Installationspaket: http://allegro-b.de/download/a35.zip

a35 : Plattformunabhängige Browser-Oberfläche für *allegro*-Datenbanken

Struktur, Arbeitsweise, Dateien

Die ab 2012 entwickelte Internet-Schnittstelle `a35` arbeitet mit den aktuellen Web-Standards:

HTML5 + JavaScript im Browser, PHP im Webserver und *acon +* allegro-FLEX im Datenserver.
Webserver und Datenserver kommunizieren per avanti über TCP/IP.

a35 bietet also für die Oberfläche das heute jedem Web-Entwickler vertraute Potential von HTML5 mit JavaScript.
a35 kann mehr als das ältere PHPAC, taugt also auch als Ablösung, besonders auf Mobilgeräten.
Ziel von a35 ist es aber auch, eine Web-Oberfläche für das *Arbeiten* mit einer Katalogdatenbank zu bieten. Zwar kann man es als Web-OPAC nutzen, aber von einem *Katalog* für Endnutzer wird heute mehr erwartet, als die Indextechnik von *allegro* leisten kann; als OPAC kann man auch VuFind oder andere Produkte nehmen, mit denen man die vielbeschworenen Qualitäten eines Discovery-Systems realisieren kann.

Zum Design-Konzept: Bei der Arbeit mit einer Datenbank gibt es vier Arten von Information, die oft alle gleichzeitig sichtbar sein sollten, weil sie logisch zusammenhängen:

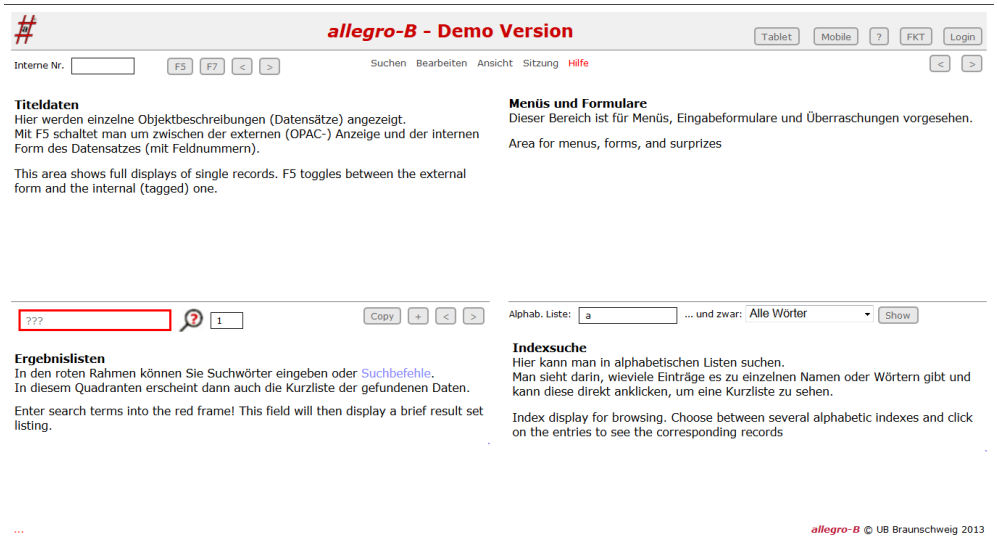
Datensatz : Metadaten zu einem Objekt (z.B. Buch, PDF-Datei, Website)
Ergebnisliste : geordnete Übersicht der Ergebnisse einer Suche
Register : geordnete Übersichten von Namen , Titeln, Schlagwörtern etc. zur Auswahl
Menüs und Formulare : Funktions- und Arbeitsdaten, auch Hilfetexte

Jeder der Bereiche braucht seine eigenen Schalt- und Eingabeelemente, die ihm jeweils auch optisch unmittelbar zugehören sollten. Z.B. die Eingabefelder für Suchbefehle bzw. Einstiegspunkte im Register sollten Bestandteile dieser Bereiche sein, ferner Buttons zum Vor- und Rückblättern.

Daraus ergab sich die Aufteilung der verfügbaren Fläche in vier Quadranten: (jeder hat ein internes, dreibuchstabiges Label, wie hier zu sehen)

1 `EXT` : Datensatz Umschaltbar : "extern"/ "intern" (F5)	**2 `INF` : Menüs, Formulare, u.a.** für jeweils gerade benötigte Aufgaben
3 `ERG` : Ergebnisliste mit Eingabefeld für Suchbefehle	**4 `REG` : Register** wie sie für allegro typisch sind.

Gleich nach dem Start sieht das so aus: (z.B. http://www.allegro-b.de/db/demo/a35-pc.php)

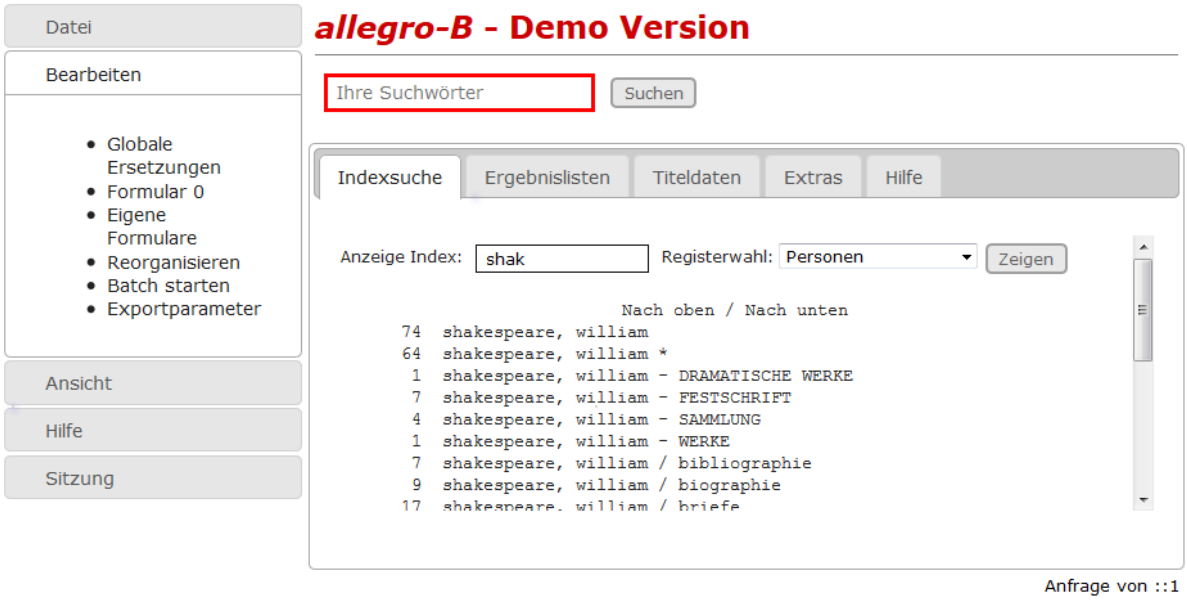


Die gesamte Struktur dieser Oberfläche steckt in der Datei `a35-pc.php`, in der man die Anordnung und alle Einzelheiten leicht modifizieren kann. Das Menü steht in `a35-pc-menu.php`, die übrigen sichtbaren Elemente in `a35-pc-cont.php`.

