper für die HTML-Seitendarstellung der Popups bei den Indizes definiert.

```
    function installTerm()\n
       {\n
       opener.document.@=form@.elements["@=arg@"].value=\n
document.catlist.execcat.options[document.catlist.execcat.selectedIndex].text;\n
       window.close();\n
       return true;\n
       }\n
  </script>\n
```

Im Package wird ebenso die JavaScript Funktion installTerm () eingeführt, in der die Datei maske2 geöffnet und über elements auf das Argument eines Ersetzungsmarkers zugegriffen wird.

- Im Package index werden durch die Parameter der Direktive Fields die Formularangaben der HTML-Seite herausgeschrieben. Ebenso wird die Conversion prepforweb hier aufgerufen, die im nächsten Abschnitt erklärt wird. Der Gebrauch von usage=primary und usage=first wird am folgenden Beispiel veranschaulicht:

```
                    Annahme der Zeichenkette: Müller


   start="<OPTION";           <OPTION VALUE = x1 > x2</OPTION>
   write=' VALUE=';           (Durch die Fieldsparameter start, write,
   connect='>';                connect und limit wird diese Zeichenkette aus-
   limit="</OPTION>";         geschrieben. x1 zeigt Müller nach Anwendung
convert prepare=prepforweb;   von usage=primary und x2 zeigt Müller nach
   usage=primary              Anwendung von usage=first.)
convert prepare=prepforweb;
   usage=first
exit name=index
```

4.78. Step 78: Datei conv.1

```
item name=prepforweb;usage=conversion;overwrite=yes
substitute current='"';result="\&quot;"
substitute current='<';result="&lt;"
substitute current='>';result="&gt;"
exit name=prepforweb
```

Erläuterung:

- In der Datei conv.1 werden bestimmte Zeichenketten (Sonderzeichen) ersetzt durch benannte Sonderzeichen bzw. Unicode, um sie so besser im Internet darstellen zu können.
- Bei der conversion Vereinbarung wird mit Hilfe der substitution Direktive und dem Parameter current, der die austauschbare Zeichenkette angibt, und dem Parameter result, dessen angegebene Zeichenkette als Ersetzung dient, nach diversen Ersetzungsregeln Zeichenketten vereinheitlicht bzw. ersetzt.

4.79. Step 79: Datei cat.2

```
query name=maler;part=:vorname
catalogue name=vornamen;part=:vorname;
          type=term;symbol=*;
          suspend="dummy"
continue
query name=abstraktion;part=:vorname
catalogue name=vornamen;part=:vorname;
          complete="dummy"
```

Erläuterung:

- In der Datei cat.2 wird ein Katalog angelegt, der auf das Element vorname zugreifen soll.

4.80. Step 80: Datei proc.8

```
item name=maske2;usage=procedure;overwrite=yes
catalogue source="maske2.txt";type=file
exit name=maske2

item name=verarbeite4;usage=procedure;overwrite=yes
catalogue source="verarbeite4.txt";type=file
exit name=verarbeite4

item name=index;usage=procedure;overwrite=yes
catalogue source="index.txt";type=file
init name=term;text="m*"
init name=dir;text="middle"
exit name=index

item name=result2;usage=procedure;overwrite=yes
catalogue source="result2.txt";type=file
init name=pocketin;text="1"
init name=pocketout;text="6"
exit name=result2;

item name=suche2;usage=procedure;overwrite=yes
catalogue source="suche2.txt";type=file
init name=person;text="m*"
init name=richtung;text="middle"
exit name=suche2;
```

Erläuterung:

- In der Datei proc.8 wird die neue Prozedur maske2 aufgerufen.

5. Weiterführende logische Objekte

Im Folgenden werden die einzelnen Projektdateien der „Digitalen Stichwerkbibliothek“ des Forschungsarchivs für Antike Plastik und der Winckelmann-Edition vorgestellt.[10] Dieses Forschungsprojekt ist am Archäologischen Institut der Universität zu Köln angesiedelt und steht unter der Leitung von Prof. Dr. R. Förtsch. Die Historisch-Kulturwissenschaftliche Informationsverarbeitung ist für die technische Entwicklung des Servers verantwortlich. Als DBMS wird Kleio verwendet, unter einer Anwendung für die „Verteilte Digitale Inkunabelbibliothek“[11].

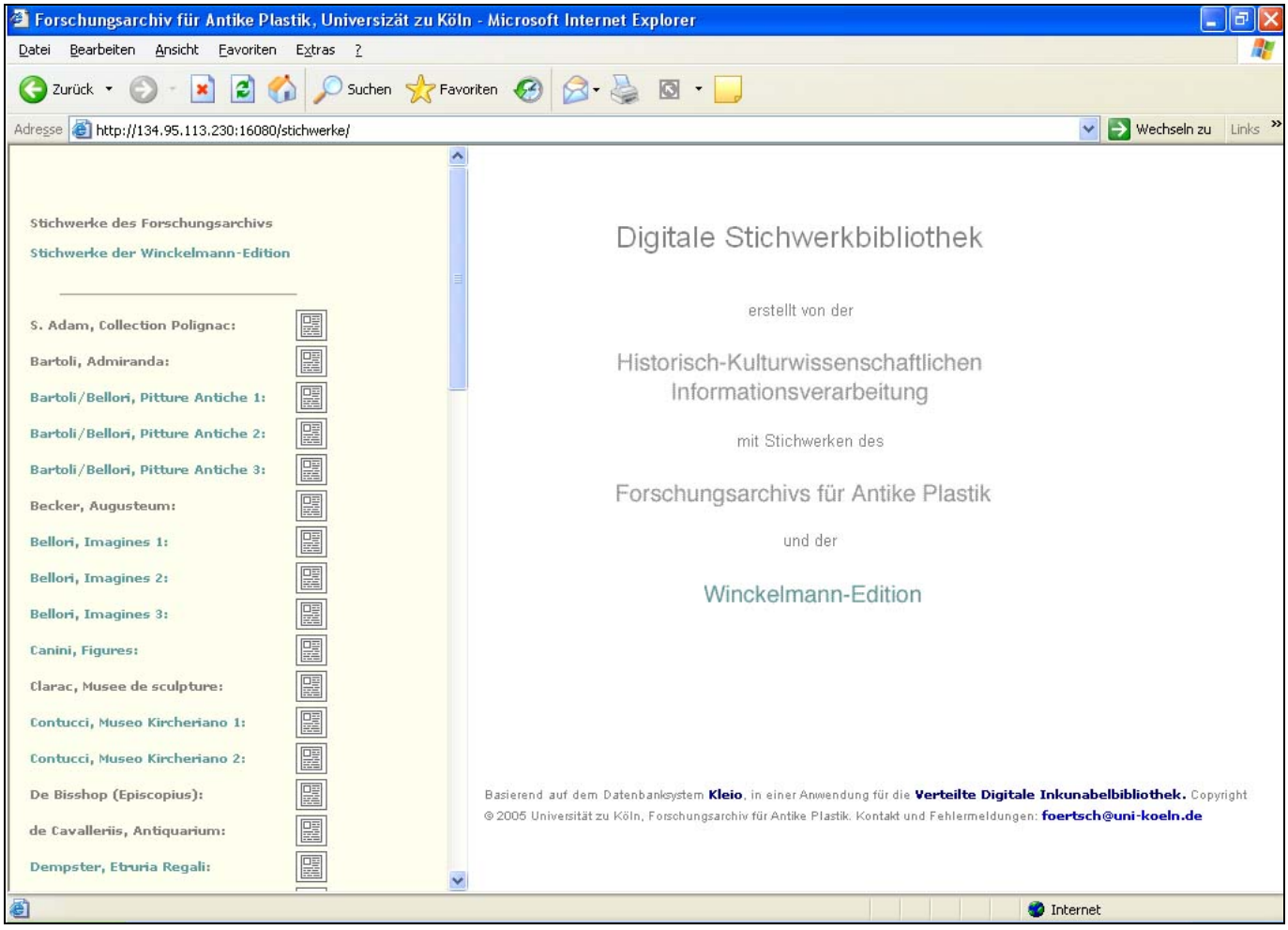


Abbildung 14 Stichwerke-Server; Stand: 15.02.2005.

Da dieser Webserver vor vergleichsweise kurzer Zeit erstmals im Internet präsentiert

[10] URL: http://134.95.113.230:16080/stichwerke. Den Entwicklungsserver der Stichwerke findet man unter der URL: http://134.95.113.230/stichtest/.
[11] URL: http://inkunabeln.ub.uni-koeln.de/vdib.

wurde, handelt es sich hierbei noch um ein relativ kleines Projekt. Durch diese Überschaubarkeit eignet es sich bestens, um vorab daran das Zusammenwirken der wichtigsten Kleio Datenbankkonzepte zu demonstrieren.

In dem Abschnitt sollen die drei wichtigsten logischen Objekte der Umwelt einer Kleio Datenbank vorgestellt werden: Kataloge, Prozeduren und Packages.

5.1 Einsatz von Ersetzungsmarkern

Bevor wir zum Thema der katalogisierten Prozeduren wechseln, muss man noch eingehender den Bereich der Ersetzungsmarker besprechen. Bei Ersetzungsmarkern handelt es sich um Variablen, die festlegen, in welcher Form verschiedene Programmteile verwendet werden sollen. Bevor nämlich eine Prozedur aufgerufen wird, sollen auf alle Fälle die Ersetzungsmarker schon definiert sein. Ersetzungsmarker sind somit eine Art Vorverarbeitungssprache auf der Ebene von Zeichenketten.

Ihren Einsatz finden sie hauptsächlich in durch Kleio generierten HTML-Seiten. Diese HTML-Seiten werden für viele Anwendungen so geschrieben, dass sie selbst wieder Aufrufe von anderen Prozeduren beinhalten. Zu diesem Zweck gibt es Mechanismen, um die an eine Prozedur übergebenen Ersetzungsmarker (selektiv), an von ihr über eine HTML-Seite aufgerufene andere, zu „vererben“.

In Kleio werden Ersetzungsmarker durch ein einleitendes und ein abschließendes @-Zeichen gekennzeichnet. Man unterscheidet dabei benannte und gezählte Ersetzungsmarker.

a) benannte Ersetzungsmarker:

```
@=AServer@
```

Es werden einige externe Dateien (z.B.: CSS, JavaScripts) und Icons für Buttons unter Zuhilfenahme dieser Ersetzungsmarker angesprochen. Eine Konstruktion wie @=AServer@ wird näher definiert in der supply.ini Datei. Wie wir von der Basisstruktur eines logischen Kleioservers (Abschnitt 3.4.) her wissen, befindet sich die supply.ini im cgi-bin Directory eines Web-Projekts. In diesem Verzeichnis befinden sich außer der

Software noch andere *.ini Dateien: bitmap.ini, datatype.ini, db.ini, explain.ini etc…(unter ini Dateien versteht man Konfigurationsdateien, in denen vornehmlich Programm- und Benutzereinstellungen einer Anwendung gespeichert werden).

Die Stichwerke supply.ini sieht wie folgt aus:

```
supply supply=AServer,"http://134.95.113.230/stichwerke/",
              ADynamic,"http://134.95.113.230/stich-cgi/kleioc/0010/exec/",
              ACgi,"http://134.95.113.230/stich-cgi/kleioc",
              ARepository,"http://134.95.113.230/stichwerke/images/";
       type=both
```

Durch den Parameter supply wird hier die Bezeichnung „AServer“ durch die Bezeichnung „http://134.95.113.230/stichwerke/“ ersetzt.

Oder der häufigere, andere Einsatzbereich ist der, des Wertes des name=“ … “ Attributs von <input> -Tags in HTML-Formularen. Wir erinnern uns

```
<input type=“hidden“ name=“Feld“ value=“Wert“>
```

führt dazu, dass dem Namen „Feld“ der Wert „Wert“ zugewiesen wird. Feld kann auch durch einen Ersetzungsmarker definiert werden, der demnach @=feld@ heißt.

b) gezählte Ersetzungsmarker:

Sie stammen hauptsächlich vom Wunsch her, möglichst kurze URLs verwenden zu können.

Regeln:

Es gibt bei Form fast immer benannte Ersetzungsmarker.

Um jedoch den URL Aufruf zu verkürzen, werden häufig gezählte Ersetzungsmarker verwendet.

```
In der Adresse
http://www.xyz/abc-cgi/kleioc/exec/beispiel/“Arg1“/“Arg2“
entspricht
der Prozedur beispiel der gezählte Ersetzungsmarker _1, für das Argument Arg1
nach dem Prozedurnamen gilt der Marker
_2 und für das Argument Arg2 nach dem Prozedurnamen der gezählte
Ersetzungsmarker _3.
Dabei ist jedoch zu beachten, dass aus _2 später in den Query Abfragen der
```

```
Ersetzungsmarker @=#1@ wird und aus _3 der Marker @=#2@.
```

Beispiel:

```
@+ref@=#1@
```

In unserem Fall ordnen wir mit der ersten Zeile an, dass dem Ersetzungsmarker ref der Inhalt des ersten gezählten Ersetzungsmarkers zugewiesen wird.

Allgemein:

-> Ersetzungsmarker entsprechen in HTML-Seiten je nach Aufruf von kleioc, entweder den durch Slashes getrennten Argumenten des nach …kleioc/exec/"…" noch folgenden Teils der URL, oder

-> dem Wert des name Attributs beim input Tag von HTML-Formularen. <input type="hidden" name="_2" value="Wert"> führt dazu, dass das erste (obwohl _2 dort steht) gezählte Argument den Wert „Wert" hat.

Für verschiedene Zwecke ist es in weiterer Folge nützlich mit namentlich bekannten Ersetzungsmarkern zu arbeiten. Bei einem Aufruf kann es aber ökonomischer sein, gezählte zu verwenden.

Ausnahmslos zählt zu den Vorteilen von Ersetzungsmarkern, dass durch ihren Einsatz der Server geschützter ist. Wenn überdies zwei Server gleichzeitig verwendet werden - ein Server für den normalen Betrieb und ein Entwicklungsserver - können Änderungen durch Ersetzungsmarker leichter von einem Server auf den anderen übertragen werden.

5.2. build.db Datei

Mit dem Shellscript build.db wird eine Datenbank kompiliert und neu aufgesetzt. Immer wenn man Änderungen an der Datenbank vorgenommen hat, kann man überprüfen, in welchem Programmelement eventuell Fehler aufgetreten sind. Man schreibt die Ergebnisse dieses Überprüfens der Datenbank in eine so genannte build.lis Datei. Die build.lis Datei gibt beispielsweise Auskunft über die verwendeten logischen Objekte

(Kataloge, Packages, Prozeduren), über die Anzahl der verwendeten Elemente, Gruppen, Dokumente und eventuelle Datenbankverbindungen zu anderen Datenbanken (Bridges). Empfehlenswert: Nach Kompilation Datei nach dem Wort Fehler durchsuchen, so dass nicht unangenehme Überraschungen durch Serverfehlfunktionen auftreten.

Beispiel der build.db Datei beim Stichwerke-Server (hier die Linux-Version).

```
export KLEIO=/Library/WebServer/stich-cgi
rm *.?_?
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.mod
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.dat
##############################
/Library/WebServer/stich-cgi/kleiob fauzk.img
/Library/WebServer/stich-cgi/kleiob cat.1
##############################
/Library/WebServer/stich-cgi/kleiob pack.1
/Library/WebServer/stich-cgi/kleiob proc.1
##############################
/Library/WebServer/stich-cgi/kleiob write.books
##############################
/Library/WebServer/stich-cgi/kleiob readonly.on
```

Die Windows-Version würde so aussehen:

```
export KLEIO=/Library/WebServer/stich-cgi
del *.?_?
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.mod
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.dat
##############################
/Library/WebServer/stich-cgi/kleiob fauzk.img
/Library/WebServer/stich-cgi/kleiob cat.1
##############################
/Library/WebServer/stich-cgi/kleiob pack.1
/Library/WebServer/stich-cgi/kleiob proc.1
##############################
/Library/WebServer/stich-cgi/kleiob write.books
##############################
/Library/WebServer/stich-cgi/kleiob readonly.on
```

Erläuterungen:

```
rm *.?_?
```

rm *.?_? führt dazu, dass vor dem Start der Datenbank erst einmal alle Dateien mit Underscore “_“ gelöscht werden. Sämtliche bestehende Kompilate werden gelöscht.

