iSeries für Einsteiger – Das Grundbuch

Autor: Holger Scherer

Inhalt: Eine Einführung in das Betriebssystem OS/400 und das System IBM AS/400 (auch genannt IBM eServer iSeries™)
Hinweis: Der Inhalt ist teilweise veraltet, dieses eBook wurde 2001 begonnen! Eine überarbeitete Version und Übersetzung ist in Arbeit. Prinzipiell sind die hier beschriebenen Konzepte aber weiterhin gültig.

Erscheinung: im Selbstverlag

Version: 08.03.2005

öffentlicher Server: http://ipublic.online (ja, kurz und schmerzlos)

Kontakt: mail: hs@RZKH.de http: www.RZKH.de

Inhaltsverzeichnis

Hinweise zum Dokument

Bezugsquelle

Dies ist die freie Ausgabe meines eBooks für AS/400-Einsteiger. Es ist (wie man schnell merkt) nicht mehr ganz aktuell, die erste Auflage wurde 2001 erstellt. Eine Überarbeitung für IBM i V7R4 ist in Arbeit, ebenso eine Übersetzung ins Englische (hier ist Unterstützung gern gesehen). Du darfst das Dokument gerne weiter geben mit Bitte um Verweis auf die Originalquelle.

Inhalt

Formatierungs- und Rechtschreibfehler in diesem Dokument sind beabsichtigt und dienen der allgemeinen Unterhaltung. Wer einen Rechtschreibfehler entdeckt, darf ihn behalten, ich hege kein Urheberrecht daran, eine kurze Meldung wäre aber nett. Hinweise zu inhaltlichen und sachlichen Fehlern nehme ich selbstverständlich auch gerne entgegen, ebenso wie Wünsche zum Inhalt, weiteren Kapiteln und Hinweisen auf das Dokumentenformat. Dieses Dokument wird ständig erweitert. Allerdings werden vorhandene Kapitel in den nächsten Updates nicht mehr signifikant umgebaut. Daher kannst Du es ruhig ausdrucken. Sollten vorhandene Teile geändert werden, wird es sich in Grenzen halten. Die meisten Leser bevorzugen allerdings das Lesen online (habe ich mir schreiben lassen).

Haftung

Alle in diesem Dokument enthaltenen Informationen wurden nach bestem Wissen zusammengestellt; jedes Programm wurde geprüft. Trotzdem kann keine Garantie auf Funktion gegeben werden. Sollten die Beispiele auf einem Produktivsystem nachvollzogen werden, garantiere ich nur, dass die Programme eventuell Speicherplatz benötigen. Es kann nicht ausgeschlossen werden, dass durch das Starten eines Beispielprogramms Dein Goldfisch Schluckauf bekommt, Deine Oma Dich enterbt oder die Festplatten sich plötzlich in die falsche Richtung drehen. Für Datenverlust und Systemausfall auf einem Produktivsystem wird keinerlei Haftung übernommen!

Ausserdem ist es möglich, dass durch zufälliges Zusammenstoßen des Dokuments mit einem Neutrino sich dieses Dokument in seine Einzelbestandteile zerlegt und danach unrettbar als kleine Atomarstruktur durch Zeit und Raum geistert, ohne Hoffnung auf Rettung! ;-)

Ach ja: Das Buch hat erst dann sein Ziel erreicht, wenn der Leser (also Du!) mir Vorschläge zur Verbesserung in meinen Beispielprogrammen oder Hinweise auf Fehler aufzeigt.

Trademarks

Die Bezeichnungen OS/400, AS/400, IBM, SEU, SDA, PDM, RPG, ILE RPG sind eingetragene Trademarks der IBM Corporation.

Die Bezeichnung „Microsoft Windows“ ist eine eingetragene Trademark der Microsoft Corporation.

Alle anderen genannten Warenzeichen sind im Besitz der entsprechenden Rechteinhaber, unabhängig einer in diesem Dokument verwendeten Nennung ohne explizite Angabe des selben. Die Anerkennung sämtlicher Inhaberrechte wird hiermit durch den Autor bestätigt.