Als Alternative dazu gibt es `a35-tab.php`, das sich besser für Tablets eignet, und `a35-app.php`, mit dem Smartphones arbeiten können.

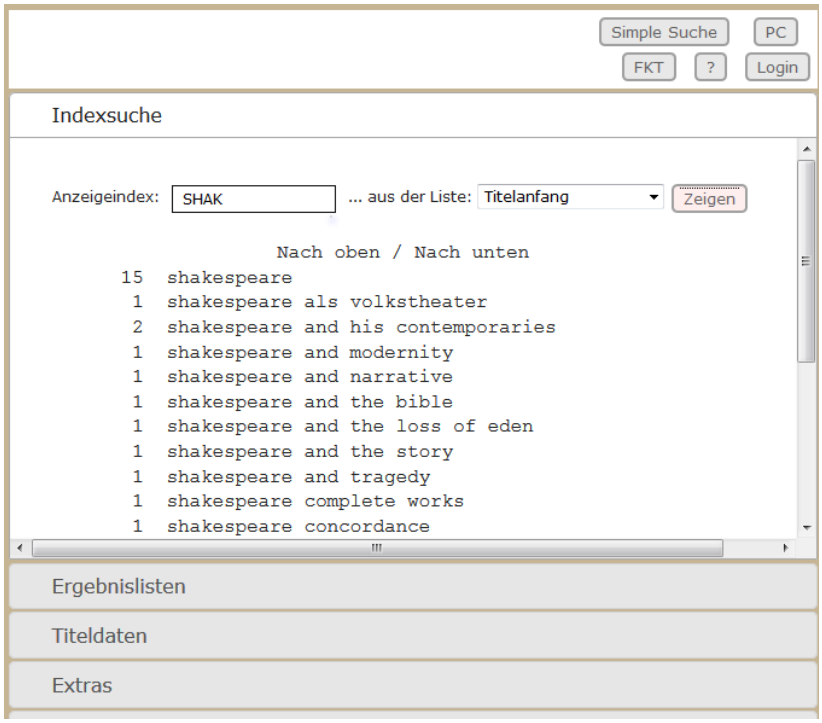
Hier die Tablet-Variante:

http://www.allegro-b.de/db/demo/a35-tab.php



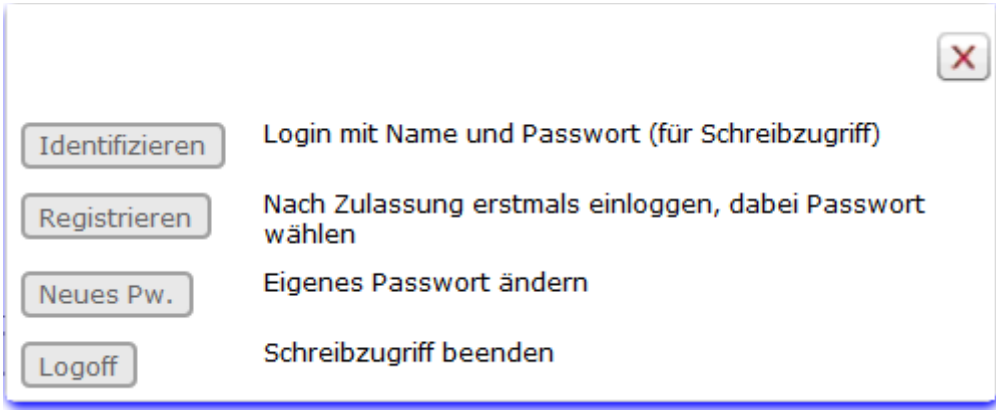
Und hier die für Smartphones:

http://www.allegro-b.de/db/demo/a35-app.php



Die rot umrandeten Felder sind gedacht zur Eingabe von Suchwörtern und Befehlen.
Tip: Mit **F2** obere und untere Hälfte vertauschen, mit **F12** die unteren wegblenden.

Für kurzzeitig wichtige Dinge gibt es ein Feld mit dem Label **FRE**, das bei Bedarf in der nötigen Größe mittig eingeblendet wird, z.B. das Login-Menü `a35login.htm`: (Aufruf: "Sitzung / Login-Menü")



Analog **FRE** gibt es **FRL** und **FRR**, zwei weitere Hilfsfenster, die links bzw. rechts unten erscheinen.

Warum so und nicht anders? Die Gestaltung des PC-Modells beruht auf folgenden Überlegungen:
In aller Regel ist das Endergebnis einer Suche ein einzelner Datensatz, d.h. eine Objektbeschreibung , denn gesucht wird ja zumeist nach ganz bestimmten, einzelnen Objekten, und jeder Datensatz beschreibt ein solches. Dafür braucht man ein Anzeigefeld. Es ist links oben, an der intuitiv i.a. wichtigsten Position.
Meistens findet eine Suche zuerst *mehrere* Objekte. Die Ergebnisliste für die Auswahl aus den gefundenen Objekten steht darunter, damit für das Auge der Weg von der Kurzangabe in der Liste zur vollständigen Angabe möglichst kurz ist, aber beides bequem im Blick bleiben kann. Das Register als Hilfsmittel zur Bildung von Ergebnismengen steht aus demselben Grund direkt rechts neben dem Ergebnislistenfeld. So bleibt der Bereich rechts oben übrig für Menüs und Formulare. Bei Bedarf eingeblendet wird ein Menüfenster unten rechts, z.B. für die Volltextsuche oder kombinierte Suche.
In den einzelnen Bereichen (außer REG) kann man, unabhängig voneinander, zu den vorher dort erschienenen Anzeigen zurückblättern (Buttons [<] und [>]).
Wichtig: Sowohl der Datensatz wie auch Hilfetexte und Formulare können recht lang werden, wobei man gern möglichst viel davon im Blick hätte. Daher sind diese zwei Quadranten oben angeordnet, und mit **F12** kann man die beiden unteren Quadranten schnell mal wegblenden, um mehr von den oberen Inhalten zu sehen. Mit **F2** dagegen kann man auch jederzeit oben und unten vertauschen, wenn man dies intuitiv besser findet. Dann ist also die Ergebnis-Kurzliste links oben, die Anzeige des ausgewählten Satzes darunter.
Bei jeder Größenänderung des Browserfensters passen sich die Quadranten automatisch an.
Die Anordnung und Gestaltung der Quadranten ist trotz alledem, da mit HTML5 + CSS realisiert, vom Anwender für eigene Anpassungen in `a35-pc-content.php` modifizierbar. Ferner könnte man an der linken und/oder rechten Seite noch Bereiche für Navigation o.a. einbauen.