Dies funktioniert natürlich nur, wenn man darauf achtet selbst keine Dateien anzulegen, die eine Extension mit Underscores enthält. Alle von Kleio angelegten tun dies.

```
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.mod
/Library/WebServer/stich-cgi/kleiob ../database/fauzk.dat
```

Durch den Befehl kleiob werden zunächst einmal die wichtigen Projektdateien fauzk.mod und fauzk.dat kompiliert. (kleiob ist ein Batchjob, der am Server ausgeführt wird.)

```
/Library/WebServer/stich-cgi/kleiob fauzk.img
/Library/WebServer/stich-cgi/kleiob cat.1
```

Kompilieren des Katalogs cat.1. fauzk.img legt fest, wo die unter symbolischen Namen angesprochenen einzelnen Bilder auf dem server physisch zu finden sind.

```
/Library/WebServer/stich-cgi/kleiob pack.1
/Library/WebServer/stich-cgi/kleiob proc.1
```

Kompilieren der erforderlichen Packages und katalogisieren der erforderlichen Prozeduren.

```
/Library/WebServer/stich-cgi/kleiob write.books
```

Einmaliger Aufruf aller Programme die semi-statische Web-Seiten generieren.

```
/Library/WebServer/stich-cgi/kleiob readonly.on
```

Umschalten aller Datenbanken in den “read only” Status. Das bedeutet, dass die Datenbank nur lesend verarbeitet werden soll.

Die Datei readonly.on im Verzeichnis server:

```
environment usage=readonly
database name=kleio;usage=readonly
database name=fauzk;usage=readonly
```

5.3. Prozeduren

Allgemein definiert sind Prozeduren Scripte bzw. Textdateien, die eine Sammlung von Anweisungen enthalten, die von Kleio als eine Art von Befehlsfolge interpretiert werden. Ihren hauptsächlichen Einsatz finden Prozeduren in der Steuerung von bestimmten Abläufen oder Komponenten.

Durch eine Prozedur können verschiedene andere Programmelemente aufgerufen und verarbeitet zu werden. Bei Kleio dienen diese Hilfsmittel in erster Linie dazu, um Kataloge zu aktivieren, deshalb nennt man sie auch „katalogisierte Prozeduren“ und um Query-Aufgaben einer Abfrage zu erstellen.

Unter einem bestimmten Namen werden diese Objekte zunächst einmal in die Umwelt des Kleio Programms eingeführt, sie werden dann unter diesem Namen aufgerufen und ausgeführt. Die Datei proc.1 definiert und aktiviert somit nur die einzelnen Prozeduren. Werden Prozeduren aktiviert, wird der Programmtext also neu „kompiliert“.

5.3.1 Datei proc.1

Der Aufruf verschiedener Prozeduren erfolgt in der Datei proc.1:

```
item name=pagesma;usage=procedure;overwrite=yes
catalogue source="pagesma.txt";type=file
exit name=pagesma

item name=pagemed;usage=procedure;overwrite=yes
catalogue source="pagemed.txt";type=file
exit name=pagemed

item name=pagepro;usage=procedure;overwrite=yes
catalogue source="pagepro.txt";type=file
exit name=pagepro

item name=pagebig;usage=procedure;overwrite=yes
catalogue source="pagebig.txt";type=file
exit name=pagebig
```

In der Datei proc.1 befinden sich vier verschiedene Prozeduren, wobei sich die Bezeichnungen der Prozeduren auf die jeweiligen Auflösungen der Digitalisate beziehen.

Einzelnes Beispiel:

```
item name=pagesma;usage=procedure;overwrite=yes
catalogue source="pagesma.txt";type=file
exit name=pagesma
```

Also bezieht sich pagesma auf die kleine Überblicksdarstellung der Digitalisate bei der Auflösung fit. Durch item name= wird die Prozedur mit dem Namen pagesma belegt und in die logische Umwelt der Kleio Datenbank eingeführt. Durch den Parameter usage wird die Klasse des logischen Objekts, in dem Fall procedure, ausgewählt. overwrite=yes bewirkt, dass der bestehende Inhalt stets durch einen neu erstellten Inhalt (bzw. durch neue Abfragen) überschrieben werden darf. Dies muss explizit durch overwrite=yes definiert werden, ansonsten würde das System sich sicherheitshalber weigern den bisherigen Inhalt zu löschen.

catalogue source=“pagesma.txt“ gibt den Pfad an, wo man den dazugehörigen Quellcode zur Prozedur pagesma finden kann. type=file gibt mehr Informationen über die Datei pagesma.txt. In dem Fall handelt es sich um eine Datei, die im ASCII-Format vorliegt.

Durch exit name=pagesma wird die katalogisierte Prozedur beendet.

Die anderen Beispiele:

```
item name=pagemed;usage=procedure;overwrite=yes
```

Die Prozedur pagemed verweist auf die Arbeitsqualität von film,

```
item name=pagepro;usage=procedure;overwrite=yes
```

pagepro auf die verbesserte Qualität der indiv Auflösung und

```
item name=pagebig;usage=procedure;overwrite=yes
```

pagebig auf die maximale Auflösung von max.

Die Dateien, auf die sich die Prozeduren beziehen, haben die formale Struktur „Dateiname“ plus der Extension *.txt. Der Inhalt des Scripts wird im ASCII-Format gespeichert. Diese Textdateien enthalten den eigentlichen Quellcode.

Was steht nun in diesen Textdateien, auf die die Prozedur proc.1 verweist?

5.3.2 Datei pagesma.txt

Als Beispiel schauen wir uns nur den Quellcode der Datei pagesma.txt an, die sich auf die kleinste Auflösung der Digitalisate unter ca. 360 x 480 Pixel bezieht:

```
01 @+ref@=#1@
02 query name=fauzk;part=/catalogue[pages,complete,"@=ref@"]
03 write part=:form['<center><form action="@=ACgi@" method="post">'],
04        :copy[:form["@=ref@"],thispage,failure,relaxed],
05        :form['<input type="hidden" name="!_kleioprot" value="0010">
06              <input type="hidden" name="_1" value="pagesma">
07              <select name="_2" size="1"
08              onChange="this.form.submit()">'],
09           /query[]/root[2]:total[=invisible,=nounpackaged,=noheader,
10                                 =groups,image,
11                                 =noheader,=ordered,=packaged,
12                                   :extref,realid,
13                                   :nativeno,showid],
14        :form['</select></form></center><center>'],
15
:package[:neighbour[:form["@=ref@",image,fit],first,group],firstsma],
16
:always[:package[:neighbour[:form["@=ref@",image,fit],previous,group],prevsma],'
<img src="@=AServer@buttons/VorherigeSrg.gif">'],
17
:always[:package[:neighbour[:form["@=ref@",image,fit],next,group],nextsma],'<img
src="@=AServer@buttons/NaechsteSrg.gif">'],
18        :package[:neighbour[:form["@=ref@",image,fit],last,group],lastsma],
19                                    :form['<img     alt="Diese     Aufl&ouml;sung"
src="@=AServer@buttons/AugeSg.gif">'],
20           :package[:form["@=ref@"],refmedpage],
21           :package[:form["@=ref@"],refpropage],
22           :package[:form["@=ref@"],refbigpage],
23           :form['</center><p><center>'],
24           :package[:form["@=ref@"],smapage]
25 stop form=html;package=pagewrap
```

Erklärung zu einzelnen Zeilen:

```
01 @+ref@=#1@
```

Zeile 01 werden zu Beginn durch das @-Zeichen die Ersetzungsmarker eingeleitet. In unserem Fall ordnen wir mit der ersten Zeile an, dass dem Ersetzungsmarker @+ref@ der

Inhalt des ersten gezählten („namenlosen“) Ersetzungsmarkers zugewiesen wird.

```
02 query name=fauzk;part=/catalogue[pages,complete,"@=ref@"]
```

Mit query name=fauzk wird eine Abfrage für die Datenbank mit dem Namen fauzk erstellt. Es wird auf den Katalog pages verwiesen, indem gesucht wird.

```
03 write part=:form['<center><form action="@=ACgi@" method="post">'],
```

Zeile 03 ist für die HTML-Anzeige einer dynamischen Web-Seite wichtig. Sie sorgt dafür, dass Teile eines CGI-Formulars in der HTML-Seite ausgeschrieben werden.

Möglich wird dies, durch die Kleio-Einbaufunktion :form[ ], die Zeichenketten ausschreibt. Das erste einfache Anführungszeichen leitet eine Zeichenkette in Kleio ein und das letzte einfache Anführungszeichen schließt diese. Zeilenumbrüche werden durch die mit dem Backslash (\) maskierte newline (\n) erstellt.

In diesem Fall wird das angesprochene Element durch das -Tag zunächst in die Mitte gerückt. Als nächstes werden Tags zur Formulargestaltung vereinbart. In Zeile 03 wird erstmals das <form>-Tag erwähnt. <form> leitet ein Formular von einem HTML-Dokument ein. Innerhalb des Einleitungstags erscheinen noch zwei andere Angaben: "method=..." und "action=...". In "method=“ wird die Methode angeführt, mit der die Formulardaten per CGI-Scripts an den Server-Rechner übermittelt werden. Hier kommen generell die Angaben method=post oder method=get in Frage.
method="POST": Bei POST werden die Daten des Formulars als eigenständiger Datenstrom an den Server gesandt. Der Server stellt die Daten dem CGI-Programm über die Standardeingabe "stdin" bereit, und das Script muss die Daten behandeln wie eine Benutzereingabe, die auf der Kommandozeile gemacht wurde. In dem Fall wird kein EndOfFile-Signal (EOF) gesendet, so dass das CGI-Programm die Standard-Umgebungsvariable CONTENT_LENGTH auslesen muss, um die Länge der übermittelten Daten und damit deren Ende zu ermitteln.
method="GET": Bei GET werden die Daten des ausgefüllten Formulars an die URL angehängt und dann vom Server in der Standard-Umgebungsvariablen QUERY_STRING gespeichert. Das CGI-Programm muss dann den Inhalt dieser Umgebungsvariablen auslesen und verarbeiten.
In Zeile 03 wird dem <form>-Tag das Attribut action beigestellt. In "action=" wird die Internetadresse angegeben, zu der das ausgefüllte Formular geschickt werden soll. In unserem Fall ist die Angabe @=ACgi@ wiederum ein Kleio Ersetzungsmarker, der in der supply.ini näher aufgeschlüsselt wird.

```
04 :copy[:form["@=ref@"],thispage,failure,relaxed],
```

In Zeile 04 wird durch copy der Inhalt von :form[ ] in die „lokale Variable“ thispage kopiert. Das hat den Vorteil, dass sich alle späteren Kommandos darauf verlassen können, dass diese lokale Variable ab dieser Stelle den richtigen Wert enthält. Würden wir dies so stehen lassen, würde der Inhalt von :form[ ] an dieser Stelle allerdings auch in die Ergebnisdatei übertragen, weshalb failure angehängt wird. Nachdem Erfolglosigkeit beim Versuch einen Pfad abzuarbeiten aber unter Umständen für die weitere Bearbeitung Konsequenzen nach sich zieht, folgt danach relaxed. failure und relaxed sorgen gemeinsam also dafür, dass hier kopiert wird, ohne dass die Ergebnisdatei verändert wird.

```
05 :form['<input type="hidden" name="!_kleioprot" value="0010">
```

In Zeile 05 wird das <input>-Tag eingeführt, was für die Eingabemöglichkeiten der Formulardaten steht. Normalerweise werden dem Attribut type Werte wie „button“, „checkbox“, „password“ etc... zugewiesen. In unserem Fall wurde type auf „hidden“ gesetzt. Das ist ein Beispiel für versteckte Felder bzw. Steuerelemente. Versteckte Felder sind z.B. wichtig, wenn man mit mehrstufigen CGI-Skripten arbeitet (beispielsweise wenn ein erstes Skript Ausgabedaten ausgibt, die als Eingabedaten eines zweiten Skriptes dienen sollen). Durch das Attribut „hidden“ wird also das Element ausgeblendet, es enthält zusätzliche Argumente, die der Benutzer nicht sieht. Das Attribut von name legt den Namen des Elements fest und value definiert einen Vorgabewert. In unserem Fall besitzt value den Wert 0010. Dieser Wert gibt Auskunft darüber in welchen Direktiven wir die Datenbank suchen sollen. Dieser Protokollcode besteht meist aus einer Kette aus 4 Ziffern (0010), oftmals gefolgt von der Environmentwurzel. Dort erfährt man, wo sich die Serverdatenbank befindet. Weitere Erklärungen zu Protokollcodes finden sich im Anhang.

```
06                  <input type="hidden" name="_1" value="pagesma">
```

In Zeile 06 wurde das Attribut type ebenfalls auf „hidden“ gesetzt.
Das name Attribute hat hier den Wert "_1". Mit Underscore 1 wird der Name der Prozedur angesprochen. Underscore 2 ist das erste gezählte Element nach dem Schrägstrich. value ist in diesem Fall die Prozedur pagesma.

```
07                  <select name="_2" size="1"
```

In Zeile 07 wird Underscore 2 ausgewählt und size besagt, dass jeweils nur eine Zeile der durch select erzeugten Auswahlliste sichtbar ist.

```
08                  onChange="this.form.submit()">'],
```

Durch Zeile 08 wird angeordnet, dass das Formular an den Server abgeschickt wird,

sobald der Benutzer in der Auswahlliste einen neuen Wert ausgewählt hat.

```
09        /query[]/root[2]:total[=invisible,=nounpackaged,=noheader,
10                               =groups,image,
11                               =noheader,=ordered,=packaged,
12                                 :extref,realid,
13                                 :nativeno,showid],
```

In Zeile 09 beginnt die eigentliche Ausgabe der Daten zu den einzelnen Seiten. Mit /query[] wird zunächst dafür gesorgt, dass wir auf alle Fälle genau zu jener Stelle der Datenbank kommen, an die uns der query Befehl in Zeile 02 positioniert hatte. (Das ist im aktuellen Kontext relativ überflüssig, aber gute Praxis, damit wir nicht überrascht werden, wenn wir später an den Anfang des write Befehls zusätzliche Pfade einfügen.) /root[0] würde uns zur Wurzel des Dokuments bringen, aus dem wir mit /catalogue[....] ausgesucht haben. /root[2] positioniert uns zwei Ebenen darunter, aber jedenfalls innerhalb des Pfades zwischen der Dokumentenebene und dem von uns durch /query[] angesteuerten Punkt innerhalb des Dokuments. Die Einbaufunktion :total[ ] sorgt jetzt dafür, dass nicht nur an dieser Stelle vorhandene Elemente ausgeben werden, sondern ab dem erreichten Punkt die Hierarchie nach unten durchlaufen und eine Teilmenge der angetroffenen Elemente - auch solche aus in der Hierarchie weiter unten stehenden Gruppen – ausgeben werden.

Für diesen "Durchgang durch die Hierarchie" werden in Zeile 09 zunächst allgemeine Regeln festgelegt, die mittels durch Ist-Gleich-Zeichen beginnender Schlüsselwörter ausgewählt werden.

=noheader legt fest, dass der Beginn einer neuen Gruppe normalerweise *nicht* in der Ausgabe vermerkt wird.