Einleitung

Einiges vorweg

Dieser Grundkurs vermittelt einen ersten Eindruck sowie Grundlagen über die Arbeit mit dem System AS/400. Besonders ist er an Benutzer von Unix- und Windows-Systemen gerichtet. Dieses Buch ist kein Werbeblatt von IBM, sondern basiert auf eigenen und fremden Erfahrungen (auch wenn IBM etwas mehr Werbung für ihr bestes System betreiben könnte).

Grundkenntnisse in EDV-technischen Fragen werden ebenso vorausgesetzt wie Kenntnisse in SQL und Datenbanken. Ein wenig Verständnis für kaufmännische Programmierung (mein Berufsfeld und Hauptanwendungsbereich einer AS/400) kann auch nicht schaden. Erwünscht und gefördert sei das Verständnis für die Datenverarbeitung mit Blick auf Stabilität, Sicherheit und Datenintegrität auch bei grossen Datenmengen (ab diversen Terabyte), weniger für das Design einer bunten und Mausunterstützten Benutzeroberfläche mit vielen Fenstern.

Wenn man sich mit der AS/400[1] beschäftigen will, sollte man zunächst alles vergessen was man über Windows oder Unix weiss☺… Die Konzepte des SLIC, des MI und des OS/400 (die Teile, die zusammen das Betriebssystem sowie einige hardwareunabhängige Teile ergeben) stammen aus der Mitte der 70er Jahre, werden ständig weiter entwickelt und verändert. Sie haben eine ganz andere Ausrichtung. Das System enthält viele moderne Konzepte, die sonst in keinem anderen Rechnersystem realisiert wurden, auch wenn die Konzepte schon vor vielen Jahren entwickelt wurden.

Weiterhin ist die Software sehr unabhängig von der verwendeten Hardware, so dass Änderungen hieran keine Auswirkung auf die Funktion, Bedienung oder Installation der Software gegenüber dem Anwender haben, wohl aber auf die Performance und erweiterte Möglichkeiten. Die PC-Technik ist zwar ähnlich alt, folgt aber ganz anderen Ansätzen. Ein PC sollte ja früher nur ein etwas intelligenteres Terminal sein :-)

Eine Erläuterung der gesamten Systemarchitektur der AS/400 dürfte einige Bücher füllen, daher folgen hier nur ein paar grobe Beschreibungen. Kenner der Materie mögen mir die teilweise sehr vereinfachte Darstellung verzeihen.

Hauptaugenmerk lege ich auf Unterschiede zu Windows und Unix. Denn hier gibt es stellenweise sehr große Differenzen, die manchmal das Verständnis etwas behindern können, nach eingehender Betrachtung aber auch die Vorteile erkennen lassen. Damit soll dem Interessierten die Hemmschwelle genommen werden.

Ich wünsche viel Spaß beim Lesen und Nachvollziehen des hier Geschriebenen, und bin für jeden Hinweis per eMail an hs@rzkh.de dankbar!

Nun erst mal genug der Vorrede. Viele der hier besprochenen Sachen können auch online auf der öffentlichen IBM i (siehe http://ipublic.online) ausprobiert werden. Die meisten administrativen Angelegenheiten und Befehle sind natürlich aus Sicherheitsgründen nur mir vorbehalten. Wer das Glück hat, eine AS/400 sein Eigen zu nennen, kann natürlich auf eigene Gefahr experimentieren.

[1] In Deutschland und Österreich wird das System „*die* AS/400“ genannt, im Schweizer Sprachraum spricht man von „*das* AS/400“. Diese Verwendung ist linguistisch sinnvoller, da „AS“ mit Applikationssystem ausgesprochen wird. Der aktuelle Name ist „IBM i“.

Hinweis zur Verwendung von Formatierungen

Einzelne Tasten, die Du drücken musst, werden in Grossbuchstaben schwarz hinterlegt geschrieben, z.B.:
Die Taste F12 drücken.

Zeichen, die Du eintippen musst, werden in Schrift Comic geschrieben und sind grau hinterlegt, z.B.:
Rufe das Menü auf mit: GO GAST DF (danach drückst Du die Taste Datenfreigabe, wie DF andeutet)

Text, der Feldnamen oder einen Text auf dem Bildschirm kennzeichnet, ist in KAPITÄLCHEN gesetzt, z.B.:
Bitte springe mit dem Cursor zum Eingabefeld NACHRICHTENTEXT.