Die PHP- und FLEX-Skripte für alle Varianten sind weitgehend dieselben, denn es handelt sich im wesentlichen um geänderte Präsentationsformen der Oberflächenelemente: die vier „Quadranten“ der Standardversion werden hierbei nicht gleichzeitig sichtbar, sondern unter Tabs bzw. in einem „Accordion“ angeordnet; man sieht dann jeweils nur den Inhalt *eines* Quadranten. Die Umschaltung erfolgt an vielen Stellen im Ablauf automatisch, ansonsten durch Mausklicks auf die Tabs.
Die vier "Quadranten" der PC-Variante sind hierbei inhaltlich unverändert unter den Tabs zu finden, wobei die Tablet-Variante noch ein zusätzliches Tab namens "Hilfe" hat, das Label ist HLP.

Technisches Konzept: a35 bietet einen Kernbestand von allgemeinen und universellen JavaScript- und PHP-Funktionen, die unverändert für jede Datenbank zum Einsatz kommen können. Spezifische Einstellungen werden in der Datei `ajax4ini.php` vorgenommen (weitgehend von `PHPAC`

übernommen). Einige wenige Export-Parameterdateien sind nötig; für die A-Konfiguration werden sie bereitgestellt (Dateityp `.apr`) und können modifiziert werden. Sie gehören aber in den Ordner der Datenbank, nicht zu den HTML-Dateien.

Neue Funktionen bindet man auf standardisierte Weise ein, wobei normalerweise keine Kenntnisse in JavaScript und PHP benötigt werden, FLEX-Jobs aber unentbehrlich sind – schließlich kann man anders gar nicht auf die Datenbank zugreifen. Mehr dazu im Anhang.
Auch die Standard-FLEX-Jobs für die Zugriffe zur Datenbank können geändert und erweitert werden, d.h. man muß auch die Standards nicht als unabänderlich hinnehmen.

Dateien des Standardumfangs (rot: datenbankspezifisch, Kommentare beachten.)
Zunächst diejenigen, die auf dem Startverzeichnis zu liegen haben (Typen `.php`und `.htm`)

`a35-pc.php` (bzw. die Varianten**`-tab`** für Tablet und **`-app`** für Mobil)

- kann zugleich als Beispiel einer Startdatei und Vorlage für eigene Gestaltungen dienen
- lädt aus **`a35-pc-menu.php`** das Menü und aus **`a35-pc-cont.php`** die **Oberfläche**, wie sie beim Start aussehen soll. Das Layout verändert sich während der Laufzeit nicht, sondern wird dynamisch mit Inhalten gefüllt (AJAX-Prinzip)
- lädt nach dem Start zuerst die Datei `a35start.htm`, die man beliebig ändern kann, um zu Beginn in den vier Quadranten sinnvolle Dinge erscheinen zu lassen, z.B. auch Bilder.
- zeigt an Beispielen, wie geeignete Links bzw. Formulare konstruiert sein müssen; diese rufen als `action` die JavaScript-Funktion `reqLoad()` oder `reqForm()` auf (in `a35.js`)
- Die Datenbank muß registriert sein in `avanti.con` (bei den Programmen avanti + acon)
- Datenbankspezifische Einstellungen in **`a35ini.php`**, das Menü in **`a35-*-menu.php`**
- **Formularelemente**, deren Inhalte an einen Job übergeben werden sollen, müssen je ein Attribut `id` der Form `id="Vuxy"`bzw. `id="Vnnn"` haben, `xy` beliebige Zeichen (daraus wird im Job dann die Variable `#uxy` bzw. das Datenfeld `#nnn`)
- Ferner: Mit `id="D`*`xyz`*`"` kann man Inhalte einbauen, die dann im Job in `$`*`xyz`* stehen
- **Oberflächenelemente**, in die per FLEX-Job etwas geschrieben werden soll, müssen ein `id` haben mit einem 3stelligen großbuchstabigen Label, z.B. `id="ABC"`. Die Funktion `receivE()` erledigt dann das Einfügen des auf `_!_ABC` folgendenT extes (beliebiges HTML) in dieses Element, d.h. man braucht sich darum nicht einzeln zu kümmern und kann sofort neue Labels einführen. Siehe unten Bemerkungen zu `a35erg.job` usw.
- Die Funktion `reqForm()` übergibt via `ajax4.php` an `acon` den Jobnamen mit ? `JOB=...` sowie die Variablen aus dem Formular mit `&Vuxy=...` bzw. `&D`*`xyz`*`=...`
- a35 enthält einige JavaScript-Funktionen, mit denen ein FLEX-Job aufgerufen und dessen Ergebnisse verarbeitet werden können, u.a. `receivE()`. In diesen Funktionen sind evtl. Eingriffe sinnvoll. Die universellen Funktionen sind ausgelagert in `../scripts/a35.js`.
- Allgemeines CSS für a35 ist gesammelt in `../scripts/a35css.php`. (!)
- Alle allgemeingültigen Jobs liegen in `../scripts/jobs`
- Benötigte Dateien aus der jQuery-Bibliothek mit zugehörigem CSS liegen auch dort

- Alle datenbankspezifischen Jobs liegen in einem Unterverzeichnis `$Jobdir`, das in `ajax4ini.php` anzugeben ist (s.u.)

`a35erg.job`, um ein Beispiel zu nehmen,

- wird wie jeder `.job` über das Skript `ajax4.php` gestartet und steckt auch hinter dem roten Eingabefeld; seine Aufgabe ist das Bilden und Anzeigen von Ergebnismengen.
- erhält vom `ajax4.php` die Variablen: aus `Vuxy` wird im Job `#uxy`, aus D*name* wird $*name,* d.h. im Job hat man diese Variablen ohne eigenes Zutun zur Verfügung, ferner `#uIP` mit der IP-Nummer des Browsers und `#uID` (codiertes Passwort) wenn der Nutzer sich vorher eingeloggt hat.
- produziert Output mit `write-` und `export`-Befehlen). Dieser Output ist eine lange Zeichenfolge, die durch Labels `_!_ABC` gegliedert ist.
- Ein solches Label ist **`_!_ERG`** und adressiert das Element `id="ERG"` im aufrufenden HTML-Code, also hier `a35-pc.php`: Der auf `_!_ERG` folgende Text kommt bis zur nächsten Markierung `_!_` in das betr. Element (Quadrant 3). Das erledigt die universelle Rückkehrfunktion `receivE()` in `a35.js`. Für eigene Erweiterungen kann man hier auch noch Sonderbehandlungen einbauen (im Abschnitt unter `switch(Label)`).