=nounpackaged legt fest, dass Elemente, für die keine Ausgabeanweisungen (eigentlich: "Verpackungsanweisungen") vorliegen, *nicht* ausgegeben werden.

=invisible bestimmt, dass auch in Gruppen, die selbst nicht bearbeitet werden sollen, nach bearbeitbaren Gruppen gesucht wird.

Diese Vorgaben gelten für alle Gruppen, für die keine anderen Anweisungen getroffen wurden – das sind *alle,* außer /image.

=groups in Zeile 10 teilt mit, dass die folgenden Anweisungen für eben diese Gruppe gelten.

Zeile 11 beginnt mit der Festlegung allgemeiner Regeln für die Ausgabe dieser Gruppe: =noheader legt fest, dass auch die Tatsache, dass diese Gruppe beginnt, nicht in der Ausgabe vermerkt wird.

=ordered und =packaged kündigen dann an, dass jetzt eine Liste von Elementen folgt, für die gilt: (a) Sie sollen in genau der Reihenfolge ausgegeben werden, in der sie auftauchen. (b) Sie sollen mit Hilfe der Packages "verpackt" werden, die hinter dem Namen des Elements jeweils angegeben sind.

In Zeile 12-13 erscheint dementsprechend eine Liste von Elementen zusammen mit den für ihre Verarbeitung anzuwendenden Packages.

```
14        :form['</select></form></center><center>'],
```

In Zeile 14 wird durch :form[ ] eine Zeichenkette ausgeschrieben. In diesem Fall ein Teil der Formular-Tags in der HTML-Seite.

```
15
:package[:neighbour[:form["@=ref@",image,fit],first,group],firstsma],
```

In Zeile 15 wird ein Package definiert, das die Einbaufunktion :neighbour[ ] aktiviert. Durch neighbour[ ] kann man einzelne Bilder innerhalb von Bildgruppen gezielter auswählen. Es wird in dieser Zeile das Package firstsma, welches in der Datei pack.1 definiert ist, zusammen mit der Einbaufunktion neighbour[ ] verpackt.

Diese Einbaufunktion :neighbour[ ] erwartet drei Argumente:

-> das erste Argument stellt einen Elementnamen dar, der vom Datentyp image sein muss und der näher die Bildnummer definiert. In unserem Fall ist es jedoch der Aufruf einer anderen Einbaufunktion, nämlich form[ ]. Die Einbaufunktion form[ ] erwartet wiederum drei Argumente, als erstes wird ein Character String vorgeschrieben, in dem Fall wird hier der Ersetzungsmarker @=ref@ angegeben. Das zweite optionale Argument muss ein Schlüsselwort sein, das genauere Informationen zum Datentyp gibt (Catecory, Date, Image, Location, Number, Relation oder Text). In Zeile 15 ist der Datentyp selbstverständlich image. Das dritte optionale Argument bezeichnet einen Objektnamen, hier die Auflösung fit.

-> das zweite Argument von neighbour[ ] ist ein Schlüsselwort, dass das Bild innerhalb einer Bildgruppe anspricht, hier wird das erste Bild innerhalb der Bildgruppe ausgewählt. Zur Auswahl stehen neben first noch previous, next und last.

-> das dritte Argument von neighbour[ ] ist ein Schlüsselwort, dass group oder document lauten kann.

```
16
:always[:package[:neighbour[:form["@=ref@",image,fit],previous,group],prevsma],'
<img src="@=AServer@buttons/VorherigeSrg.gif">'],
```

In Zeile 16 werden durch die Einbaufunktion :always[ ] noch verschiedene andere Direktiven aktiviert.

Die Einbaufunktion always[ ] erwartet vier Argumente:

- ➔ Das erste erforderliche Argument gibt einen Hinweis auf den Pfad mit einem Element als letzter Komponente.
- ➔ Das zweite erforderliche Argument ist ein Charakter String.
- ➔ Das dritte optionale Argument ist ein Schlüsselwort, das den Datentyp definiert.
- ➔ Das vierte Argument bezeichnet einen Objektnamen.

Unter anderem wird durch :form[ ] das Icon für die „Vorherige Seite“ bei dem Package prevsma aktiviert.

```
17
:always[:package[:neighbour[:form["@=ref@",image,fit],next,group],nextsma],'<img
src="@=AServer@buttons/NaechsteSrg.gif">'],
```

In Zeile 17 werden durch die Einbaufunktion :always [ ] verschiedene andere Direktiven aktiviert. Unter anderem wird das Icon für die „Nächste Seite“ auf der HTML-Seite ausgegeben.

```
18        :package[:neighbour[:form["@=ref@",image,fit],last,group],lastsma],
19 :form['<img alt="Diese Aufl&ouml;sung" src="@=AServer@buttons/AugeSg.gif">'],
```

In Zeile 18 umschließt ein Package die Einbaufunktionen neighbour[ ], die diesmal das Package für die lastsma, verantwortlich für die Darstellung des Icons „Letzte Seite“, aufruft. Durch die Einbaufunktion :form[ ] wird ebenso das Icon für die nächste Stufe der Auflösung in die HTML-Seite ausgeschrieben.

```
20           :package[:form["@=ref@"],refmedpage],
21           :package[:form["@=ref@"],refpropage],
22           :package[:form["@=ref@"],refbigpage],
```

Zeile 20-23 definiert die Ausgabe der Packages für die Auflösungen von indiv, film und max.

```
23          :form['</center><p><center>'],
```

In Zeile 23 wird durch form eine Zeichenkette zur Positionierung des HTML-Codes ausgeschrieben.

```
24          :package[:form["@=ref@"],smapage]
```

Zeile 24 definiert speziell die Ausgabe des Packages für die fit Auflösung.

```
25 stop form=html;package=pagewrap
```

In Zeile 25 wird durch form=html vereinbart, dass der Inhalt als Character Entities ausgegeben wird. Zum Schluss wird das Package pagewrap aufgerufen.

5.4. Kataloge

Kataloge sind Vorlagen, die die Suchzeit während einer Datenbankabfrage verkürzen. Bei einem Katalog handelt es sich um eine Liste von Suchbegriffen, die anhand eines Indexes sortiert werden.

Wenn man z.B. einen bestimmten Begriff in einer Datenbank finden möchte, so werden ohne Kataloge alle Dokumente der Datenbank durchsucht, die angeforderten Informationen ausgewählt und dann ausgegeben. Der Einsatz von Katalogen verkürzt diese Prozedur, weil nur ein spezieller vorgefertigter Teil der Datenbank abgefragt wird. Das bedeutet, dass bestimmte Elemente (wie Namen, Begriffe etc...) nur einmal im Katalog vorkommen, egal wie oft sie wirklich in der Datenbank enthalten sind. Nach Abfrage des Katalogs werden dann alle Ergebnisse angezeigt, in denen das gesuchte Element enthalten war. Kataloge sind somit Kurzlisten, Ausschnitte aus der Datenbank.

Nutzen von Katalogen:

1. um den Abfrageprozess zu beschleunigen,
2. um die Volltextabfrage zu erleichtern,
3. und um Information zu kategorisieren (’keywording’).

Bei Katalogen werden weiterhin 5 verschiedene Abfragekontexte unterschieden:
1) Abfrage nach einem Wort: [complete]

2) Abfrage nach dem Wortanfang: [start]

3) Abfrage nach der Wortendung: [limit]

4) Abfrage nach Algorithmen über soundex [algorithm]

5) Abfrage nach mehreren Begriffen oder umfangreichen Texten (dann können der gesamte String oder die Einzelworte im Katalog verzeichnet werden).

Ein Beispielkatalog des Stichwerke-Servers:

```
query name=fauzk;part=/image
catalogue name=pages;part=:extref;overwrite=yes;
          type=term
```

Bei der Query-Abfrage wird zunächst der Name der Datenbank angesprochen, auf die sich später der Katalog beziehen soll. Durch part wird ein bestimmter Teil der Datenbank festgelegt, den der Katalog einschließt. Der Katalogname ist in unserem Fall pages; davon wiederum der Teil extref. Der Befehl type=term bedeutet, dass ein Eintrag als Ganzes in den Katalog eingehen soll. Ein Eintrag darf 255 Zeichnen lang sein. Wenn type=term nicht spezifiziert worden wäre, dann wäre versucht worden, den aus der Datenbank gewonnen Text in Einzelworte zu zerlegen. Als Worttrenner hätten standardmäßig Sonderzeichen (+-,*;:/!) und die Leerstelle gegolten.

5.5. Packages

Grundlegend haben Packages zwei Aufgaben:

- Packages sind „Verpackungsanweisungen“ und dienen dazu eine bestimmte Anzahl von Zeichenketten anzusprechen, um an unterschiedlichen Stellen der Kommandosprache die Textausgabe zu beeinflussen.
- Packages haben darüber hinaus die Aufgabe dafür zu sorgen, dass das was verpackt werden soll, zunächst noch andere Schritte durchlaufen soll.

Mit Packages lassen sich Datenbestandteile nach bestimmten Regeln verpacken. Konkret eignen sich Packages als sehr gute Hilfsmittel, um einen dynamischen Webserver aufzubauen. Durch Packages werden einzelne Bestandteile einer HTML-Seite generiert. Diese HTML-Seiten enthalten neben ihrer Grundformatierung beispielsweise CSS-

Anweisungen, JavaScripts und beliebige andere Elemente. Das Resultat einer Datenbankabfrage wird dabei direkt in die durch die jeweiligen Packages erzeugte HTML-Seite integriert.

Das Package demo wird aus einer Prozedur heraus aufgerufen

```
package[:form["Beispiel"],demo]
```

so sieht die Grundform des Package demo aus

```
01 item name=demo;usage=package
02 fields start="<start>";
03       first="<first>";
04       second="<second>";
05       limit="<limit>"
```

und die Ausgabe erscheint auf diese Weise

```
<start>Beispiel<first>Beispiel<second>Beispiel<limit>
```

Erläuterung des Package demo: Durch das Kommando item name wird das Package mit dem Namen demo in die logische Umwelt von Kleio eingeführt. Durch die Anweisung usage=package wird seine Verwendung als logisches Objekt festgelegt.
In Zeile 02 beginnt die Fields Direktive. Durch diese Direktive wird die Datenbankausgabe mit unterschiedlichen Formatierungsanweisungen versehen.

Der Fieldsparameter start in Zeile 02 markiert den Anfangswert, er zeichnet den Beginn des zu verpackenden Inhalts aus. Nach fields start= folgt eine Zeichenkette (bzw. Konstante) mit beliebigem Inhalt, die in Hochkommata stehen muss. Die Länge aller Zeichenketten beim Fieldsparameter ist unbegrenzt. Der Parameter first in Zeile 03 bewirkt, dass der Inhalt zu Anfang ausgegeben werden muss. Der jeweilige Inhalt von first wird wiederum als eine in Hochkommata ausgezeichnete Zeichenkette (bzw. Konstante) ausgegeben. Der Parameter second in Zeile 04 wird demnach nach first ausgeführt. Die Parameter first und second bezeichnet man auch als Komplementärparameter. Letztlich entspricht der Parameter limit in Zeile 05 dem Ende einer Package Anweisung. Der jeweilige Inhalt von limit wird erneut als eine in Hochkommata ausgezeichnete Zeichenkette (bzw. Konstante) ausgegeben. Die Parameter start und limit sind gleichfalls Komplementärparameter.

Die Parameter start und limit beeinflussen die Ausgabe von Elementen, während sich die in ihrer Bestimmung ähnlichen Parameter origin und complete auf die Ausgabe von Gruppen beziehen.[12]

5.5.1 Packages des Stichwerke-Servers

Nun wird das grob skizzierte logische Objekt der Klasse Package im Detail noch einmal am konkreten Beispiel erläutert. Die Datei pack.1 des Stichwerke-Servers unterscheidet sich zunächst einmal von den bisherigen Übungsdateien, da es sich in dieser Datei schon um eine Zusammenstellung von ca. 30 Packages handelt. Wobei man noch betonen muss, dass dieses Projekt wirklich erst am Anfang der Entwicklung steht.

5.5.2 Textpräsentation

Zunächst werden die Packages für die Darstellung des Textes bzw. des Icons erläutert.

[12] Siehe zu weiteren Packages Erläuterungen Nemitz: Die neue Kleio Dokumentation. Hier unter: commands/item/usage/package.

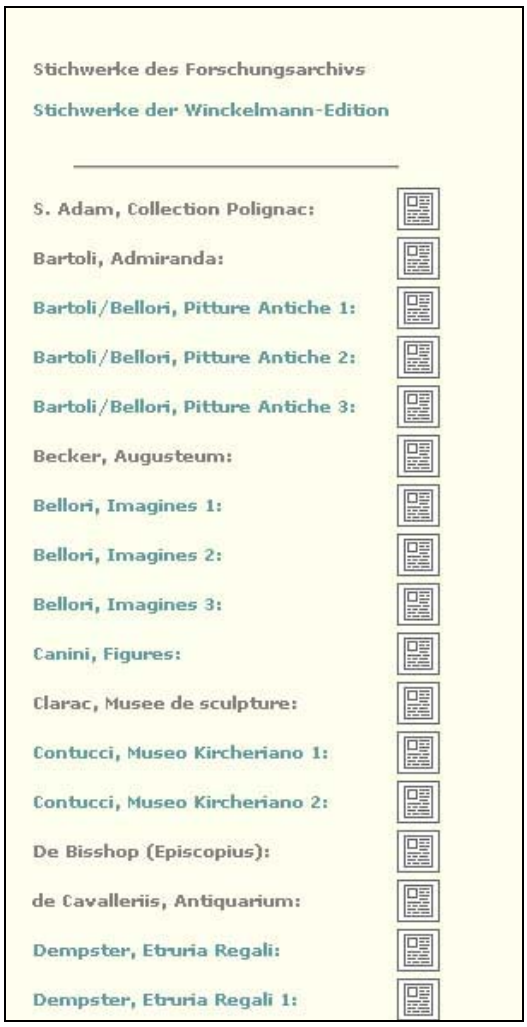


Abbildung 15 Darstellung der Auswahlliste für die digitalen Stichwerke.

Dieser Abbildung entspricht der folgende Seitenquelltext.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
   <html>
   <head>
         <title>Statische Seite</title>
         <link                    rel=stylesheet                    type="text/css"
href="http://134.95.113.230/stichwerke/css/body.css">
   </head>
   <body background="http://134.95.113.230/stichwerke/buttons/ivory.gif">
        <table border="0">
                      <tr>
                              <td><b><font size="-7" color="gray"><br>
                                                    <br>
                                                </font><font              size="-7"
color="#5f9ea0"><br>
                                                </font></b></td>
```

```
                        <td></td>
                    </tr>
                    <tr>
                        <td><b><font   size="-7"   color="gray">Stichwerke
des Forschungsarchivs <br>
                                                <br>
                                            </font><font             size="-7"
color="#5f9ea0">Stichwerke der Winckelmann-Edition<br>
                                            </font></b></td>
                        <td></td>
                    </tr>
                    <tr>
                        <td><b><font size="-7" color="#5f9ea0"><br>
                                            </font></b></td>
                        <td></td>
                    </tr>
                    <td colspan="3">
                        <hr>
                    </td>
                    </tr>
                    <tr>
                        <td><b><font   size="-7"   color="gray">S.   Adam,
Collection Polignac:</font></b></td>
                        <td><a                              target="pages"
href="http://134.95.113.230/stich-cgi/kleioc/0010/exec/pagesma/%22recueil001-
0001.jpg%22"><img                                                  border="0"
src="http://134.95.113.230/stichwerke/buttons/seite.gif"   alt="Abbildungen   des
Drucks (erste Seite)"></a></td>
                    </tr>
                    <tr>
                        <td><b><font    size="-7"    color="gray">Bartoli,
Admiranda:</font></b></td>
                        <td><a                              target="pages"
href="http://134.95.113.230/stich-cgi/kleioc/0010/exec/pagesma/%22bartolo001-
0001.jpg%22"><img                                                  border="0"
src="http://134.95.113.230/stichwerke/buttons/seite.gif"   alt="Abbildungen   des
Drucks (erste Seite)"></a></td>
                    </tr>
                    <tr>
                        <td><b><font                                size="-7"
color="#5f9ea0">Bartoli/Bellori, Pitture Antiche 1:</font></b></td>
                        <td><a                              target="pages"
href="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagesma/%22bartolpitture01front-0001.jpg%22"><img
```

```
border="0"              src="http://134.95.113.230/stichwerke/buttons/seite.gif"
alt="Abbildungen des Drucks (erste Seite)"></a></td>
                      </tr>
                      ...
                      ...
                      ...
                      <tr>
                             <td><b><font  size="-7"  color="#5f9ea0">Zanetti,
Delle Statue Antiche 2:</font></b></td>
                             <td><a                              target="pages"
href="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagesma/%22zanetti02front-0001.jpg%22"><font
color="#5f9ea0"><img                                                 border="0"
src="http://134.95.113.230/stichwerke/buttons/seite.gif"  alt="Abbildungen  des
Drucks (erste Seite)"></font></a></td>
                      </tr>
               </table>
       </body>
</html>
```

Beispielcode für die Anzeige eines einfachen HTML-Grundgerüsts beim Package wrapbooks.