Text, der eine Ausgabe markiert, die Du beachten solltest, ist in Lucida geschrieben, z.B.:
Betrachte die Meldung „Datei TEST gelöscht.“

Dieses Buch ist bewusst in einem lockeren Stil geschrieben, um das Verstehen und Nachvollziehen zu erleichtern, sowie das Durchhaltevermögen des Lesers zu stärken. Wer sich durch das hier verwendete „Du“ gestört fühlt, möge es einfach durch „Sie“ ersetzen.

Viele Abbildungen erleichtern die Orientierung und unterstützen den eigenen Experimentiertrieb.

Wenn Du die hier gezeigten Beispiele auf Deiner oder einer anderen AS/400 nachvollziehen willst, kann es sein, dass auf Grund unterschiedlicher Betriebssystem-Versionen leichte Unterschiede in der Funktion oder der Bildschirmdarstellung auftreten, da selbst bei IBM der Fortschritt nicht Halt macht. (Geschrieben wurde dieses Buch hauptsächlich mit einem System unter V4R5 und teilweise mit einem System unter V3R2).

Dies gilt aber nur für die Entwicklungstools oder die Befehle. Ein Programm, das Du auf einem System mit V3R2M0 entwickelst, läuft zu 99,99(9)% auf einer höheren Systemversion. Umgekehrt funktioniert es nur, wenn Du nicht Befehle aus höheren System-Versionen verwendest. (Wenn man auf einem modernen System die Daten im Format für ein älteres System speichert, konvertiert OS/400 automatisch alle Objekte. Bei Problemen gibt es dann Hinweise). Allerdings ist dieses Speichern für ältere Versionen immer nur auf wenige Versionssprünge beschränkt.

Aus Zeitgründen konnte ich nicht alle Beispiele auf allen Systemvarianten prüfen; generell sollten aber alle auch noch mit V3R2M0 laufen. Da ich momentan kein älteres CISC-System zur Verfügung habe, möge man mir Probleme nachsehen (ein Hinweis per eMail wird aber erwartet).

Ein erster Einstieg

Das Konzept „AS/400“

Das Konzept der AS/400 ist eigentlich recht einfach, aber doch so anders. Zur Vereinfachung soll hier erklärt sein, dass „AS/400“ eigentlich eher ein Konzept von IBM zur Abstraktion eines Betriebssystems gegenüber der Hardware bezeichnet, hingegen ist „OS/400“ das entsprechende Betriebssystem, welches auf der von IBM zur Verfügung gestellten Hardware läuft[1]. Ziel der Entwicklung dieser Plattform war es, ein System für kommerzielle Anwendungen zu bauen. Diese zeichnen sich dadurch aus, dass sie viele Daten mit wenigen Änderungen verarbeiten zu haben, im Gegensatz zu technisch orientierten Systemen, die wenige Daten mit vielen Fließkommaberechnungen verarbeiten müssen.

Die Systemarchitektur baut auf folgende Grundeigenschaften auf:

- Objektbezogenheit
- Relationale, integrierte Datenbank
- Single-Level-Storage (Einstufiger Speicher)
- hoher Grad der Integration einzelner Komponenten in Software und Hardware
- Daten- und Programmunabhängigkeit
- Client/Server-Umgebungen

Hier eine Erklärung dieser Punkte:

Objektbezogenheit

Unter der Objektbezogenheit versteht man die wichtigste Grundlage des OS/400. Auf anderen Systemen versteht man unter einem Objekt meist nur ein ausführbares Programm, unter Unix eventuell die Objektdateien, die der Linker später zu einem ausführbaren Programm zusammenfügt. Die AS/400 bezeichnet (fast) jeden Softwareteil als Objekt, sei es ein Programm, eine Datei, ein Benutzerprofil, eine Jobbeschreibung oder ein am System angeschlossenes Gerät. Jedes Objekt hat bestimmte Eigenschaften und Methoden. Man kann ein Objekt nicht für einen anderen als den gedachten Zweck missbrauchen, da jeglicher Zugriff auf die Objekte auf der untersten Softwareebene geprüft wird.