Weitere wichtige Jobs [Diese Sammlung wird weiter ausgebaut. Die mit den schwarzen Dateinamen liegen in `../scripts/jobs`, weil sie für alle Datenbanken funktionieren.]

`a35ind.job` zeigt einen Registerauszug in Quadrant 4 bzw. unter Tab 4 (Label `_!_REG`)
`a35get.job` besorgt einen Datensatz und zeigt ihn in Quadrant 1 (Labels`_!_EXT/_!_INT`)
(ACHTUNG: Hierin können**lokale Anpassungen** nötig sein für die Präsentation, Verlinkung z.B. zum WorldCat, Google Booksearch u.a.) Eingebaut ist auch, daß der angezeigte Satz zum Editieren oder als Kopie für einen neuen Satz bereitgestellt werden kann, oder auch zum Löschen. Alles unter Voraussetzung der Schreibberechtigung, s. `a35id.job`)
Unter dem Label `_!_INT` kann dieser Job zusätzlich eine andere Darstellung des Datensatzes liefern; normalerweise die interne, kategorisierte Form, die dann mit F5 im Quadranten 1 sichtbar wird.

`a35admin.htm` Ausbaufähiges Menü mit Admin-Funktionen, zum Aufruf einiger der folgenden:
`a35id.job` Login etc. (alle Funktionen, gestartet über `a35admin.htm`)
`a35gre.job` Globale Ersetzungen (analog zu a99, aber als Job für acon eingerichtet)
`a35findf.job/a35find.job` Experten-Suchmenü aufmachen / Suche ausführen
`presto.job` Aktuellen Datensatz zur Bearbeitung bereitstellen, Internformat
`form1.job` denselben Satz in einem selbstgestalteten Formular bereitstellen (nur als Beispiel)
`prsave.job` Speicherung des bearbeiteten Satzes (gilt für jedes solche Bearbeitungsformular)
`a35del.job` den aktuellen Satz aus der Datenbank löschen
`a35org.job` Index erneuern etc. ,wie in a99 (Eingeben: `h a35org.htm`)
`a35par.job` Eine Parameterdatei auswählen, bearbeiten und zurückspeichern
`a35bat.job` Eine Batchdatei ausführen lassen (Eingeben: `h a35bat.txt`)
`a35sperr.job` Den aktuellen Satz oder die Satztabelle sperren
`srugbv.job` Einen Fremddatensatz vom GBV per ISBN abrufen

`a35.js` liegt in `../scripts`

- enthält die universellen JavaScript-Funktionen, die man in jeder HTML5-Datei braucht. Eingriffe sind nicht nötig:
- `cReqObj()` : Ein "request object" anlegen (für die Ajax-Technik)

- `reqLoad()` : Aus einem Hyperlink einen Satz laden
- `reqRes()` : Eine Suchanfrage ausführen lassen
- `reqInd()` : Einen Registerabschnitt anzeigen lassen
- `reqForm()` : Aus einem Formular einen Job starten (s. a35-pc.php : `<form id=…`)
  (entnimmt alle V- und D-Variablen des Formularsund liefert sie an den Job)
- `receivE()` : Verarbeitet den Datenstrom, den eine AJAX-Anfrage liefert (also ein Job)

`jquery-min.js` notwendige jQuery-Funktionen, liegt auch in `../scripts`

`a35css.php` liegt auch in `../scripts` und enthält die CSS-Formatanweisungen

`ajax4.php + ajax4ini.php[+ a35alf.js ]` (das letzte nur für OPAC mit Ausleihe)

- ist ein universelles Skript, das selber nichts ausgibt; es
- startet einen Job, z.B. `a35get.job`, dessen Name ihm mit `?JOB=…` übergeben wird, und
- reicht die Variablen `Vuxy` bzw. D*name* an diesen Job weiter. Sie kommen im Job als `#uxy` bzw. $*name*an.
- Was der Job mit `write-` und `export`-Befehlen ausgibt, sendet nach seinem Ende `ajax4ini.php` als Ganzes weiter an den Browser, d.h. nur der Job produziert den Output, und zwar mit den Befehlen `write` und `export`.
- Der Output geht an die Funktion `receivE()` in `a35.js`
- In dieses Skript braucht man nicht einzugreifen. Nur in `ajax4ini.php` sind einige Angaben zur Datenbank nötig (dieselben wie bei PHPAC in `av_ini.php`).

Die Jobdateien sollen in einem Unterordner des HTML-Ordners der Datenbank liegen. Dessen Name muß ebenfalls in `ajax4.ini` stehen, z.B.

```
$Jobdir="abcjobs/";
```

sonst werden die Jobdateien direkt im HTML-Ordner gesucht. Nachteil: dort wären sie von außen sichtbar. Um sie gegen direkten Lesezugriff zu sichern, richtet man eine `.htaccess` mit entspr. Inhalt im `$Jobdir`-Ordner ein, desgl. im `../scripts`-Ordner.

Parameterdateien (evtl. spezif. f.d. Datenbank und deren CFG)
(im *Datenbank*ordner, z.B. `c:\allegro\demo2` , *oder* wo `acon` liegt, z.B. `c:\allegro`)

`d-html.apr` : Anzeigeparam.; Name in `ajax4ini.php` setzbar. Verwendet in `a35get.job`

`a35erg.apr/a35fts.apr` : Für die Ergebnis- bzw. Volltext-Kurzliste
(Verwendung in `a35erg.job/a35fts.job`)

`e-unihtm.apr, e-unicod.apr, utf.apr:` Umcodierungsparameter

`ad-utf.apt, ad-htm.apt, d-htm.apt, ucodes.apt` : Umcodier- und Hilfstabellen

Berechtigungen: Nutzer und Passwörter
zum Schreiben sind genauso organisiert wie bei a30. Im Menü „Sitzung“ findet man „Login-Menü“, damit wird die Datei `a35login.htm` geladen. Diese zeigt 4 Buttons: [Identifizieren],

[Registrieren], [Passwort ändern] und [Logout]. Jeder startet das Skript `a35id.job`, in dem alle diese Funktionen versammelt sind. Wichtig sind dann folgende Punkte:

- *Voraussetzung:* Der User, der `avanti` gestartet hat, besitzt Schreibrecht im Datenordner, und für die in `ajax4ini.php` mit `$ID=…` definierte Kennung besteht in `avanti.con` eine Berechtigung >0. Diese Kennung muß nichts zu tun haben mit den a35-Nutzern:
- An den Datenordner hängt man einen Ordner `users`
- Darin legt der Admin für jeden User xyz (frei wählbarer Name ohne Leerzeichen) eine Datei xyz an, in der zunächst nur *** steht. Soll der User Admin-Rechte haben, ist eine Zeile zu ergänzen, in der nur "`admin`" steht. Er kann damit per Browser neue User zulassen (s.u.) .
- Den Namen xyz teilt der Admin nur dem User mit, der im System xyz heißen soll
- Der Nutzer xyz kann sich dann sofort registrieren: „Sitzung / Login-Menü“, dann auf den Button [Registrieren] und die Eingaben tätigen. Dann wird *** ersetzt durch das eingegebene, aber codierte und nicht entschlüsselbare Passwort.
- Danach kann sich der Nutzer jederzeit mit dem Namen xyz und dem selbstgewählten Passwort identifizieren und erhält bis zum Sitzungsende Schreibrecht. Die "Sitzung" endet mit dem Abschalten des Browsers oder dem Ende des Tages oder der Funktion „Logout“, je nachdem, was zuerst kommt.