```
item name=wrapbooks;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">\n
  <html>\n
  <head>\n
        <title>Statische Seite</title>\n
        <link rel=stylesheet type="text/css" href="@=AServer@css/body.css">\n
  </head>\n
  <body background="@=AServer@buttons/ivory.gif">\n
  <table border="0">\n
  <tr><td nowrap><b>FAUzK Nr.</b></td><td colspan="2"><b>Inhalt</b></td></tr>\n
  <tr><td colspan="3"><hr></td></tr>';
       limit='</table></body>\n</html>'
exit name=wrapbooks
```

Hier wird das einfache Grundgerüst für die statische HTML-Seite angezeigt. Mit Hilfe von Ersetzungsmarkern werden Stylesheet-Anweisungen und Hintergrundbilder eingebunden. Dass, was durch die Ersetzungsmarker aktiviert werden soll, wird innerhalb der @-Zeichen

eingefügt. Das Datenbanksystem Kleio findet in einer supply.ini Datei die Werte, die den Ersetzungsmarkern zugeordnet werden. So wird in unserem Fall dem Ausdruck @=AServer@ der Wert http://134.95.113.230:16080/stichwerke/ zugewiesen.

Beispielcode des Packages tabeline für eine Tabelle:

```
item name=tableline;usage=package;overwrite=yes
fields start='<tr><td><b>';
       limit=':</b></td><td>'
exit name=tableline
```

Die Tabelle schreibt eine Tabellenzeile und eine Datenzelle aus. Der dynamisch generierte Textinhalt wird außerdem noch durch die Ausgabe des Bold-Tags fett formatiert und am Ende mit einem Doppelpunkt versehen.

Beispielcode für die dynamische Generierung der Buchdarstellung mit Hilfe eines Packages:

```
item name=startbook;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<a target="pages" href="@=ADynamic@pagesma/%22';
       limit='%22"><img       border="0"       src="@=AServer@buttons/seite.gif"
alt="Abbildungen des Drucks (erste Seite)"></a>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=startbook
```

Durch anklicken des Icons seite.gif () wird die erste Seite eines Stichwerks im rechten Frame der Web-Seite angezeigt. Das Package startbook generiert folglich im HTML-Code einen Link, bewirkt das Anzeigen des Icons seite.gif, ordnet die Darstellung der Images im rechten Frame an und zeigt einen alternierenden Text an.

5.5.3. Bildpräsentation

Nun werden näher die Packages für die Image Präsentation erläutert.

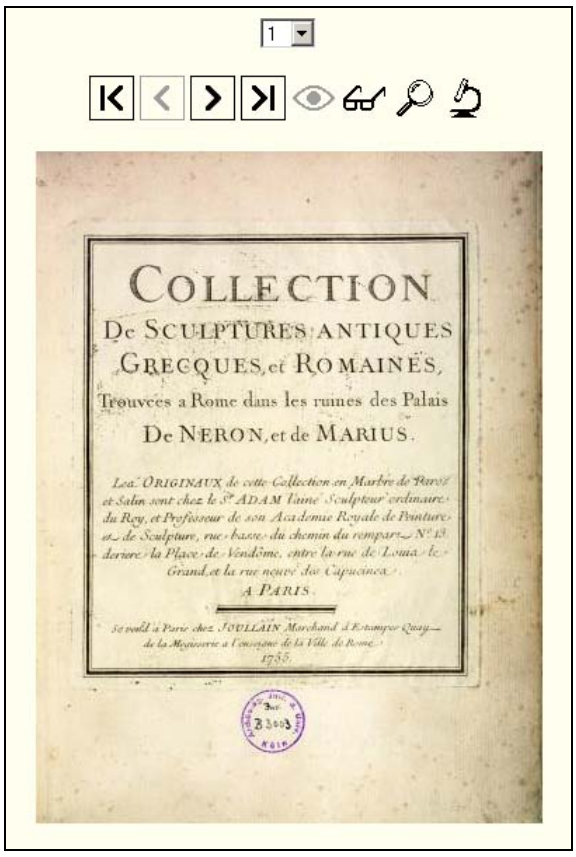


Abbildung 16 Bildpräsentation beim Stichwerke-Server.

Dieser HTML-Seite entspricht folgender Seitenquelltext:

```
<html><head>
         <link                    rel=stylesheet                    type="text/css"
href="http://134.95.113.230/stichwerke/css/body.css">
         <title>Suchergebnis</title>
   <BASE target="_self">
   </head>
   <body background="http://134.95.113.230/stichwerke/buttons/ivory.gif">
   <basefont size="3" face="arial,sans-serif">

<center><form action="http://134.95.113.230/stich-cgi/kleioc" method="post">
<input type="hidden" name="!_kleioprot" value="0010">                    <input
type="hidden" name="_1" value="pagesma">                    <select name="_2"
size="1" onChange="this.form.submit()">
                         <option selected value="recueil001-0001.jpg">
                         1</option>
                      <option  value="recueil001-0002.jpg">
                      2</option>
                   <option  value="recueil001-0003.jpg">
```

```
                    3</option>
                                                ...
                                                ...
                                                ...
                          <option  value="recueil001-0029.jpg">
                          26</option>
                       <option  value="recueil001-0030.jpg">
                       27</option>
                       <option  value="recueil001-0031.jpg">
                       28</option>
</select></form></center><center>
<A                                              HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagesma/%22recueil001%2d0001%2ejpg%22">   <img   alt="Erste
Seite"                                                                  border="0"
SRC="http://134.95.113.230/stichwerke/buttons/AnfangSr.gif"></A>
<img src="http://134.95.113.230/stichwerke/buttons/VorherigeSrg.gif">
<A                                              HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagesma/%22recueil001%2d0002%2ejpg%22">                <img
alt="N&auml;chste                          Seite"                       border="0"
SRC="http://134.95.113.230/stichwerke/buttons/NaechsteSr.gif"></A>
<A                                              HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagesma/%22recueil001%2d0065%2ejpg%22">   <img   alt="Letzte
Seite" border="0" SRC="http://134.95.113.230/stichwerke/buttons/EndeSr.gif"></A>
<img                           alt="Diese                            Aufl&ouml;sung"
src="http://134.95.113.230/stichwerke/buttons/AugeSg.gif">
<A               target="inhalt"                HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagemed/%22recueil001%2d0001%2ejpg%22">                <img
alt="Arbeitsqualit&auml;at"                                             border="0"
SRC="http://134.95.113.230/stichwerke/buttons/BrilleS.gif"></A>
<A               target="inhalt"                HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagepro/%22recueil001%2d0001%2ejpg%22">                <img
alt="Verbesserte                       Qualit&auml;t"                   border="0"
SRC="http://134.95.113.230/stichwerke/buttons/LupeS.gif"></A>
<A               target="inhalt"                HREF="http://134.95.113.230/stich-
cgi/kleioc/0010/exec/pagebig/%22recueil001%2d0001%2ejpg%22"> <img  alt="Maximale
Aufl&ouml;sung"                                                         border="0"
SRC="http://134.95.113.230/stichwerke/buttons/MikroS2.gif"></A>
</center><p><center>
<center><IMG
SRC="http://134.95.113.230/stichwerke/images/fit/recueil_sulptures/recueil001-
0001.jpg"></center>
</body>
</html>
```

Im Folgenden werden schrittweise die jeweiligen Packages erklärt, mit deren Hilfe diese HTML-Seite generiert wird.

Das Grundgerüst für die HTML-Ausgabe der Bildbearbeitungsseite sieht beim Package pagewrap folgendermaßen aus:

```
item name=pagewrap;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head>\n
        <link rel=stylesheet type="text/css" href="@=AServer@css/body.css">\n
        <title>Suchergebnis</title>\n
  <BASE target="_self">\n
  </head>\n
  <body background="@=AServer@buttons/ivory.gif">\n
  <basefont size="3" face="arial,sans-serif">';
       limit='</body>\n</html>'
exit name=pagewrap
```

Das Package smapage für die Bildpräsentation sieht folgendermaßen aus:

```
item name=smapage;usage=package;overwrite=yes
brackets sensitive=yes
field start='<center><IMG SRC="@=ARepository@fit/';
      limit='"></center>';
convert current=rawname;
        path=":filename[:image[:dump[rawname],text,fit]]";
exit name=smapage
```

Dieses Package sorgt dafür, dass das Bild in der Mitte dargestellt wird. Angezeigt wird in dem Fall die kleinste Auflösung „fit“. Im HTML-Code der späteren Internetseite wird nach der CGI-Generierung nur das Fettmarkierte jeweils dynamisch erzeugt werden:

```
<center><IMG
SRC="http://134.95.113.230/stichwerke/images/fit/recueil_sulptures/recueil001-0001.jpg"></center>
```

Bei der kleinsten Auflösung fit werden die Bilder in einer Größe von ca. 360 x 480 Pixel ausgegeben, so dass es bei dieser Auflösung möglich ist, die gesamte Buchseite auf einen Blick zu betrachten, ohne vertikal oder horizontal scrollen zu müssen.

Das Package smapage legt die Positionierung des Bildes in der Mitte fest und vergibt mit

cgiencode die URL Adresse des Digitalisats. Die Funktion cgiencode setzt Sonderzeichen in CGI-Convention um. Gemäß der Form: **:cgiencode[:element].**

:cgiencode[:element] bedeutet, dass der Inhalt von Element genommen wird und alles was nicht den lateinischen Grundalphabet von 10 Ziffern entspricht, kodiert wird, als durch %-Zeichen eingeleitete hexadezimale Codes wie oben beschrieben.

Darüber hinaus kann der Nutzer bei den Digitalisaten folgende Auflösungen einstellen:

 Kleinste Auflösung: Die Bilder stehen unter ca. 360 x 480 Pixel zur Verfügung. Vorteil: Bietet einen schnellen Überblick für den Nutzer.

Beispielcode für die kleinste Auflösung:

```
item name=refsmapage;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A target="inhalt" HREF="@=ADynamic@pagesma/%22';
       limit='%22">         <img         alt="&Uuml;berblick"         border="0"
SRC="@=AServer@buttons/AugeS.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=refsmapage
```

 Auflösung: ca. 980 x 1300 Pixel. Mit der Breite des Bildes auf etwa 1000 Pixel kann der Benutzer mit einer durchaus gängigen Monitorbildschirmauflösung (z. B.: bei 17 Zoll Monitoren) von 1024 x 768 Pixeln eine ganze Buchseite sich auf dem Monitor anzeigen lassen.

Beispielcode für die kleinste Auflösung:

```
item name=refmedpage;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A target="inhalt" HREF="@=ADynamic@pagemed/%22';
       limit='%22">       <img       alt="Arbeitsqualit&auml;at"       border="0"
SRC="@=AServer@buttons/BrilleS.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=refmedpage
```

 Auflösung: ca. 1580 x 2080 Pixel. Um noch mehr Teilstücke der Ansicht betrachten zu können, eignet sich die Ansicht mit der besserten Qualität.

```
item name=refpropage;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A target="inhalt" HREF="@=ADynamic@pagepro/%22';
       limit='%22">    <img    alt="Verbesserte    Qualit&auml;t"    border="0"
SRC="@=AServer@buttons/LupeS.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=refpropage
```

 Auflösung: ca. 2760 x 3666 Pixel. Um ganz wichtige Details des Digitalisats betrachten zu können, gibt es die höchste Auflösung.

```
item name=refbigpage;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A target="inhalt" HREF="@=ADynamic@pagebig/%22';
       limit='%22">    <img    alt="Maximale    Aufl&ouml;sung"    border="0"
SRC="@=AServer@buttons/MikroS2.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=refbigpage
```

Die Auflösung differiert natürlich geringfügig ja nach Format des jeweiligen Buches.

Innerhalb des einzelnen Buchbandes besteht weiterhin die Möglichkeit anhand folgender Navigationselementen zu manövrieren:

 Anfangsseite des Textes.

Beispielcode für das Package der ersten Seite:

```
item name=firstsma;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A HREF="@=ADynamic@pagesma/%22';
       limit='%22">       <img       alt="Erste       Seite"       border="0"
SRC="@=AServer@buttons/AnfangSr.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=firstsma
```

 Vorherige Seite.

Beispielcode für das Package der vorherigen Seite:

```
item name=prevsma;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A HREF="@=ADynamic@pagesma/%22';
       limit='%22">       <img       alt="Vorherige       Seite"       border="0"
SRC="@=AServer@buttons/VorherigeSr.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=prevsma
```

 Nächste Seite.

Beispielcode für das Package der nächsten Seite:

```
item name=nextsma;usage=package;overwrite=yes;
brackets sensitive=yes
fields start='<A HREF="@=ADynamic@pagesma/%22';
       limit='%22">      <img      alt="N&auml;chste      Seite"      border="0"
SRC="@=AServer@buttons/NaechsteSr.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=nextsma
```

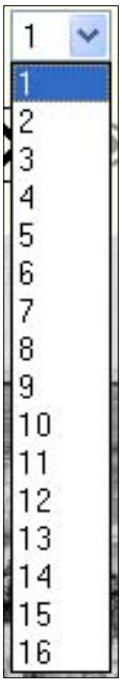
 Letzte Textseite.

Beispielcode für das Package der letzten Seite:

```
item name=lastsma;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<A HREF="@=ADynamic@pagesma/%22';
       limit='%22">       <img       alt="Letzte       Seite"       border="0"
SRC="@=AServer@buttons/EndeSr.gif"></A>'
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
exit name=lastsma
```

Zum Auswählen der Optionen für die Klappliste werden die Packages realid und showid

verwendet.