Relationale, integrierte Datenbank

Die Datenbank der AS/400 ist in das Betriebssystem integriert und basiert auf dem relationalen Datenmodell. Das Datenbanksystem muss nicht – wie bei anderen Systemen – nachträglich hinzugekauft und installiert werden. Die Wartung und Pflege der Datenbank fügt sich in das Betreiben des Systems ein, und sie kann alle Vorzüge des Betriebssystems nutzen. Es ist keine eigene Verwaltung der Berechtigungen nötig, das Betriebssystem kann sich selbst um die Konsistenz der Datenbanken kümmern. Da die AS/400-Datenbank nicht nur auf SQL aufbaut, sind einige Begriffe für Teile einer Datenbank anders als in der SQL-Namensgebung. Diese Unterschiede werden uns aber nicht weiter stören.

Intern arbeitet das System viel mit SQL, oft aber auch mit eigenen „Denkweisen“. In diesem Buch wird aber zunächst auf die Arbeit mit DDS-Dateien (alte Form der Dateibeschreibung) eingegangen, damit Du als zukünftiger AS/400-Programmierer auch alte AS/400-Programme lesen kannst. Ein Teil mit SQL (im Client/Server-Betrieb sowie in RPG-Programmen) wird aber ebenso erscheinen.

[1] „Abstrahierung“ bezeichnet eine Methode, Software und Hardware zu trennen. Natürlich verwendet IBM den Namen „AS/400“ für die Hardware- und Software als Gesamtheit, inzwischen „eServer iSeries“ genannt. Prinzipiell ist es allerdings möglich, OS/400 auf einer anderen Hardware laufen zu lassen, wenn das gesamte Konzept der AS/400 auf eine andere Hardware portiert wird. (Man munkelte von einem OS/400 für OS/2 Anfang der 90er). Allerdings würde IBM sich dann das Hardwaregeschäft kaputt machen...

Einstufiger Speicher

Hierunter versteht man das einzigartige Speichermanagement der AS/400. Es handelt sich hier um eine besondere Variante des virtuellen Speichers. Im Gegensatz zu Systemen, die zwischen einer geringen Menge physikalisch vorhandenem RAM und dem größeren Festplattenplatz trennen, unterscheidet die AS/400 nicht zwischen dem Ort der Objektablage und gibt den Speicher transparent an den Benutzer weiter. Auf der AS/400 ist – einfach gesehen – der gesamte Festplattenplatz nur virtueller Speicher. Es gibt keine Trennung zwischen Festplattenplatz und RAM. Es gibt nur einen Adressraum, der RAM und alle Festplatten beinhaltet sowie möglichst durchgängig adressiert wird. Programme nutzen keinen eigenen (virtuellen) Speicherbereich, sondern adressieren direkt in diesem riesigen Adressraum, der beim ersten Bit im RAM anfängt und auf dem letzten Sektor der letzten Festplatte aufhört. Somit spart sich das System die sonst ständig nötige Umadressierung zwischen virtueller und realer Adresse. Systemprogrammierer auf der AS/400 merken das immer wieder, da fast alle Pointer zur Sicherheit gleich 128 Bit breit definiert sind.

Hoher Integrationsgrad

Einer der Hauptvorteile des Systems ist der hohe Integrationsgrad der einzelnen Systembestandteile. Viele andere Systeme bestehen aus einem Betriebssystem vom Hersteller X, einer Datenbanksoftware vom Hersteller Y, einem Sicherheitsmanagement des Unternehmens Z, dazu eine Storageverwaltung von der Firma V und einem Satz Entwicklungstools von E und L. Auf der AS/400 ist dies alles zusammengefasst und in einer einheitlichen Umgebung verschmolzen. Hier könnte Microsoft ins Träumen geraten, alles aus einer Hand, denn auf der AS/400 funktioniert es sogar. Man könnte von einer Abhängigkeit vom System und Hersteller sprechen, aber man sucht es sich ja freiwillig aus, und die Vorteile überwiegen hier.

Daten- und Programmunabhängigkeit

Dieses Merkmal war zur Vorstellung der Systeme in den achtziger Jahren eine Besonderheit. In den Anfangszeiten der kommerziellen EDV war die Struktur einer Datenbank im Programm hinterlegt. Änderte sich der Aufbau der Datenbank, so mussten alle beteiligten Programme angepasst werden. Man nannte dies „intern beschriebene Tabellen“.
Auf der AS/400 ist die Struktur einer Tabelle in dieser selbst hinterlegt. Öffnet ein Programm diese Tabelle, so wird zur Laufzeit die Struktur ausgelesen und dem Programm mitgeteilt. So kann die Reihenfolge der Felder in einer Tabelle geändert werden. Man spricht hier von „extern beschriebenen Tabellen“. Auch neue Felder sind möglich, ohne die Programme anzupassen, man muss sie nur compilieren (was man am besten in einem Rutsch erledigt). Ein Entfernen von Feldern bringt natürlich trotzdem Arbeit mit sich.