Besteht Schreibberechtigung, setzt `a35get.job` über die Datenanzeige drei Links: "Edit - Copy - Delete". Diese starten `presto.job` bzw. `a35del.job`. Der `presto.job` erzeugt ein Bearbeitungsformular im Feld `INF` . Das Formular hat nur ein einziges Eingabefeld, in dem der ganze Satz mit allen Feldern steht - wie beim alten Programm PRESTO. Darüber ein Button [Speichern], und das besorgt das Skript `prsave.job`. Analog zu `presto.job` kann es andere Skripte geben, um selbstgestaltete Formulare einzurichten, wobei alle Möglichkeiten von HTML5 ausnutzbar sind. Als Muster wird `form1.job` mitgeliefert, darin Kommentare zur Methodik.

Der Admin startet mit Eingabe von `h a35admin.htm` (in das rote Eingabefeld) ein Menü, auf dem eineFunktion [New User] ist. Die neue Userdatei im Ordner `users` wird dabei von `a35nwu.job` angelegt.

Grafikdateien [alle durch eigene austauschbar]
Standard: Nur `close.png, up.ico, down.ico` zum Schließen eines Hilfsfensters (`_!_FRE` und `_!_ FRR`) bzw. Blättern nach oben und unten im Index. Ferner `a10.png` ganz oben links.
Nur optional: `gbs.png, wcat.png.` Diese *kann* man in die Anzeige von Datensätzen einbauen, wie es in `a35get.job` zu sehen ist, um eine Verlinkung zur Google Buchsuche bzw. zum WorldCat bereitzustellen. Ansonsten können überall beliebige Grafikelemente zum Einsatz kommen, weil man alle Möglichkeiten von HTML nutzen kann. Eine Anzahl von `.gif`- und anderen Grafikdateien werden von den Varianten `a35-app.php` bzw. `a35-tab.php` verwendet.
Wenn man in a35ini.php die Varianten für „Corporate Design“ einschaltet (`a35_head_pc1.php` etc.), kann man mit eigene Grafiken das Erscheinungsbild individualisieren.

aLF : allegro-LeihFunktionen

Diese wurden aus PHPAC übernommen und für a35 angepaßt. Notwendige Dateien:

Konto, Verlängerungen[genau wie in PHPAC, dieselben Dateien]

a-okonto.htm : Aufruf vom Menü „Sitzung / Ausleihkonto“ oder aus eigenem HTML mit dem Link Konto
Die Daten erscheinen in einem externen Fenster. Das Script enthält die JavaScript-Funktionen zum Ausführen der Routinen, dieselben wie für PHPAC: insbes. req(X) zum Ausführen der Funktion X. Zuerst erscheint ein Dialog zum Eingeben von Nutzername + Passwort, dann kommt

a-okonto.php : Start von a-okonto.job, Präsentation im externen Fenster:

a-okonto.job : Erstellung des Kontos und Ausgabe ins externe Fenster, wobei z.B. der Link „Verlängern“ die Funktion req('v') auslöst.

a-overl.php : Start der Verlängerungsfunktionen
a-overl.job : Ausführung der Funktionen

Vormerkung

d-khtm.apr : Anzeigeparameter; Zeigt einen Link „Vormerken“, wenn alle Exemplare verliehen.

a35alf.js : Funktion vorm() wird bei Klick auf „Vormerken“ ausgelöst; Fragt Nutzername+Passwort ab und startet dann a-ovorm.job

a-ovorm.job : Führt die Vormerkung aus und liefert Rückmeldung an a35

Volltextsuche

Diese Methode wurde für a35 neu geschaffen.

a35fts.htm : Macht das Hilfsfenster FRR auf und zeigt darin den Dialog zum Eingeben des Suchbefehls, genauer: eines „Regulären Ausdrucks“. (Zusätzlich ein Button zum Auslösen der normalen ALL-Suche mit derselben Eingabe.)

a35fts.job : Erstellt a35fts.bat auf dem DbDir (!), das Skript (Batch bzw. Shell) zum Start des Programms srch mit den nötigen Optionen; es benötigt a35fts.apr und produziert damit (ebenfalls im DbDir) die Datei

fts.erg : Fertige Ergebnisliste mit Link für jeden Datensatz, der mit a35get.job den Satz ganz normal in a35 hineinholt. Anzeige der Liste im Quadranten ERG.

a35fts.apr : Parameter zum Erstellen der Kurzliste aus srch heraus, d.h. nicht aus einem Job.

fts.htm : Hilfetext zur Volltextsuche mit regulären Ausdrücken (kommt bei Klick auf [Help]).

ANHANG für Entwickler und Admins

"Interne Links" in a35: Konstruktion

Ein "interner Link" ist dazu da, Daten vom Server abzurufen (sog. AJAX-Technik), die dann im a35-Fenster an bestimmten Stellen erscheinen sollen. (Das Fenster selbst wird während einer "Sitzung" nicht neu geladen,sondern immer mit dieser Methodik intern mit neuen Daten gefüllt.)
Viele Beispiele in a35examp.htm (Aufruf per Menü "Hilfe / Link-Beispiele")

Der Server muß dazu diese Daten als Text mit Markierungen liefern,die jeweils einem Bereich, meistens einem `<div>`, in a35 entsprechen.Diese Markierungen haben immer die Form `_!_XYZ` , und direkt dahinter folgt HTML-Text, der dann in `<div id="XYZ">`landen soll.
Die nächste Markierung `_!_` beendet den Text. Ausgewertet und verarbeitet wird der vom Server kommende Text in der Funktion `receivE()` in `a35-pc.php`. Spezialbehandlungen für einzelne Markierungen oder eigene, neue `div`'s kann man hier zusätzlich einbauen.
(Wenn ausnahmsweise innerhalb eines *anzuzeigenden* Textes mal die Zeichenfolge `_!_` vorkommen soll, dann ist sie als `_!!_` zu schreiben.)