```
item name=realid;usage=package;overwrite=yes
fields start='<option ';
       first='value="';
       limit='">';
convert usage=primary;current=rawname;
      path="/test[:dump[rawname]   =   :dump[thispage]]:form['   selected
']";
exit name=realid

item name=showid;usage=package;overwrite=yes
fields limit="</option>"
exit name=showid
```

5.6. Zusätzliche Dateien

Außer den wichtigsten Grunddateien gibt es noch verschiedene Textdateien, die jeweils bestimmte Anweisungen an das System stellen.

5.6.1. write.books

Dateien dieses Typs schreiben für gewöhnlich die Ergebnisse in einzelne, besonders entworfene Ausgabeseiten. In diesem Fall wird die Ausgabe in das Verzeichnis pages in eine books.html geleitet.

Aufruf der Datei write.books:

```
options signs=10000
query name=fauzk;part=/itemstmt/bibid
index part=:value;
         package=tableline;
```

```
      part=:package[/back[2]/part[/div,1]/part[/page,1]/image:extref,startbook];
      part=:form["</td></tr>"];
      type=list
stop target="../pages/books.html";overwrite=yes;
     form=html;package=wrapbooks
```

Für die Ausgabe der HTML-Seite books.html wird der genaue Aufbau festgelegt. Bei der Datenbank fauzk wird der Inhalt des Elements /itemstmt/bibid abgefragt. Durch den index Befehl soll sie eine sortierte Liste ausgeben. Es werden weiterhin verschiedene Packages für die Ausgabe vereinbart. Ferner wird durch eine form:[ ] Vereinbarung die End-Tags einer Tabelle ausgeschrieben. Als Typ wird eine Liste vereinbart. Als Ziel der Datenausgabe wird die books.html festgelegt. form=html führt dazu, dass noch verbleibende Sonderzeichen zu Character Entities umgewandelt werden.

5.7. Einbindung von ToC Dateien

Mit Hilfe einer ToC (**T**able **o**f **C**ontent) Datei kann man für die einzelnen digitalisierten Abbildungen eines Stichwerks, bzw. jeder beliebigen Abbildung von Druckwerken allgemein, eine Art „Inhaltsverzeichnis“ erstellen. Die ToC Dateien werden mit Hilfe eines von der HKI entwickelten ToC Editors erstellt, auf den hier jedoch an dieser Stelle nicht weiter eingegangen wird.

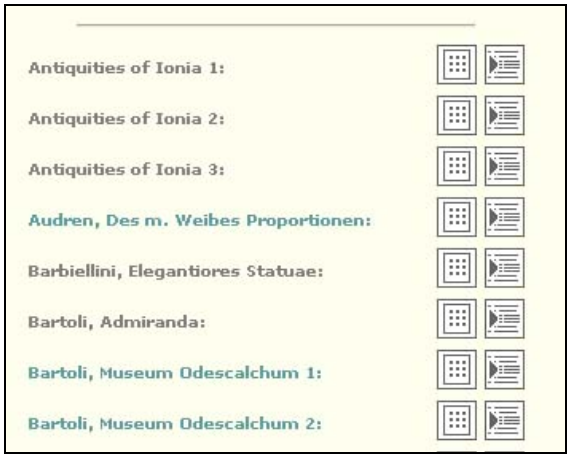


Abbildung 17: Darstellung der Auswahlliste für die digitalen Stichwerke erweitert um ein Icon für die ToC Dateien.

Zunächst einmal wird im Auswahlbereich des Stichwerke Servers nur ein weiteres Icon für

die Darstellung des Inhaltsverzeichnisses der ToC Dateien ( ) eingeführt. Aktiviert man dieses Icon, so erscheint im rechten Anzeigebereich des Stichwerkeservers eine Art „Inhaltsverzeichnis“, in unserem Fall wurde das Stichwerk: *„Elegantiores Statuae“* von Barbiellini ausgewählt.

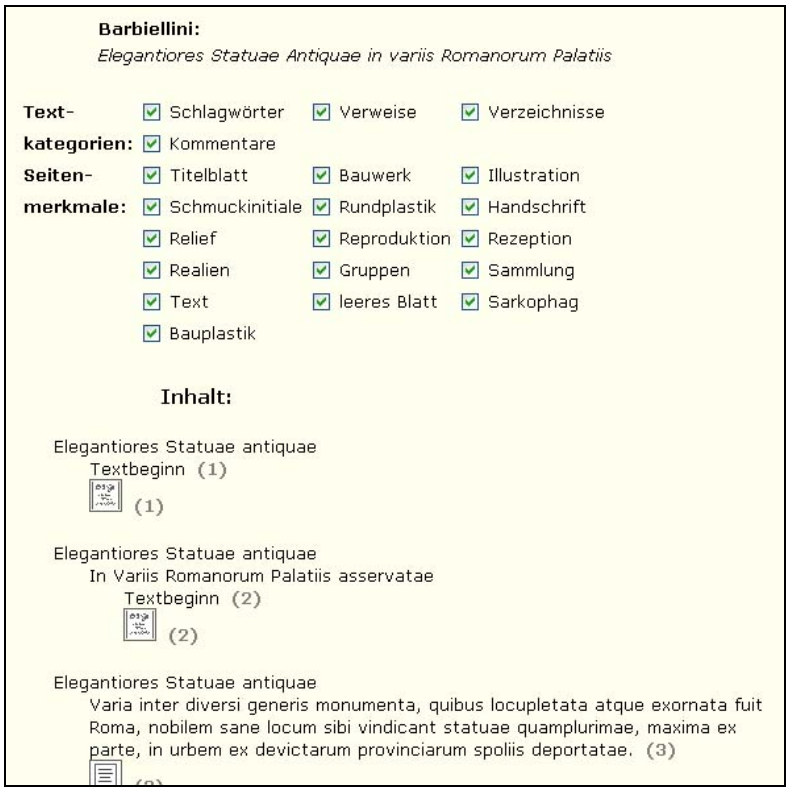


Abbildung 18: Anzeige des ToC "Inhaltsverzeichnisses".

Bei der Inhaltsübersicht besteht die Möglichkeit, zusätzliche Informationen zu den Stichwerken (Textkategorien) oder zu den Seiten (Seitenmerkmale) auszuwählen. Die Checkboxbuttons sind per Standard aktiviert, können jedoch nach Wunsch ausgeschaltet werden. Unter der Rubrik Textkategorien erhält man weitere Informationen zu Schlagwörtern, Verweisen, Verzeichnissen und Kommentaren. Unter der Rubrik Seitenmerkmale finden sich Informationen zu Titelblatt, Bauwerk, Illustration, Schmuckinitiale, Rundplastik, Handschrift, Relief, Reproduktion, Rezeption, Realien, Gruppen, Sammlung, Text, leeres Blatt, Sarkophag und Bauplastik. Jede dieser Kategorien und Merkmale ist im Inhaltsverzeichnis mit einem Icon vertreten. Sobald hinter dem jeweiligen Icon auf die geklammerte Seitenzahl geklickt wird, kann der Benutzer sich das dazugehörige Digitalisat anzeigen lassen. Als nächstes stellt sich die Frage, wie diese ToC Dateien in den Webserver eingebunden werden.

5.7.1 Datei pack.1 (Teil I)

```
. . .
item name=starttoc;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<a onClick="javascript:supplyOptions(\'@=ADynamic@fauzktoc/%22';
       limit='%22\');"><img     border="0"     src="@=AServer@buttons/inhalt.gif"
alt="Inhalts&uuml;bersicht"></a></td></tr>'
exit name=starttoc
. . .
```

Erläuterung:

- Der erste Schritt, um die ToC Dateien anzeigen zu lassen, muss durch ein Package in der Datei pack.1 vorgenommen werden. Hier wird zunächst das Package starttoc definiert, dass die ToC Dateien dynamisch verknüpft und ein Icon im Auswahlbereich für die digitalen Stichwerke erzeugt.

5.7.2 Datei write.books

```
01 options signs=10000
02 query name=fauzk;part=/itemstmt/bibid
03 index part=:interpret[:value,books,label] &* :form['>'];
04         write=no;
05      part=:interpret[:value,books,label];
06         package=tableline;without=yes;
07      part=:package[:value,startthumb];
08      part=:package[:value,starttoc];
09      part=:form["</td></tr>"];
10      type=list
11
12 stop target="../pages/books.html";overwrite=yes;
13     form=html;package=wrapbooks
```

Erläuterung:

- Die Datei write.books hat sich hier an einigen Stellen verändert, wovon für unsere Zwecke nur die Zeile 08 von Interesse ist. Der Parameter part schreibt hier das Package starttoc mit dem Element value aus. Das linke Auswahlbereich für die

digitalen Stichwerke würde nun so aussehen, wie in Abb. 17 dargestellt.

5.7.3 Datei pack.1 (Teil II)

```
. . .
note ================================================================
note ================ table of content packages =====================
note ================================================================

item name=tocwrap;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head>\n
        <link rel=stylesheet type="text/css" href="@=AServer@css/body.css">\n
        <script           language="JavaScript"           type="text/javascript"
src="@=AServer@scripts/supplyOptions.js">\n
        </script>\n
        <script           language="JavaScript"           type="text/javascript"
src="@=AServer@scripts/displayOptions.js">\n
        </script>\n
        <title>Inhaltsverzeichnis.</title>\n
  <BASE target="_self">\n
  </head>\n
  <body background="@=AServer@buttons/ivory.gif">\n';
       limit='</body>\n</html>'
exit name=tocwrap

item name=newlevel;usage=package;overwrite=yes
fields origin='<UL compact="yes">';
       complete='</UL>'
exit name=newlevel

item name=headerline;usage=package;overwrite=yes
fields start='';

convert current=currenttext;
        path=':dump[currenttext]                                                |+
:package[/part[/page,1]/image:extref,pageref]'
exit name=headerline

note ================ packages addline und addicon ========================

item name=addline;usage=package;overwrite=yes
brackets sensitive=yes
```

```
condition condition=':type="keyword"'
convert current=currenttext;
        path='@(Trigcatkeyw@form["<br>Schlagw&ouml;rter:         <i>"]         &+
:dump[currenttext]               &+               :form["</i>"]               |+
:package[/back[1]/image:extref,pageref]@)Trigcatkeyw@@[Trigcatkeyw@:form[""]@]Tr
igcatkeyw@'

condition condition=':type="reference"'
convert current=currenttext;
        path='@(Trigcatrefe@form["<br>Verweis:  <i>"]  &+  :dump[currenttext]  &+
:form["</i>"]                                                                  |+
:package[/back[1]/image:extref,pageref]@)Trigcatrefe@@[Trigcatrefe@:form[""]@]Tr
igcatrefe@'

condition condition=':type="tableHeading"'
convert current=currenttext;
        path='@(Trigcattabl@form["<br>Verzeichnis:  <i>"]  &+  :dump[currenttext]
&+                               :form["</i>"]                                 |+
:package[/back[1]/image:extref,pageref]@)Trigcattabl@@[Trigcattabl@:form[""]@]Tr
igcattabl@'

condition condition=':type="comment"'
convert current=currenttext;
        path='@(Trigcatcomm@form["<br>Kommentar: <i>"] &+ :dump[currenttext] &+
:form["</i>"]                                                                  |+
:package[/back[1]/image:extref,pageref]@)Trigcatcomm@@[Trigcatcomm@:form[""]@]Tr
igcatcomm@'
exit name=addline

item name=addicon;usage=package;overwrite=yes
brackets sensitive=yes

condition condition=':property="titlepage"'
convert
        path='@(Trigpagtitl@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/titelblatt.gif\"                       alt=\"Titelblatt\"
onMouseOver=\"window.status=\'Titelblatt\'\"
onMouseOut=\"window.status=\'\'\">"]                                           |+
:package[/image:extref,pageref]@)Trigpagtitl@@[Trigpagtitl@:form[""]@]Trigpagtit
l@'

condition condition=':property="bauwerk"'
convert
        path='@(Trigpagbauw@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/bauwerk.gif\" alt=\"Bauwerk\"
```

```
         onMouseOver=\"window.status=\'Bauwerk\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagbauw@@[Trigpagbauw@:form[""]@]Trigpagbau
w@'

condition condition=':property="illustration"'
convert
        path='@(Trigpagillu@form["<br><img                        border=\"0pt\"
src=\"@=AServer@buttons/illustration.gif\"                  alt=\"Illustration\"
onMouseOver=\"window.status=\'Illustration\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagillu@@[Trigpagillu@:form[""]@]Trigpagill
u@'

condition condition=':property="initial"'
convert
        path='@(Trigpaginit@form["<br><img                        border=\"0pt\"
src=\"@=AServer@buttons/schmuck.gif\"                   alt=\"Schmuckinitialen\"
onMouseOver=\"window.status=\'Schmuckinitialen\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpaginit@@[Trigpaginit@:form[""]@]Trigpagini
t@'

condition condition=':property="rplastik"'
convert
        path='@(Trigpagrund@form["<br><img                        border=\"0pt\"
src=\"@=AServer@buttons/rplastik.gif\"                      alt=\"Rundplastik\"
onMouseOver=\"window.status=\'Rundplastik\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagrund@@[Trigpagrund@:form[""]@]Trigpagrun
d@'

condition condition=':property="notes"'
convert
        path='@(Trigpagnote@form["<br><img                        border=\"0pt\"
src=\"@=AServer@buttons/handschrift.gif\"  alt=\"Handschriftliche  Anmerkungen\"
onMouseOver=\"window.status=\'Handschriftliche                  Anmerkungen\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagnote@@[Trigpagnote@:form[""]@]Trigpagnot
e@'

condition condition=':property="bplastik"'
convert
        path='@(Trigpagbpla@form["<br><img                        border=\"0pt\"
src=\"@=AServer@buttons/bplastik.gif\"                      alt=\"Bauplastik\"
```

```
onMouseOver=\"window.status=\'Bauplastik\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagbpla@@[Trigpagbpla@:form[""]@]Trigpagbpl
a@'

condition condition=':property="relief"'
convert
        path='@(Trigpagreli@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/relief.gif\"                                alt=\"Relief\"
onMouseOver=\"window.status=\'Relief\'\" onMouseOut=\"window.status=\'\'\">"] |+
:package[/image:extref,pageref]@)Trigpagreli@@[Trigpagreli@:form[""]@]Trigpagrel
i@'

condition condition=':property="repro"'
convert
        path='@(Trigpagrepr@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/repro.gif\"                           alt=\"Reproduktion\"
onMouseOver=\"window.status=\'Reproduktion\'\"
onMouseOut=\"window.status=\'\'\">"]                                          |+
:package[/image:extref,pageref]@)Trigpagrepr@@[Trigpagrepr@:form[""]@]Trigpagrep
r@'

condition condition=':property="rezep"'
convert
        path='@(Trigpagreze@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/rezept.gif\"                              alt=\"Rezeption\"
onMouseOver=\"window.status=\'Rezeption\'\" onMouseOut=\"window.status=\'\'\">"]
|+
:package[/image:extref,pageref]@)Trigpagreze@@[Trigpagreze@:form[""]@]Trigpagrez
e@'

condition condition=':property="real"'
convert
        path='@(Trigpagreal@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/real.gif\"                                 alt=\"Realien\"
onMouseOver=\"window.status=\'Realien\'\"   onMouseOut=\"window.status=\'\'\">"]
|+
:package[/image:extref,pageref]@)Trigpagreal@@[Trigpagreal@:form[""]@]Trigpagrea
l@'

condition condition=':property="grup"'
convert
        path='@(Trigpaggrup@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/gruppen.gif\"                              alt=\"Gruppen\"
onMouseOver=\"window.status=\'Gruppen\'\"   onMouseOut=\"window.status=\'\'\">"]
```