Client/Server

Client/Server-Computing ist seit über 15 Jahren ein Schlagwort der Branche, aber kein Computersystem konnte sich so sehr an die Anforderung dieser Techniken anpassen wie die AS/400. Ursprünglich als reines Host-System mit vielen Terminals entworfen, konnte durch Erweiterungen und neuen Systemsoftware-Teilen die Unterstützung von Clients wie PCs implementiert werden. Natürlich ist weiterhin parallel der klassische Terminal-Betrieb möglich, ebenso wie die Verwendung von Internet-basierten Anwendungen.

Details zum System

Die ersten Modelle

Als 1988 die AS/400 vorgestellt wurde, basierte die Hardware auf 48Bit-breiten Bipolar-Prozessoren. Diese sind mit recht wenigen MHz getaktet gewesen, aber recht flott für die damalige Zeit, da die Bipolartechnik nicht auf den Stromverbrauch und die Baugröße achten musste. Diese CPUs sind CISC-Modelle gewesen (Complex Instruction Set Code), ähnlich wie die PC-Prozessoren mit einem großen Befehlssatz ausgelegt. Heute gibt es für die AS/400 RISC (Reduced Instruction Set Code) Prozessoren, mit wenigen, aber dafür parallelisierbaren Befehlen.

Da die CPU-Leistung nicht alles in einem Mehrbenutzer-fähigen Datenbanksystem ist, packt IBM diverse Zusatzprozessoren (damals der Motorola 680x0-Klasse, heute PowerPC) für Bus-Steuerung, Speichergerätesteuerung etc. dazu. Die Systeme haben mehrere Bussysteme für Speichermedien, die mit eigenen Prozessoren ausgestattet sind. Somit konnte auf dem damaligen Einstiegsmodell mit wenigen MHz und 4-8MB RAM auch eine mittlere Firma mit mehreren Anwendern auf dem System arbeiten und programmieren. Der Leistungsindex für ein solches Gerät wurde auf 1.0 festgelegt, inzwischen ist mehr als das vierzigtausendfache möglich. Große Systeme bringen es auf über 70.000 Benutzer, sechsstellige Benutzerzahlen gehen auch, wohlgemerkt auf einem einzigen Rechner.

Diese Hardware war auf Stabilität, Dauerhaftigkeit und Erweiterbarkeit ausgelegt, nicht unbedingt auf Geschwindigkeit. IBM wollte dem Kunden damit den Investitionsschutz näher bringen, so dass einmal angeschaffte oder geschriebene Anwendungs-Software auf Jahre hinaus auf der jeweils aktuellsten Hardware laufen kann. Die CPU-Leistung der Systeme ist heute noch relativ moderat, es gibt einen Seti@home-Client[I] für OS/400, der jedoch keine Spitzenleistungen bringt. Die Stärke der AS/400 ist die Datenverarbeitung im großen Stil, weniger in der Berechnung von Fliesskomma-Operationen.

Die Stärken

Dieses Prinzip gilt auch heute noch, wichtiger als das Erreichen eines Benchmark-Höchstwerts ist IBM wohl eher, dass die Systeme Jahrelang im Dauerbetrieb laufen ohne abzustürzen oder wegen eines umgekippten Bits auf der Strecke zu bleiben. Zum Thema Absturz lässt sich sagen, dass kaum ein anderes System auf dem Markt so stabil läuft. Klar, zu 100% sicher ist es nicht, es gibt Berichte über Fehler im System, die den Dauerbetrieb unterbrechen. So etwas kommt (wenn überhaupt) im Schnitt einmal alle paar Jahre vor, und der Grossteil der Anwender hatten noch nie ein solches Problem. Hingegen kennt wohl jeder die berüchtigten „bluescreens“ unter Windows. Ich selbst habe aber noch keinen Kunden gehabt, der einen Systemausfall hatte. Wenn ein Leser hierzu eine Geschichte zu erzählen hat, ich habe immer ein Ohr offen :-)

Ein nicht unbedeutender Automobilhersteller in Deutschland hat 2002 sein Mail- und Dokumentenverwaltungssystem (neudeutsch: Collaboration Software) auf eine (gespiegelte) AS/400 umgestellt, die weltweit ca. 70.000 Benutzer betreut. Das Adäquat an Windows-Server-Farm möchte ich gerne sehen. (Bei besagtem Autobauer waren es ca. 230 PC-Server weltweit, diese wurden abgelöst).