Vier allgemeine Fälle von "internen Links"

1. Eine **statische Datei** mit Markierungen, sagen wir `help.txt`
   (Statt `.txt` ist alles andere erlaubt, solange es eine HTML-Textdatei mit geeigneten Markierungen _!_XYZ ist). Diese Datei wird abgerufen mit einem Link dieser Form:
   `<a href="javascript:reqHelp("help.txt",mode);">Hilfe</a>`

   Hierbei ist *mode* entweder `"INF"`, wenn der Text in jedem Fall im Quadranten `INF` erscheinen soll, oder `"0"`, wenn der Ausgabetext eigene Markierungen mit _!_ enthält.

   Beispiele: [in den roten Rahmen eingeben: `h a35examp.htm`]
   ```
   <a href="javascript:reqHelp("a35admin.htm","INF");">Admin-Funktionen</a>
   <a href="javascript:reqHelp('a35kal.htm','0');">Kalender</a>
   ```

   Ferner die Dateien `a35start.htm` , die gleich nach dem Start geladen wird, und `a35login.htm`

   Test: Die Dateien kann man alle aus dem rot umrandeten Eingabefeld direkt abrufen mit `h a35admin.htm` bzw. `h a35kal.htm`, `h a35start.htm`, `h a35login.htm`.

2. Statt einer statischen Datei `xyz.txt` kann es genausogut ein **PHP-Aufruf sein mit einem oder mehreren Attributen**, z.B.
   `<a href="javascript:reqHelp('ajax4.php?JOB=kalend','0');">Dieser Monat</a>`
   So kann man statt `kalend.job` auch jeden anderen Job starten, der im Job-Ordner liegt und keine variablen Attribute braucht. Falls es im Beispiel ein ganz bestimmter Monat sein soll, kann man dies auch so machen:
   `<a href="...?JOB=kalend&Vuyr=1789&Vumo=9','0');">Sept. 1789</a>`
   In beiden Fällen kann man das Laden der Datei auch manuell auslösen. Z.B. indas rot umrandete Befehlsfeld folgendes eingeben:
   `h ajax4.php?JOB=kalend&Vuyr=1789&Vumo=9`

   Aber dann könnte ja jeder, der Bescheid weiß, mit diesem Trick jeden Job mit jedem Argument starten, z.B. `a35del.job` zum Löschen eines beliebigen Satzes? Theoretisch ja, praktisch aber nur, wenn er eingeloggt ist, also Schreibberechtigung hat. In jeden kritischen Job baut man dazu nur, meist am Anfang, diese Zeile ein: (z.B. in `a35del.job`)
   `perform authent`
   und hängt ganz unten den Abschnitt mit der authent-Routine an:(die Variable `#uID` liefert a35 nach korrektem Login stets mit.)
   ```
   :authent
   var #uId
   if "no" return
   var #dts(0,8) #uID(e"K")
   crypt
   ins #uid
   var #uID(e"K") "K" #uid
   ins #uid
   if #uid = #uID return
   wri "_!_POP Sorry, keine Berechtigung" n
   end
   ```

3. **Formulare** [Anzeige meistens im Quadranten `INF`, aber nicht zwingend]
   Hat man ein Formular konstruiert und soll beim Abschicken ein Job ausgeführt werden, der die ins Formular eingegebenen Daten zu verarbeiten hat, geht man so vor:

   ```
   <form id="eingabe" action="javascript:reqForm('verarb','eingabe');">
   ....</form>
   ```

   Aufzurufen ist damit der Job `verarb.job`. Die Funktion `reqForm()` in `a35.js` sammelt die Formularfelder ein, deren Namen mit V beginnen, und sendet sie allesamt an `ajax4ini.php`, welches dann den Job `verarb.job` startet und ihmjede Variable Vuxy als allegro-Variable `#uxy` überreicht und jede `Vnnn`-Variable als `#nnn`., z.B. V20 als #20.
   In `verarb.job` muß man also nichts tun, um sich diese Variablen erst zu besorgen, sie sind schon da. *Besonderheit:* Die Variablen `Vnnn`, also z.B. V20, liegen dann im "Objekt 2", d.h. man muß sie sich mit `set obj 2` besorgen. Dies ist so, damit man unbesorgt einen Satz laden kann, ohne daß die Variablen `Vnnn` dann überschrieben würden.
   Frage: Warum ist im Aufruf von `reqForm(...)` oben der Name des Formulars in der Klammer nochmals anzugeben? Weil dann ein Button oder Link zum Absenden auch außerhalb des Formulars plaziert werden kann. Z.B. kann man zwei oder mehr Buttons für mehr als ein Formular (was der Endnutzer gar nicht immer erkennen kann) in einer Reihe nebeneinander anordnen, alle außerhalb der Formulare, zu denen sie gehören. Jeder Button kann einen anderen Job starten und sich auf dasselbe oder ein anderes Formular beziehen. Nur kann man nicht die Daten von mehr als einem Formular zugleich absenden.

   Standardisierte Methodik für den Normalfall
   Beispiele: `presto.job` oder `form1.job` erzeugt ein Formular,
   `prsave.job` verarbeitet in beiden Fällen den Inhalt.
   `prtest.job` genauso, aber nur zwecks Anzeige, ohne Speichern.
   URL: `href="javascript:reqForm('presto','ext');"` bzw. mit `form1` statt presto

   Nach dem Muster von `form1.job` kann man sich beliebige Formulare bauen, die man alle mit dem gezeigten Aufruf verarbeiten kann, d.h. man braucht sich dann nicht bei jedem Formular wieder neue Gedanken zu dessen Verarbeitung zumachen.

   Tip zum Testen: in separatem Browserfenster eingeben
   `href=".../ajax4ini.php?JOB=form1&VurN=satznr;"`
   und dann die Quelldaten betrachten, dann sieht man, wie das aktive Formular aussieht. In gleicher Weise kann man meistens außerhalb von a35 prüfen, wie das Ergebnis eines Jobs wohl intern aussieht und ob es formal korrekt ist.

   Einfacher Fall: Es ist nur der Job auszuführen, ohne eine Variable zu übergeben:
   `javascript:reqForm('a35dbi','0');` führt `a35dbi.job` aus.

   ABER mitgeliefert an den Job werden in jedem Fall `#uRN` mit der aktuellen Satznummer und `#urS` mit dem letzten Suchbefehl, nach Login auch der Identifikator `#uID`.

4. **Externes Fenster aufmachen**, um z.B. einen Text außerhalb von a35 erscheinen zu lassen
   Form:`href="javascript:extWin(URL);"` // oder statt `URL`ein lokaler Dateiname
   Beispiel:`javascript:extWin('fts.htm');`

Fünf Spezialfälle für interne Links

Dies sind Links für Standardaufgaben, die in jeder Anwendung auf Basisvon a35 auftreten können.Diejenigen mit `req...` machen jeweils einen AJAX-Aufruf.Die gleichnamigen Funktionen stehen in `a35.js`.