```
|+
:package[/image:extref,pageref]@)Trigpaggrup@@[Trigpaggrup@:form[""]@]Trigpaggru
p@'

condition condition=':property="samm"'
convert
        path='@(Trigpagsamm@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/sammlung.gif\"                          alt=\"Sammlung\"
onMouseOver=\"window.status=\'Sammlung\'\"  onMouseOut=\"window.status=\'\'\">"]
|+
:package[/image:extref,pageref]@)Trigpagsamm@@[Trigpagsamm@:form[""]@]Trigpagsam
m@'

condition condition=':property="text"'
convert
        path='@(Trigpagtext@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/text.gif\"                                alt=\"Text\"
onMouseOver=\"window.status=\'Text\'\"  onMouseOut=\"window.status=\'\'\">"]  |+
:package[/image:extref,pageref]@)Trigpagtext@@[Trigpagtext@:form[""]@]Trigpagtex
t@'

condition condition=':property="leer"'
convert
        path='@(Trigpagleer@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/leer.gif\"             alt=\"leeres             Blatt\"
onMouseOver=\"window.status=\'leeres                                Blatt\'\"
onMouseOut=\"window.status=\'\'\">"]                                        |+
:package[/image:extref,pageref]@)Trigpagleer@@[Trigpagleer@:form[""]@]Trigpaglee
r@'

condition condition=':property="sarg"'
convert
        path='@(Trigpagsarg@form["<br><img                          border=\"0pt\"
src=\"@=AServer@buttons/sarg.gif\"                              alt=\"Sarkophag\"
onMouseOver=\"window.status=\'Sarkophag\'\" onMouseOut=\"window.status=\'\'\">"]
|+
:package[/image:extref,pageref]@)Trigpagsarg@@[Trigpagsarg@:form[""]@]Trigpagsar
g@'


exit name=addicon

note ================================================================

item name=pageref;usage=package;overwrite=yes
```

```
brackets sensitive=yes
fields start=' <a href="@=ADynamic@pagesma/%22';
       first='%22"> (';
       limit=")</a>"
convert current=usesignature;
     path=':cgiencode[:dump[usesignature],url]'
convert usage=first;
     path=':nativeno';
exit name=pageref

item name=headingin;usage=package;overwrite=yes
fields start='<table border="0pt" width="80%" align="center"><tr><td><b>';
       limit=': </b></td></tr>'
exit name=headingin

item name=headingout;usage=package;overwrite=yes
fields start='<tr><td><i>';
       limit='</i></td></tr></table>'
exit name=headingout

note ====================packge textsorten ================================

item name=textsorten;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<form name=toccontrol>\n
              <table border="0pt">\n
                 <tr><td><b>Text-</b></td>\n
                     <td><input  type="checkbox"  name="catkeyw"  value="catkeyw"
onClick="javascript:refreshOptions(\'catkeyw\',\'@=ref@\');"></td><td>Schlagw&ou
ml;rter</td>\n
                     <td><input  type="checkbox"  name="catrefe"  value="catrefe"
onClick="javascript:refreshOptions(\'catrefe\',\'@=ref@\');"></td><td>Verweise</
td>\n
                     <td><input  type="checkbox"  name="cattabl"  value="cattabl"
onClick="javascript:refreshOptions(\'cattabl\',\'@=ref@\');"></td><td>Verzeichni
sse</td>\n
                 </tr>\n<tr><td><b>kategorien:</b></td>\n
                     <td><input  type="checkbox"  name="catcomm"  value="catcomm"
onClick="javascript:refreshOptions(\'catcomm\',\'@=ref@\');"></td><td>Kommentare
</td>\n
                 </tr>\n
                 <tr><td><b>Seiten-</b></td>\n
                     <td><input  type="checkbox"  name="pagtitl"  value="pagtitl"
onClick="javascript:refreshOptions(\'pagtitl\',\'@=ref@\');"></td><td>Titelblatt
</td>\n
```

```
                    <td><input  type="checkbox"  name="pagbauw"  value="pagbauw"
onClick="javascript:refreshOptions(\'pagbauw\',\'@=ref@\');"></td><td>Bauwerk</t
d>\n
                    <td><input  type="checkbox"  name="pagillu"  value="pagillu"
onClick="javascript:refreshOptions(\'pagillu\',\'@=ref@\');"></td><td>Illustrati
on</td>\n
                </tr>\n
                <tr><td><b>merkmale:</b></td>\n
                    <td><input  type="checkbox"  name="paginit"  value="paginit"
onClick="javascript:refreshOptions(\'paginit\',\'@=ref@\');"></td><td>Schmuckini
tiale</td>\n
                   <td><input  type="checkbox"  name="pagrund"  value="pagrund"
onClick="javascript:refreshOptions(\'pagrund\',\'@=ref@\');"></td><td>Rundplasti
k</td>\n
                    <td><input  type="checkbox"  name="pagnote"  value="pagnote"
onClick="javascript:refreshOptions(\'pagnote\',\'@=ref@\');"></td><td>Handschrif
t</td>\n
                </tr>\n<tr><td> </td>\n
                    <td><input  type="checkbox"  name="pagreli"  value="pagreli"
onClick="javascript:refreshOptions(\'pagreli\',\'@=ref@\');"></td><td>Relief</td
>\n
                    <td><input  type="checkbox"  name="pagrepr"  value="pagrepr"
onClick="javascript:refreshOptions(\'pagrepr\',\'@=ref@\');"></td><td>Reprodukti
on</td>\n
                    <td><input  type="checkbox"  name="pagreze"  value="pagreze"
onClick="javascript:refreshOptions(\'pagreze\',\'@=ref@\');"></td><td>
                    Rezeption</td>\n
                </tr>\n
                </tr>\n<tr><td> </td>\n
                    <td><input  type="checkbox"  name="pagreal"  value="pagreal"
onClick="javascript:refreshOptions(\'pagreal\',\'@=ref@\');"></td><td>Realien</t
d>\n
                    <td><input  type="checkbox"  name="paggrup"  value="paggrup"
onClick="javascript:refreshOptions(\'paggrup\',\'@=ref@\');"></td><td>Gruppen</t
d>\n
                    <td><input  type="checkbox"  name="pagsamm"  value="pagsamm"
onClick="javascript:refreshOptions(\'pagsamm\',\'@=ref@\');"></td><td>
                    Sammlung</td>\n
                </tr>\n
                     </tr>\n<tr><td> </td>\n
                   <td><input  type="checkbox"  name="pagtext"  value="pagtext"
onClick="javascript:refreshOptions(\'pagtext\',\'@=ref@\');"></td><td>Text</td>\
n
```

```
                    <td><input   type="checkbox"   name="pagleer"   value="pagleer"
onClick="javascript:refreshOptions(\'pagleer\',\'@=ref@\');"></td><td>leeres
Blatt</td>\n
                    <td><input   type="checkbox"   name="pagsarg"   value="pagsarg"
onClick="javascript:refreshOptions(\'pagsarg\',\'@=ref@\');"></td><td>Sarkophag<
/td>\n
                  </tr>\n
                  </tr>\n<tr><td> </td>\n
                   <td><input   type="checkbox"   name="pagbpla"   value="pagbpla"
onClick="javascript:refreshOptions(\'pagbpla\',\'@=ref@\');"></td><td>Bauplastik
</td>\n
                  </tr>\n
               </table>\n
               </form>\n
   <script language="JavaScript">\n
   <!--\n
   loadOptions();\n
   -->\n
   </script>'
exit name=textsorten
```

Erläuterungen zu Packages:

- In der Datei pack.1 werden nun die Packages für die Darstellung der Überschriften, der Checkboxbuttons, der Einbindung der Icons und der einzelnen Inhaltsebenen definiert. Im Folgenden wird ihre Funktionsweise Schritt für Schritt eingehender erklärt.

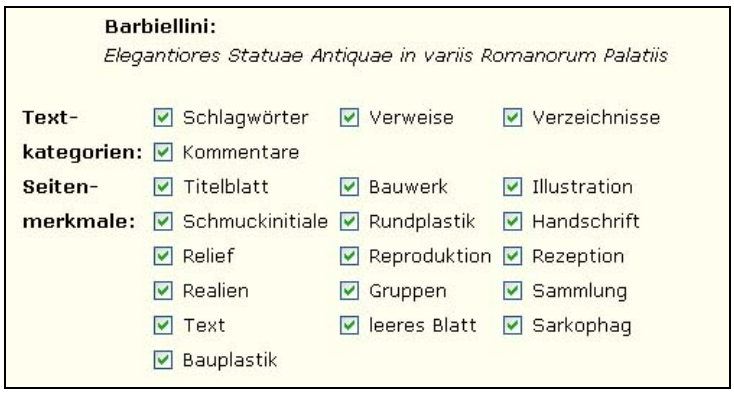


Abbildung 19: Auswahlmöglichkeiten beim ToC Menü.

Das zu Anfang aufgeführte Package tocwrap bedarf keiner Erklärung mehr, weil es den allgemeinen Seitenwrapper für die HTML-Seite generiert. Dies wurde schon zuvor des

öfteren aufgezeigt.

Besondere Aufmerksamkeit verdienen die Packages addline und addicon, die die Checkboxbuttons erzeugen.

Erklärung zu einzelnen Zeilen:

```
01 item name=addline;usage=package;overwrite=yes
02 brackets sensitive=yes
03 condition condition=':type="keyword"'
04 convert current=currenttext;
05        path='@(Trigcatkeyw@form["<br>Schlagw&ouml;rter:<i>"]
06        &+:dump[currenttext]&+:form["</i>"]|+
07        :package[/back[1]/image:extref,pageref]@)
08        Trigcatkeyw@@[Trigcatkeyw@:form[""]
09        @]Trigcatkeyw@'
10
11 condition condition=':type="reference"'
. . .
```

Aus dem Package addline wird hier beispielhaft nur die Vereinbarung für die Schlagwörter (“keyword“) herausgegriffen. In Zeile 03 wird der condition Befehl eingeführt. Durch condition kann man von verschiedenen Verpackungsanweisungen eine bestimmte Anweisung durch eine zuvor definierte Bedingung auswählen. Der Parameter condition erwartet einen Pfad mit Angabe einer Bedingung. Die Pfadangabe muss als Konstante in Hochkommata angegeben werden. Durch den Kleio Befehl convert wird weiter die Direktive Path ausgeführt. Mit Hilfe des Parameters current, bei dem generell ein beliebiger Name einer Variablen definiert wird, wird hier die Variable currenttext eingeführt. Der folgende Pfad (Zeile 05-09) ist durch Ersetzungsmarker, die das ansteuern/auslösen (triggern) von Verarbeitungsschritten aufzeigen, durchsetzt. Die Ausgabe der Schlagwörter wird durch den Ersetzungsmarker @=Trigcatkeyw@ gesteuert. Diese Trigger werden noch näher in der Datei fauzktoc.txt vorgestellt.

```
01 item name=addicon;usage=package;overwrite=yes
02 brackets sensitive=yes
03 . . .
04 condition condition=':property="bauwerk"'
05 convert
06        path='@(Trigpagbauw@form["<br><img border=\"0pt\"
07      src=\"@=AServer@buttons/bauwerk.gif\"
08      alt=\"Bauwerk\"
09        onMouseOver=\"window.status=\'Bauwerk\'\"
```

```
10      onMouseOut=\"window.status=\'\'\">"] |+
11      :package[/image:extref,pageref]@)
12      Trigpagbauw@@[Trigpagbauw@:form[""]@]Trigpagbauw@'
. . .
```

Aus dem Package addicon wird hier beispielhaft nur die Vereinbarung für das Bauwerk (“bauwerk“) herausgegriffen. Wenn per Standard der Checkboxbutton des Bauwerks aktiviert ist, dann wird im Script property=“bauwerk“ ausgeführt. Durch den Kleio Befehl convert wird weiter wie gehabt die Direktive Path ausgeführt.

Path enthält verschiedene Arten von Hochkommata und Anführungszeichen, die man im Folgenden so unterscheiden sollte:

a)

```
path='@(Trigpagbauw@form["<br><imgborder=\"0pt\"src=\"@=AServer@buttons/bauwerk.
gif\"alt=\"Bauwerk\"onMouseOver=\"window.status=\'Bauwerk\'\"onMouseOut=\"window
.status=\'\'\">"]|+package[/image:extref,pageref]@)Trigpagbauw@@[Trigpagbauw@:fo
rm[""]@]Trigpagbauw@'
```

Das erste einfache Anführungszeichen leitet eine Zeichenkette in Kleio ein. D.h. der gesamte Path Befehl wird als ein String definiert.

b)

```
path='@(Trigpagbauw@form["<br><imgborder=\"0pt\"src=\"@=AServer@buttons/bauwerk.
gif\"alt=\"Bauwerk\"onMouseOver=\"window.status=\'Bauwerk\'\"onMouseOut=\"window
.status=\'\'\">"]|+package[/image:extref,pageref]@)Trigpagbauw@@[Trigpagbauw@:fo
rm[""]@]Trigpagbauw@'
```

Nach der beginnenden eckigen Klammer folgt ein doppeltes Anführungszeichen, welches eine zweite Zeichenkette einleitet, die mit der schließenden eckigen Klammer endet. Der Inhalt der eckigen Klammern wird durch Maskierungen (\ Backslash) für die Ausgabe der HTML-Seiten sichtbar gemacht.

Wie in anderen Programmiersprachen auch, benützt Kleio Reguläre Ausdrücke, um Strings zu spezifizieren und/oder Vorschriften zu beschreiben, die Strings durch ihn abdecken.[13]

So besitzt auch hier das Metazeichen \ die Sonderbedeutung andere Zeichen

[13] Vgl. zu den Regulären Ausdrücken: Friedl: Reguläre Ausdrücke; ferner Ditchen: Shell-Script Programmierung, S. 362 ff. und Herold: awk & sed, S. 24 ff.

auszuschalten. Und weiterhin gibt es bei Kleio eine Reihe der üblichen Escape-Sequenzen:

\b Backspace, Zurücksetzzeichen

\f Formfeed, Seitenvorschub

\n Newline, Zeilenvorschub

\r Return, Wagenrücklauf

\t Tabulator

\v Vertikal-Tabulator.

Die path Codesequenz kann man außerdem noch in wahr und nicht wahr bzw. trifft zu/trifft nicht zu unterteilen.

```
path='@(Trigpagbauw@form["<br><imgborder=\"0pt\"src=\"@=AServer@buttons/bauwerk.
gif\"alt=\"Bauwerk\"onMouseOver=\"window.status=\'Bauwerk\'\"onMouseOut=\"window
.status=\'\'\">"]|+package[/image:extref,pageref]@)Trigpagbauw@@[Trigpagbauw@:fo
rm[""]@]Trigpagbauw@'
```

Der erste nicht markierte Abschnitt beinhaltet die Funktion form["
<img border=\"0pt\" src=\"@=AServer@buttons/bauwerk.gif\"alt=\"Bauwerk\" onMouseOver=\"window.status=\'Bauwerk\'\" onMouseOut=\"window.status=\'\'\">"] und die Packagevereinbarung |+package[/image:extref,pageref]. Wenn dieser erste Abschnitt jedoch nicht zutrifft, d.h. der Benutzer hat den Checkboxbutton wieder deaktiviert, dann ist die Direktive form[ ] ohne Inhalt belassen, so wie es beim markierten Abschnitt der Fall ist.

Die Packages von addicon sorgen dafür, dass ein Icon innerhalb des „Inhaltsverzeichnisses“ angezeigt wird. Sobald man mit dem Mousezeiger darüber fährt, erscheint auch noch ein alternierender Text. Durch die JavaScript Funktion onMouseOver/onMouseOut wird in der Statusleiste des Webbrowsers außerdem noch ein beschreibender Text zum Icon angezeigt.

```
item name=pageref;usage=package;overwrite=yes
brackets sensitive=yes
fields start=' <a href="@=ADynamic@pagesma/%22';
       first='%22"> (';
       limit=")</a>"
convert current=usesignature;
```

```
     path=':cgiencode[:dump[usesignature],url]'
convert usage=first;
     path=':nativeno';
exit name=pageref
```

Beim weiteren Aufruf des Packages von Bauwerk wird auf die Gruppe image, das Element extref und das oben angeführte Package pageref zugegriffen.