1. `reqLoad`: Einen Datensatz holen und zeigen (erscheint im Quadranten EXT oben links)
   Form: `href="javascript:reqLoad(`*satznr*`);"`
   mit der internen Satznummer num. Beispiele dazu: in `a35ind.job` und `a35erg.job`
   Gestartet wird der Job `a35get.job` und ihm wird die *satznr* als `#urN` übergeben, ferner ein codierter Passwortstring als `#uID`, falls vorher ein Login erfolgte.

2. `reqRes`: Eine Ergebnismenge bilden und zeigen
   Form: `javascript:reqRes('PER shakespeare, william');`
   mit dem Suchbefehl in der Klammer.
   Beispiele dazu sieht man in den Zeilen jeder Ergebnisanzeige, produziert von `a35erg.job`

3. `reqInd`: Einen Registerabschnitt anzeigen
   Form: `javascript:reqInd('PER _shakespeare, william');`
   (Der _ bewirkt, daß beim Zugriff auf das Register die Umcodierung nicht ausgelöst wird.)
   Option: Vor den Befehl ein '–' setzen, dann wird von der gewünschten Stelle aus rückwärts gegangen, hier also ('-PER ...')
   Beispiele dazu in `a35ind.job` für die Links "Nach oben" und "Nach unten"

4. `reqEdit` : Einen Datensatz zum Bearbeiten bereitstellen (nur nach erfolgreichem Login)
   Form: `javascript:reqEdit('jobname','satznummer')`
   Damit wird ein Job aufgerufen, dem explizit die Satznummer zu übergeben ist
   Beispiele für `jobname: presto` oder `form1` oder `a35del` (s.o. Allgemeine Fälle 3.)
   Option: Ein ? vor `jobname` setzen, dann kommt zuerst eine Rückfrage zur Bestätigung, bevor der Job ausgeführt wird, z.B. `javascript:reqEdit('?a35del','5660');`

5. `delHist()`: Löscht die vorher angezeigten Sätze, Ergebnismengen und Hilfetexte, die während der Arbeit kopiert wurden und zum Blättern bereitliegen.
   Wird ausgelöst von dem Link "Historie löschen" im Admin-Menü

Test außerhalb von a35, um zu sehen, was der Server bzw. Job wirklich liefert. Es folgen einige Beispiele, die man leicht variieren kann:

Indexanzeige (Quadrant 4 = _!_REG):

`http://localhost/demo/ajax4ini.php?JOB=a35ind&VurS=per shakespeare`

Ergebnisanzeige (Quadrant 3 = _!_ERG):

`http://localhost/demo/ajax4ini.php?JOB=a35erg&VurS=per goethe?`

Einzelner Datensatz (Quadrant 1, über seine interne Nummer oder auch einen Find-Befehl):

`http://localhost/demo/ajax4ini.php?JOB=a35get&VurN=1234` bzw.
`http://localhost/demo/ajax4ini.php?JOB=a35get&VurF=PER+shakesp?`

Dann den Quelltext betrachten. Er wurde komplett von dem jeweiligen Job erstellt.Den Suchbefehl am Ende beliebig variieren und schauen, wie sich das auswirkt.

Der erste Teil ist immer gleich: `http://localhost/demo/ajax4ini.php?JOB=`
Dann kommt der Name der auszuführenden Jobdatei, hier `a35ind`, `a35erg` u.a.
Am Ende hinter & angehängt folgen die zu übergebenden Variablen, hier z.B. `VurS`; in der Jobdatei steht dieser Wert dann als `#urS` zur Verfügung.

Sonderfall in Formularen: Datumswahl

1. Man bindet in seinen HTML-Text ein (in dem das Formular steht):
   ```
   <script src="http://code.jquery.com/ui/1.10.0/jquery-ui.js"></script>
   ```

2. Man schreibt in dem HTML-Text, der dem Formular vorangeht:
   ```
   <script> $(function()
   {   $( "#Vudp" ).datepicker({ dateFormat: "yymmdd" });
   $( "#Vudp" ).datepicker();
   });</script>
   ```

3. Und dann im Formular das input-Feld:
   ```
   Datum: <input type="text" id="Vudp" />
   ```

Klickt man in dieses input-Feld, erscheint der Datumswähler. Hat man ein Datum ausgewählt, landet es in der Form `yymmdd` im input-Feld (z.B. `20130218`), und beim Abschicken des Formulars kommt es im Job an in der Variablen `#udp`.

Geschäftsgangsfunktionen: Aufruf mit `h a35order.htm`. Hierzu gehören einige spezifische Jobs, z.B. `a-exemp.job` und `o-bestel.job`, die zu den Datenstrukturen der Standard-Funktionspakete ORDER und aLF passen. Dieser Bereich ist nicht so voll entwickelt wie bei a99.

ALLGEMEINER TIP, wenn JavaScript involviert ist:

Strg+Shift+J: Fehlerkonsole zeigen lassen (beim FireFox)
JavaScript-Fehler können damit leicht und schnell gefunden werden.
Für avanti: Logging einschalten (in avanti.con, dann in der Logdatei nachschauen, wenn a35 ohne Fehlermeldung einfach nicht funktioniert.

2013-06-05/2014-02-17

Kurzfassung: Notwendige Schritte zur a35-Webanbindung einer Datenbank

Tip: Drucken Sie diese Liste aus und notieren Sie die Namen etc. für Ihre Datenbank bei den Punkten 0. - 7.
  * markiert die in jedem Fall zu beachtenden Dateien
  + Diese Dateien sind bei Bedarf zu ändern, für Grundfunktionen nicht nötig

Vier verschiedene Ordner sind zu berücksichtigen:
(Datenbankserver und Webserver können physisch dieselbe Maschine sein)

Datenbankserver

ProgDir : Hier liegen die Programme avanti und acon, sowie allgemeine Parameter
/jobs Unterordner (optional) für spezifische .job-Dateien

DbDir : Jede Datenbank liegt auf einem eigenen Verzeichnis mit beliebigem Namen und besteht aus mindestens folgenden Dateien: (z.B: DBN = cat und X=a)
DBN_n.Xld Datendateien (bis zu 255 Stück),
DBN.Xdx Indexdateien, enthalten die Zugriffsregister
DBN.tbl Satztabelle (Adressen der Datensätze)
DBN.stl Kurztiteldatei: eine Zeile je Datensatz für Übersichten
/users = Unterordner mit den Namen schreibberechtigter Nutzer

WebServer

WebDir : Am HTML-Ordner des Webservers, ein Unterordner für jede Datenbank, die ans Netz soll. Der Name ist beliebig
Inhalt: php und html-Dateien und einige andere (s. README)
/xxx Daran hängt ein Unterordner mit datenbankspezifischen Jobdateien

ScriptDir: Parallel zu den WebDirs, ein Ordner namens "scripts", mit einem
Hier liegen .js und .css
/jobs Unterordner mit allgemeingültigen Jobdateien

0. Beispiele für Verzeichnisnamen
a) DbDir d:\datenbanken\katalog mit Unterordner users
UNIX: /home/data/catalog
b) ProgDir c:\allegro mit Unterordner jobs (optional)
/home/allegro
c) WebDir c:\xampp\htdocs\db\katalog
/var/apache2/htdocs/db/katalog
d) ScriptDir c:\xampp\htdocs\db\scripts mit Unterordner jobs
/var/apache2/htdocs/db/scripts

Inhalte der Ordner und Dateien

1. ProgDir : a35 braucht nur 4 Dateien:
avanti[.exe] Der Datenbankserver; Läuft als Daemon (Unix) bzw. Dienst (Win)
acon[.exe] Führt die Jobs aus; Erledigt Zugriffe und liefert Ergebnisse
* avanti.con Liste der angebundenen Datenbanken mit Eigenschaften, s. 2.
uifsger Texte der Meldungen der Programme
Hier können auch weitere allegro-Programme liegen: srch, import, index, qrix ...
sowie unter Windows: a99 und alle anderen.

2. Eigenschaftsliste der Datenbanken
In avanti.con braucht jede anzubietende Datenbank einen Abschnitt,
der mit dem [symb.Namen] der Datenbank beginnt, z.B.:

```
[opac]     // das ist der symbolische Name der Datenbank, sie liegt hier:
directory = d:\datenbanken\katalog    oder    /home/data/catalog
access = 3          // d.h. Schreiben soll grundsätzlich möglich sein
konfiguration = a         // es wird $a.cfg benutzt
indexparameter = cat      // cat.api sind die Indexparameter
# username = password:rights
# Nur mit  rights>1  hat der user Schreibberechtigung
opac = OPAC:0
master = xyzabc:3
```

Hier stehen *nicht* die Namen der schreibberechtigten Endnutzer (s. 3. "users")
Der symbolische Name und die Angabe eines der users mit Passwort gehören
in die Datei ajax4ini.php, siehe 5.
Der username in avanti.con wird nur von acon gebraucht, damit dieses Programm weiß, ob Schreibzugriffe überhaupt zulässig sein sollen. Durch Einstellung access=0 kann man den gesamten Schreibbetrieb kurzfristig stoppen.

avanti entnimmt die Angaben aus ajax4ini.php
   und hängt dann an einen Job eine Zeile mit folgender Struktur an:
@ DB=opac ID=master/xyzabc

3. DbDir
Gebraucht wird eine .stl-Datei (Kurztitelregister)
  Dazu folgende Zeilen in den Indexparametern (s.4.):

```
i0=92     Länge der Kurzzeilen
ak=zz+0   Damit wird die Erzeugung einer Kurzzeile veranlaßt
...
#-0       Abschnitt f.d. Struktur der Kurzzeile
...         Befehle für die Erzeugung der einzelnen Teile der Zeile
#+#
```

Dann in a99 die Funktion "Kurzanzeige erneuern" (oder Index erneuern)

DbDir/users: für jeden schreibberechtigten Endnutzer "xyz" eine Datei
             mit dem Namen "xyz", sein codiertes Passwort enthält, das er
             sich selber gegeben hat (Funktion "Registrieren" in a35)

4. ParameterDateien und CFG
In ProgDir stehen die allgemeingültigen Parameter, spezifische im DbDir

```
X.cfg           Konfigurationsdatei (Default: $a.cfg)
                WICHTIG: Eintrag ce für letztes Änderungsdatum
dbn.Xpi         Indexparameter, z.B. dbn=cat, also cat.api.
      WICHTIG:  tucodes einbauen; evtl. #-1 ... für Umcodierung u. ic=1
X.cfl           nur falls Anzeige mit Feldbedeutungen gewünscht (cfga.job)
* d-html.Xpr    Param.f. Anzeige von Datensätzen, A-Format, nutzt ad-utf.Xpt
                WICHTIG: Abschnitt #-( f. interne Anzeige
* a35erg.Xpr    Param.f. Erstellung der Einträge in Ergebnislisten
+ a35fts.Xpr    entspr. f.d. Erstellung von Ergebnislisten der Volltextsuche
d-htm.Xpt       Codes f. HTML-Ausgabe (Kopie von d-htm.apt)
ad-utf.@pt      Zeichenwandlung  intern -> utf-8  f. Datensatzanzeige
e-unihtm.Xpr    Zwei allg.gültige Parameter f.d. Zeichenumcodierung
e-unicod.Xpr          aus dem internen Code in UTF-8, nutzt ad-utf.apt
```

5. WebDir : HTML und PHP Dateien, z.B. auf .../htdocs/db/catalog
            Je ein solches Verzeichnis f. jede anzubietende Datenbank

```
a35-pc.php          Startdatei für normalen Browser
 + a35-pc-menu.php       Datei mit dem Inhalt des Menüs

a35-tab.php         Endnutzer-Version, auch f. Tablets
 + a35-tab-cont.php      die sichtbaren Elemente

a35-app.php         Version f. Smartphones
                       Diese Version hat hat kein Menü.

a35-head-*.php      Zwei alternative Kopfdateien f. jedes der drei Modelle
                       Enthält das Material f.d. Seitenkopf
                       a35ini.php  enth. die Namen der zu benutzenden Dateien

* ajax4ini.php      Enth. den log. Namen, z.B. opac, der in avanti.con steht.
* a35ini.php        Einige spez. Setzungen f.d. Datenbank: Titel, Kopfdateinamen,
                          Liste der anzubietenden Suchregister

a35find.txt         Sehr einfaches Beisp., Aufzurufen mit "Simple search"
                         oder Eingabe:  h a35find.txt  im roten Eingabefeld
                         (Dateityp .txt or .htm ist beim h-Befehl unwichtig)
```

6. Unterhalb des WebDir
Spezif. Jobs f.d. Datenbank, z.B. auf .../htdocs/db/catalog/catjobs
(Der Name dieses Ordners muß in ajax4ini.php stehen)
Hinw.: Jeder Job wird zuerst hier gesucht, dann auf ../scripts/jobs
      Datenbankspezif. Versionen von allg. Jobs also hier unterbringen.

```
+ a35get.job    Der wichtigste spezifische Job, f.d. Anzeige eines Datensatzes.
+ form1.job     Formular-Editierung eines Datensatzes
```

7. ScriptDir : .../htdocs/db/scripts
Hierliegen allgemeingültige Dateien für alle anzubietenden Datenbanken

```
.js             Javascript
.css            CSS3
jobs/*.job      Allgemeingültige Jobs for a35. Spezifische s. 6.
```