```
item name=headingin;usage=package;overwrite=yes
fields start='<table border="0pt" width="80%" align="center"><tr><td><b>';
       limit=': </b></td></tr>'
exit name=headingin

item name=headingout;usage=package;overwrite=yes
fields start='<tr><td><i>';
       limit='</i></td></tr></table>'
exit name=headingout
```

Damit die Überschrift und der Titel des Stichwerkes auch die gewünschte Positionierung und Formatierung erhält, werden noch die Packages headingin und headingout eingeführt. So wird der Name des Autors fett ausgedruckt (headingin) und der Titel des Werks (headingout) wird kursiv wiedergegeben.

```
item name=textsorten;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<form name=toccontrol>\n
              <table border="0pt">\n
                 <tr><td><b>Text-</b></td>\n
                     <td><input  type="checkbox"  name="catkeyw"  value="catkeyw"
onClick="javascript:refreshOptions(\'catkeyw\',\'@=ref@\');"></td><td>Schlagw&ou
ml;rter</td>\n
. . .
```

Das Package textsorten sorgt dafür, dass alle Beschriftungen der Checkboxes und die Checkboxbuttons selber in eine Tabelle gefasst werden.

5.7.4 Datei fauzktoc.txt

```
@(Trigcatkeyw@@>Trigcatkeyw@supply@@)Trigcatkeyw@
@(Trigcatrefe@@>Trigcatrefe@supply@@)Trigcatrefe@
@(Trigcattabl@@>Trigcattabl@supply@@)Trigcattabl@
@(Trigcatcomm@@>Trigcatcomm@supply@@)Trigcatcomm@
@(Trigpagtitl@@>Trigpagtitl@supply@@)Trigpagtitl@
@(Trigpagbauw@@>Trigpagbauw@supply@@)Trigpagbauw@
@(Trigpagillu@@>Trigpagillu@supply@@)Trigpagillu@
@(Trigpaginit@@>Trigpaginit@supply@@)Trigpaginit@
@(Trigpagrund@@>Trigpagrund@supply@@)Trigpagrund@
@(Trigpagnote@@>Trigpagnote@supply@@)Trigpagnote@
@(Trigpagbpla@@>Trigpagbpla@supply@@)Trigpagbpla@
@(Trigpagreli@@>Trigpagreli@supply@@)Trigpagreli@
@(Trigpagrepr@@>Trigpagrepr@supply@@)Trigpagrepr@
@(Trigpagreze@@>Trigpagreze@supply@@)Trigpagreze@
@(Trigpagreal@@>Trigpagreal@supply@@)Trigpagreal@
@(Trigpaggrup@@>Trigpaggrup@supply@@)Trigpaggrup@
@(Trigpagsamm@@>Trigpagsamm@supply@@)Trigpagsamm@
@(Trigpagtext@@>Trigpagtext@supply@@)Trigpagtext@
@(Trigpagleer@@>Trigpagleer@supply@@)Trigpagleer@
@(Trigpagsarg@@>Trigpagsarg@supply@@)Trigpagsarg@
@+ref@=#1@@>ref@supply@

query name=fauzk;part=/catalogue[fauzktoc,complete,"@#1@"]
write part=/back[1]:
     package[:author,headingin],:parcel[:titleproper,headingout],
          :package[:form[""],textsorten],
          :form["<h2>Inhalt:</h2>"],
          /back[1]/div:total[
            =groups,div,=invisible,=header,newlevel,
                    =always,
                    :head,"Textbeginn",
                    =packaged,
                    :head,headerline,
                 =groups,page,=noheaders,
                    =packaged,
                    :property,addicon,
                 =groups,text,=noheaders,
                    =packaged,
                    :text,addline
                ]
stop form=html;package=tocwrap
```

Erläuterung:

- In Zeile 01 bis 21 von fauzktoc.txt werden die Ersetzungsmarker der einzelnen Checkboxbuttons definiert. Für Trigcatkeyw (keywords) bedeutet dies:

```
01 @(Trigcatkeyw@@>Trigcatkeyw@supply@@)Trigcatkeyw@
```

Der innere Ausdruck @>Trigcatkeyw@supply@ wird nur verwendet, wenn überhaupt ein Ersetzungsmarker Trigcatkeyw vorhanden ist. @>Trigcatkeyw@supply@ ist dafür zuständig, dass in „ersetzungssensitive“ Packages Ersetzungsmarker exportiert werden.

- Bei der Datenbankabfrage in Zeile 23 wird der Katalog fauzktoc in Gang gesetzt. Der Katalog fauzktoc wurde in cat.2 zuvor definiert.

- In Zeile 24 wird per write Befehl die Ausgabe gestartet. Ein Katalog zeigt an, worauf sich der aktuelle Pfad bezieht. Später orientiert man sich bei der write Abfrage anhand dieses Pfades. Per back[ ] kann man nun jeweils zwischen den Pfaden springen. Die Pfade verweisen dabei auf die unterschiedlichen Hierarchien in der *.mod Datei. In unserem Fall wird per back Funktion ein Schritt zurück in der Hierarchie gegangen. Als Erstes wird nun ein Package ausgegeben, indem das Element author zusammen mit dem in pack.1 definiertem Package headingin verarbeitet wird (führt zur Ausgabe des Autorennamens in fett gedruckten Lettern mit anschließendem Doppelpunkt). Als Zweites verarbeitet die Funktion parcel das Element titleproper zusammen mit dem Package headingout (dies führt zur Ausgabe des Titels in kursiver Schrift).

- In Zeile 25 wird ein Package definiert, dass zusammen mit dem form Parameter das Package textsorten ausgibt (das Package textsorten enthält Angaben zu den Checkboxbuttons).

- Zeile 26 gibt durch den form Parameter die Zeichenkette „<h2>Inhalt:</h2>“ aus.

- In Zeile 27 wird wiederum ein Schritt zurück in der Hierarchie gegangen. Dort auf die Gruppenfunktion div, von der durch die Einbaufunktion total[ ] alles ausgegeben werden soll.

- In Zeile 28 bis 38 wird nun zusammen mit verschiedenen Verpackungsanweisungen und der in pack.1 definierten Packages die Ausgabe der ToC Dateien weiter spezifiziert.

5.7.5 Datei cat.2

```
query name=fauzk;part=/bibid:name="FAUzK"
catalogue name=fauzktoc;part=:value;overwrite=yes;
          type=term
```

Erläuterung:

- In der Datenbank fauzk wird ein Katalog mit dem Namen fauzktoc angelegt. Gesucht wird hier in den ToC Dateien nach der Gruppe bibid, dort noch spezieller nach dem Element name, welches die Zeichenkette “FAUzK“ beinhalten muss.

6. Erweiterte Konzepte: Mehrsprachigkeit

6.1.1 Einleitung

Grundsätzlich wurde das Verfahren der Mehrsprachigkeit beim Datenbankmanagementsystem Kleio 1999 entwickelt. Die Mehrsprachigkeit diente damals dazu, den Inhalt von Faktendatenbanken bzw. Datenbanken mit kontrolliertem Vokabular darzustellen. Es ging nicht wie heute, um Datenbanken, die essentiell natürlichsprachige Texte innerhalb von Webinterfaces präsentieren sollen.

Im Folgenden beziehen die Beispiele sich auf den Titelblatt Datenbankserver (Projekt: „Die Entstehung und Entwicklung des Titelblatts in der Inkunabel- und Frühdruckzeit“ vom Institut für Buchwissenschaft, Universität Erlangen-Nürnberg). Link: http://inkunabeln.ub.uni-koeln.de/titelblatt/

6.1.2 Grundlegende Annahmen

- Das System arbeitet zu jedem Zeitpunkt in einer Systemsprache. Per Standard ist das die Sprache, in der die Systemmitteilungen an den/die Benutzer/in geladen werden. Bei den Sprachabkürzungen gibt es derzeit einbuchstabige Codes (g=german, e=english). Auf Anwenderseite können beliebig viele andere Sprachen verwendet werden.

- In der Kommandosprache können Blöcke von Kommandos sprachsensitiv sein, also nur aktiviert werden, wenn eine bestimmte Systemsprache vorliegt. Dazu müssen die Sprachcodes im ersten Buchstaben eindeutig sein (s. oben), da später aber die Normsprachbezeichnungen unterstützt werden, wird jetzt schon empfohlen, längere Bezeichnungen zu verwenden.
- Für die symbolischen Namen in der Kommandosprache - Namen von Elementen, Gruppen und die der Objekte der 43 Objektklassen - können variable Beschriftungstabellen definiert werden, die explizit geladen werden können.

6.2 Umschalten der Systemsprache

Zum Testen kann mit dem Befehl
options language=english
eine bestimmte Systemsprache eingestellt werden (dabei wird nur das erste Zeichen der Sprachbezeichnung verwendet).

6.2.1. Protokollcode der kleioc-Aktivierung

Der Protokollcode der kleioc-Aktivierung wurde um ein optionales, nichtnumerisches erstes Zeichen ergänzt.
../kleioc/0010
wird angezeigt, wenn das System in der lokalen Sprache operiert,
../kleioc/e0010
wird angezeigt, wenn das System in der Sprache ’e’ operiert.

6.3 Sprachabhängige Symbole in Konstanten

Überall, wo Ersetzungsmarker eingesetzt werden können (also in Prozeduren und Packages) wird empfohlen, die sprachabhängigen Teile durch Ersetzungsmarker zu ersetzen und diese in der supply.ini Datei zu definieren. Laut der Basisstruktur eines Kleio Webservers, die folgende Unterteilung voraussetzt,

```
Software:
cgi-bin
cgi-bin/kleiob
```

```
cgi-bin/kleioc
cgi-bin/kl__.*
cgi-bin/*.ini

Datenbank(en)
database
database/fauzk.dat
database/fauzk.mod

Eigentliche Programme:
server
server/build.db
```

befindet sich die supply.ini Datei demnach in dem Software Verzeichnis unter cgi-bin.

Beispiel der supply.ini Datei für die Ersetzung der sprachabhängigen Teile durch Ersetzungsmarker in der Titelblatt Datenbank:

```
supply supply=LServer,"http://inkunabeln.ub.uni-koeln.de/titelblatt/",
            LCgi,"http://inkunabeln.ub.uni-koeln.de/tb-cgi/",
            LProt,"KlTB";
       type=both
#german
supply supply=SymSelect,"Ausw&auml;hlen",
            SymShowDescription,"Zeige Beschreibung";
       type=both
#/german
#english
supply supply=SymSelect,"Select",
            SymShowDescription,"show description";
       type=both
#/english
#spanish
supply supply=SymSelect,"Seleccionar",
            SymShowDescription,"Mostrar contexto";
       type=both
#/spanish
```

Die Sprachcodes g(erman), e(nglish) und (s)panish müssen unmittelbar auf ein in der ersten Spalte stehendes Nummernzeichen (#) folgen. Bei den Sprachcodes ist zu beachten, dass der erste Buchstabe dort stehen muss, der Rest kann mitgeteilt werden. Weiterhin können sie überall stehen, wo ein Kommando oder eine Direktive legal ist.

Im konkreten Beispiel:

Wird das System mit .../kleioc/0010... aktiviert, so verhält es sich, als ob die supply.ini Datei den folgenden Text enthielte:

```
supply supply=LServer,"http://inkunabeln.ub.uni-koeln.de/titelblatt/",
            LCgi,"http://inkunabeln.ub.uni-koeln.de/tb-cgi/",
            LProt,"KlTB";
      type=both
supply supply=SymSelect,"Ausw&auml;hlen",
            SymShowDescription,"Zeige Beschreibung";
      type=both
```

wird das System dagegen mit .../kleioc/e0010... aktiviert, so verhält es sich, als ob die supply.ini Datei den folgenden Text enthielte:

```
supply supply=LServer,"http://inkunabeln.ub.uni-koeln.de/titelblatt/",
            LCgi,"http://inkunabeln.ub.uni-koeln.de/tb-cgi/",
            LProt,"KlTB";
      type=both
supply supply=SymSelect,"Select",
            SymShowDescription,"show description";
      type=both
```

die solcherart definierten Marker können wie alle anderen Ersetzungsmarker verwendet werden. Hier ist das Beispiel der Datei pack.1 des Package tbsearch angeführt:

```
item name=tbsearch;usage=package;overwrite=yes
brackets sensitive=yes
fields origin='<center><FORM ACTION="@=LCgi@kleioc" METHOD="POST" name="basicselector">\n
 <input type="hidden" name="_kleioprot" value="@<Language@0010@=LProt@">\n
 <INPUT TYPE="hidden" NAME="_1" VALUE="ergebnis">';
       before='<INPUT TYPE="submit" VALUE="@=SymLast@" NAME="doprev" onClick="return
dynamicTarget(3)"><BR>\n
 <INPUT TYPE="hidden" NAME="prevterm" VALUE="';
      more="">\n';
      first='<SELECT NAME="execterm" SIZE="10">';
      second='</SELECT><BR>\n';
      after='<INPUT TYPE="submit" VALUE="@=SymNext@" NAME="donext" onClick="return
dynamicTarget(3)">\n
 <INPUT TYPE="hidden" NAME="nextterm" VALUE="';
      complete='<INPUT TYPE="submit" VALUE="@=SymShowDescription@" NAME="doexec"
onClick="return dynamicTarget(4)">\n
 </FORM></center>';
      start="<OPTION";
```

```
      write=' VALUE=';
      connect='>';
      limit="</OPTION>";
      supply=yes;
      preserve=yes
convert prepare=cleanforweb;
      usage=primary
exit name=tbsearch
```

6.4. Ändern der Systemsprache in der Website

Es wird empfohlen, durch den Sprachwähler der Site eine für alle Frames zugängliche Variable zu definieren (wie genau das Frameset für diesen Sprachwähler eingerichtet wird, wird bei den jeweiligen Projekten natürlich eigens variiert, das einfachste Sprachwähler-Beispiel wäre hierfür jedoch die Titelblatt Datenbank).

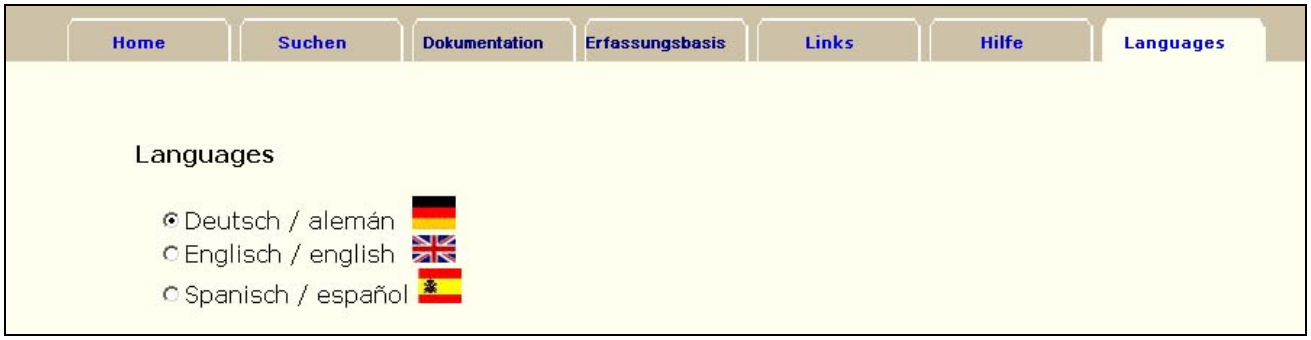


Abbildung 20: Sprachwähler des Titelblatt Servers.

Diese für alle Frames zugängliche Variable kann sich beispielsweise in einem invisible Form, ggf. auch in einem Cookie befinden.
Beim Titelblatt Server entspricht dem Sprachwähler der Abb. 1 folgende Codesequenz (Auswahl der englischen Sprache mit value=’e’) in der optionen.html:

```
<form …>
…
<input type="radio" name="sprache" value="en"
onClick=
"javascript:top.menue.location='../menue/menue-
opt_engl.html';top.body.location='optionen_engl.html';top.options.document.langSelect.currentLanguage.val
ue='e';">
Englisch /
```

```
 english  
<img src="../buttons/engl.gif">
</input>
<br>
...
</form>
```

Innerhalb der switch.html, die für den Steuerungsbereich der Site zuständig ist, wird das Dummyform definiert, welches zunächst auf ’g’ eingestellt ist.

```
<form name=”langSelect” ACTION="http://inkunabeln.ub.uni-koeln.de/tb-cgi/kleioc"
  METHOD="POST">
<input type="hidden" name="currentLanguage" value="g">
</form>
```

Die sicherste Methode, um zu garantieren, dass alle Sprachänderungen sofort aktuell werden, besteht darin, beim Aufruf das verwendete Protokoll dementsprechend dynamisch zu generieren, also z.B.:

```
 <form ACTION="@=LCgi@kleioc" METHOD="POST" target="MenuTB" name="basic
par">\n
<input type="hidden" name="_kleioprot" value="@<Language@0010@=LProt@"
 >\n
...
  limit='<input TYPE="submit" VALUE="@=SymSelect@" onClick="return
selectLanguage(\'basicpar\');"><br>\n
```

Diese Codesequenz ist aus der Datei pack.1 des Titelblatt Server, genauer aus dem Package init.

Dazu gibt es eine JavaScript Funktion selectLanguage (), die für Generierung der Sprachcodes verantwortlich ist:

```
 <script language="JavaScript" type="text/javascript">\n
          function selectLanguage(useform)\n
          {\n
          document.forms[useform].elements["_kleioprot"].value=\n
parent.parent.options.document.langSelect.currentLanguage.value+\n
             "0010@=LProt@";\n
          }\n
       </script>\n
```

Das komplette Package init in der pack.1 sieht also folgendermaßen aus:

```
item name=init;usage=package;overwrite=yes
brackets sensitive=yes
fields start='<html><head>\n
            <style type="text/css">\n
      </style>\n
      <link rel="stylesheet" type="text/css" href="css/body.css">\n
            <title>Willkommen</title>\n
      <script language="JavaScript" type="text/javascript">\n
         function selectLanguage(useform)\n
         {\n
         document.forms[useform].elements["_kleioprot"].value=\n
            parent.parent.options.document.langSelect.currentLanguage.value +\n
            "0010@=LProt@";\n
         }\n
      </script>\n
            </head>\n
 <BODY TEXT="#9C0000" BGCOLOR="ivory" LINK="#0000FF" VLINK="#000099">\n
 <base target="MenuTB">\n
 <basefont face="helvetica,arial,sans-serif">
 <font face="helvetica,arial,sans-serif" color="#660000" size="5">\n
 <center>\n
  <p>\n
 <strong>@=SymTitlepage@</strong></font><p>\n
 <font size="4" color="#8B0000"><strong>@=SymSimple@</strong></font><p>\n
 <font color="#8B0000">\n
 <strong>@=SymIndex@</strong></font>\n
        <form ACTION="@=LCgi@kleioc" METHOD="POST" target="MenuTB" name="basicpar">\n
        <input type="hidden" name="_kleioprot" value="@<Language@0010@=LProt@">\n
        <input type="hidden" name="_1" value="newcat">\n
        <input type="hidden" name="_kleioinherit" value="cat2use">';
 limit='<input TYPE="submit" VALUE="@=SymSelect@" onClick="return
selectLanguage(\'basicpar\');"><br>\n
      <font color="#8B0000">\n
      </form></center>\n
  <p><hr><p>\n
 </body> </html>\n'
exit name=init
```

Zusatz:

@<Language@

ist hierbei ein neuer Ersetzungsmarker zum Abfragen der Systemeigenschaften. Er steht für den einbuchstabigen Code der aktuellen Systemsprache.

6.5. Mehrsprachige Symbole

Es gibt seit längerer Zeit die Objektklasse „Symbol“, die bisher aber nur dazu diente, die Namen der Elemente und Gruppen mehrsprachig zu beschriften.

Diese Objektklasse kann jetzt auch verwendet werden, um für andere Objekte mehrsprachige Beschriftungen zu definieren. Empfehlung: Am besten die Symbol Vereinbarungen in der Datei definieren, die die beschrifteten Objekte generiert.
Für den Titelblatt Server lautet also die Symbol Vereinbarung in cat.1:

```
item name=gObjects;usage=symbol

type usage=catalogue
symbol symbol=personen;text="Personen";
symbol symbol=autor;text="Autoren";
symbol symbol=drucker;text="Drucker";
symbol symbol=worte;text="Worte";
symbol symbol=titelworte;text="Titelworte";
symbol symbol=istc;text="ISTC";
symbol symbol=gw-nr.;text="GW-Nr.";
symbol symbol=Hain_Copinger_Reichling;text="Hain/Copinger/Reichling";
symbol symbol=druckort;text="Druckort";
symbol symbol=jahr;text="Jahr";
symbol symbol=drucktypen;text="Drucktypen";
exit name=gObjects

item name=eObjects;usage=symbol
type usage=catalogue
symbol symbol=personen;text="Persons";
symbol symbol=autor;text="Authors";
symbol symbol=drucker;text="Printer";
symbol symbol=worte;text="Words";
symbol symbol=titelworte;text="Words in titles";
symbol symbol=istc;text="ISTC";
symbol symbol=gw-nr.;text="GW-Nr.";
symbol symbol=Hain_Copinger_Reichling;text="Hain/Copinger/Reichling";
symbol symbol=druckort;text="Place of printing"
```

```
symbol symbol=jahr;text="Year";
symbol symbol=drucktypen;text="Type of print";
exit name=eObjects

item name=sObjects;usage=symbol
type usage=catalogue
symbol symbol=personen;text="Persons";
symbol symbol=autor;text="Authors";
symbol symbol=drucker;text="Printer";
symbol symbol=worte;text="Words";
symbol symbol=titelworte;text="Words in titles";
symbol symbol=istc;text="ISTC";
symbol symbol=gw-nr.;text="GW-Nr.";
symbol symbol=Hain_Copinger_Reichling;text="Hain/Copinger/Reichling";
symbol symbol=druckort;text="Place of printing"
symbol symbol=jahr;text="Year";
symbol symbol=drucktypen;text="Type of print";
exit name=sObjects
```

Mit dem Befehl:

language symbol=symbolSet

kann dann eines dieser Sets von Beschriftungen ausgewählt werden.

Zum Aktivieren der Mehrsprachigkeit fungiert bei den Titelblättern die Prozedur init.txt:

```
language symbol=@<Language@Objects
describe usage=catalogue;write=cgi;package=init
```

7. Literaturverzeichnis

- Abeck, Sebastian; Lockemann, Peter C.; Schiller, Jochen; Seitz, Jochen: Verteilte Informationssysteme. Integration von Datenübertragungstechnik und Datenbanktechnik. Heidelberg 2003.
- Ditchen, Patrick: Shell-Skript Programmierung. Bonn 2003.
- Elmasri, Ramez; Navathe, Shamkant B.: Grundlagen von Datenbanksystemen. München 2002.
- Friedl, Jeffrey E. F.: Reguläre Ausdrücke. 2. Aufl., Köln 2003.
- Gross, Gabriele: Kleio. Eine Einführung in die Menüsteuerung. (Halbgraue Reihe zur Historischen Fachinformatik, Serie B, Bd. 10), St. Katharinen 1992.
- Grotum, Thomas: Das digitale Archiv. Aufbau und Auswertung einer Datenbank zur Geschichte des Konzentrationslagers Auschwitz. Frankfurt/New York 2004.
- Herold, Helmut: awk & sed. Die Profitools zur Dateibearbeitung und –editierung. 3., überarbeitete Aufl., München 2003.
- Heuer; Andreas; Saake, Gunter: Datenbanken: Konzepte und Sprachen. 2., akt. und erw. Aufl., Bonn 2000.
- Kemper, Alfons; Eickler, André: Datenbanksysteme. München 2001.
- Tanenbaum, Andrew; van Steen, Marten: Verteilte Systeme. Grundlagen und Paradigmen. München 2003.
- Thaller, Manfred: Kleio 4. Ein Datenbanksystem. (Halbgraue Reihe zur Historischen Fachinformatik, Serie B, Bd. 1), St. Katharinen 1992.
- Thaller, Manfred: Texts, databases, Kleio: a note on the architecture of computer systems for the humanities. In: Dino Buzzetti, Giuliano Pancaldi, Harold Short (Hg.): Augmenting Comprehension. Digital Tools and the History of Ideas. (Publication 17/Office for Humanities Communication). London 2004, S. 49-76.
- Thaller, Manfred: Reproduktion, Erschließung, Edition, Interpretation: Ihre Beziehungen in einer digitalen Welt. In: Mitteilungen des Instituts für Österreichische

Geschichtsforschung. Wien 2005. (im Druck)

- Vorländer, Martin: CGI – kurz&gut. 2. Aufl., Köln 2003.
- Woollard, Matthew; Denley, Peter: Source-Oriented Data Processing for Historians: A Tutorial for Kleio (Halbgraue Reihe zur historischen Fachinformatik, Serie A, Bd. 23), St. Katharinen 1993.

8.1. Referenzierte WWW-Seiten

Arachne Stichwerkebrowser am Forschungsarchiv für Antike Plastik, Universität zu Köln

http://134.95.113.230:16080/stichwerke/

Der Entwicklungsserver für die Stichwerke

http://134.95.113.230/stichtest/

Buchtitelblatt in der Inkunabel- und Frühdruckzeit am Institut für Buchwissenschaft, Universität Erlangen-Nürnberg

http://inkunabeln.ub.uni-koeln.de/titelblatt/

Hochstätter, Katrin: Die Rolle digitaler „Cultural Heritage Systeme“ als Bestandteil des kulturellen Gedächtnisses.

Als PDF: http://www.hki.uni-koeln.de/studium/MA/MA_hochstaetter.pdf

kleio – preliminary homepage

http://www.hki.uni-koeln.de/kleio/

Loebbecke, Claudia; Thaller, Manfred: Preserving Europe’s Cultural Heritage in the Digital World.

Als PDF: http://is.lse.ac.uk/asp/aspecis/20050000.pdf

Nemitz, Jürgen: Die neue Kleio Dokumentation

http://www.hki.uni-koeln.de/kleio/new/index.htm

Thaller, Manfred: Hybride und verteilte Datenbanken: Kleio. Hier: Netzwerkorientierte / „native XML“ Datenbanken

http://www.hki.uni-koeln.de/teach/ss04/U_Kleio/tag1/index.html

vdIb – Verteilte Digitale Inkunabelbibliothek

http://inkunabeln.ub.uni-koeln.de/vdib/

8.2. Anhang: Protokollcodes und Ersetzungsmarker

Ergänzungen zum Kleio Protokollcode:

```
Eine Kette aus vier oder fünf Ziffern, meist:
0010
optional (ohne Leerstelle) gefolgt von einer "Environmentwurzel", z.B:
0010KlKurs
Protokollnummer:
Null: Keine Benutzeridentifikation gewünscht.
Eins: Benutzeridentifikation gewünscht.
Command Echo Request:
Null: Die ausgeführten Befehle werden nicht gelistet.
Eins: Die ausgeführten Befehle werden elistet.
CGI Header Request:
Null: Es wird kein Standard CGI Header erzeugt.
Eins: Es wird Standard CGI Header erzeugt.
Argument Echo Request:
Null: Die übergebenen Argumente werden nicht gelistet.
Eins: Die übergebenen Argumente werden gelistet.
Header Type Request:
Null: Standard Texttyp Header wird zum frühest möglichen Zeitpunkt generiert.
Eins: Bildtyp Header wird generiert, sobald Bildtyp feststeht.
Environmentwurzel:
Serverdatenbanken werden in dem Directory erwartet, das in der hier angegebenen
Environmentvariable definiert ist.
```

Ergänzungen zu Kleio Ersetzungsmarkern:

```
Gegeben sei:
```

```
execute name=xxx;supply="ein-Wert",Variable,"noch-ein-Wert"

@#1@
==> ein-Wert

@=Variable@
==> noch-ein-Wert

@(xxx@ ... @)xxx@
... wird nur verwendet, wenn ein Marker "xxx" bekannt ist.

@?xxx@yyy@ ... @)xxx@
... wird nur verwendet, wenn der Marker "xxx" den Wert "yyy" hat.

@[xxx@ ... @]xxx@
... wird nur verwendet, wenn ein Marker "xxx" nicht bekannt ist.

Sowohl @?xxx@yyy@ ... @)xxx@ als auch @[xxx@ ... @]xxx@ akzeptieren durch
Fragezeichen eingegrenzte Modifier.

@??start?xxx@yyy@ ... @)xxx
... wird nur verwendet, wenn der Marker "xxx" mit dem Wert "yyy" beginnt.

@??limit?xxx@yyy@ ... @)xxx
... wird nur verwendet, wenn der Marker "xxx" mit dem Wert "yyy" endet.

@??password?xxx@yyy@ ... @)xxx
... wird nur verwendet, wenn der Marker "xxx" unter Anwendung
des Passwordverschlüsselungsalgorithmus den selben Wert wie "yyy" ergibt.


@_xxx@yyy@ ... @)xxx@
... wird nur verwendet, wenn der Marker "xxx" nicht den Wert "yyy" hat.
@+legalquery@:yes@
Existiert der Marker "legalquery" noch nicht, wird er
eingeführt. Er erhält in jedem Fall als Wert die
Zeichenkette "yes".

@+legalquery@=xxx@
Existiert der Marker "legalquery" noch nicht, wird er
eingeführt. Er erhält in jedem Fall als Wert den
des Markers "xxx".

@-legalquery@
Der Marker "legalquery" wird gelöscht.
```

@>xxx@...@
Der Wert des Markers "xxx" wird nach den Anweisungen "..." bearbeitet.
Dabei gilt:

@>xxx@clean@
Der Wert von "xxx" wird normalisiert; d.h., alle mehrfachen Leerstellen
werden zu einfachen.

@>xxx@convert,yyy@
Das "Conversion" Objekt "yyy" wird auf den Inhalt von "xxx" angewendet.

@>xxx@pack,yyy@
Das "Package" Objekt "yyy" wird auf den Inhalt von "xxx" angewendet.

@>xxx@expression,op1,string1,op2,string2 ...@
Der Inhalt von "xxx" wird nach den regeln der Pfadausdrücke
durch op1 string1, op2 string2 ... ergänzt.
Beispiel:
@>xxx@expression,+,_@
An den Inhalt von "xxx" wird '_' angehängt.

@>xxx@translate,yyy,zzz@
Ersetzt "xxx" durch den ersten Term im Thesaurus "yyy", der ihm durch die
Relation "zzz" verbunden ist. (*Nicht* Gegenstand dieses Kurses.)

@>xxx@substring,i,j@
Ersetzt "xxx" durch die an Position "i" beginnende, "j" Zeichen lange
Zeichenkette. (Fehlt j: Bis zum Ende.)

@>xxx@supply@
und
@>xxx@inherit@
beziehen sich zum Teil auf Mechanismen, die *nicht* Gegenstand dieses
Kurses sind.

@>xxx@supply@

ist jedoch insoweit wichtig, als damit in "ersetzungssensitive" packages
Ersetzungsmarker exportiert werden können.

@<user@
Dreiteiliger Username des derzeitigen Benutzers.

```
@<pass@
Password Username des derzeitigen Benutzers.

@<cgiuser@
Dreiteiliger Username des derzeitigen Benutzers, für CGI Transfer encoded.

@<cgipass@
Password Username des derzeitigen Benutzers, für CGI Transfer encoded.

@<name@
Einteiliger Username des derzeitigen Benutzers.

@<variable,xxx@
Derzeitiger Wert der lokalen Variablen "xxx".
```