[I] Siehe http://setiathome.ssl.berkeley.edu/ Du benötigst das Lizenzprogramm PASE zur Ausführung!

Bis heute ist die AS/400 wohl das einzige System außer sündhaft teuren Großrechnern, das man nach dem Einschalten vergessen kann. Abstürze mit darauf nötigem Herunterfahren des Systems gibt es nur sehr, sehr selten (ein Leser dieses Manuals schrieb von einem Fehler im TCP/IP-Stack, der zu einem System-Dump führte, aber IBM hat doch Abhilfe schaffen können); im Durchschnitt ist OS/400 das einzige Betriebssystem, dass eine Verfügbarkeit von 99,99x% wirklich erreicht, und somit im Jahr nur für ein paar Minuten nicht verfügbar ist. (Keine Rückschlüsse von meiner AS/400 auf das System generell ziehen, ich baue aus Spaß gelegentlich einfach mal eine andere Platte zum Testen ein, da muss mein System runtergefahren werden).

Sicherlich kann durch Programmierfehler ein Programm nicht immer wie gewünscht reagieren, aber dadurch wird niemals die Stabilität des gesamten Systems in Mitleidenschaft gezogen. Der Autor arbeitete im Jahr 2002 lange bei einem Kunden mit einer AS/400 Modell 823. Dieses Gerät von der Größe eines Kühlschranks (mit besserem Design, aber schlechteren Kühleigenschaften) hat 16GB RAM, 12CPUs und ca. 2 Terabyte Festplatten. Dieses System ist in zwei logische Partitionen (also voneinander unabhängig laufende logische Rechner) aufgeteilt. Auf der einen Partition arbeiten ca. 5.000 Anwender mit produktiven Daten und Programmen, auf der anderen Partition sind ca. 40 Entwickler mit der Programmierung und mit SAP-Umstellungen beschäftigt, und wir schonten das System nicht wirklich ;-) Die Kiste ist nur einmal ausgefallen, aber nur, weil jemand (warum auch immer) im angeschlossenen Storage-System die AS/400 Platten im laufenden Betrieb formatiert hat. Das überlebt kein Rechner...

Die Schwächen

Nachteile gibt es an sich nicht wirklich viele auf der AS/400. Für kurzfristig denkende Leute (meist auf der Entscheider-Ebene) mag der recht hohe Preis der Hardware ein Strich auf der Negativliste darstellen. Sicherlich kostet ein System AS/400 inklusive Betriebssystem und Wartung eine Menge. Eine Windows-Büchse mit billiger Hardware eines PC-Herstellers und Linux-System ist auf den ersten Blick sicherlich um einen beträchtlichen Betrag günstiger.

Kalkuliert man jedoch längerfristig, z.B. auf drei bis fünf Jahre, kehrt sich die Rechnung um. Eine AS/400 ist im Betrieb fast immer weitaus günstiger als andere Systeme. Dies liegt zum Einen am problemlosen Betrieb ohne Absturz, zum anderen daran, dass für die Pflege einer AS/400-Umgebung wenig Personal nötig ist. In kleineren und mittleren Betrieben kümmert sich meist ein Anwender um alle AS/400-Belange, während allein für das PC-Netzwerk mit 200 Benutzern und den 10 nötigen Windows-Servern schnell 4-5 Mann nötig sein können. Ausserdem kann man auf der AS/400 alles selbst machen. Es gibt Kleinbetriebe mit einer AS/400-Anwendung, die keinen AS/400-Fachmann haben und nur bei Bedarf einen externen Programmierer anrufen, mit der Zeit aber viele administrative Dinge selbst tun können. Das System ist in der Hinsicht auch recht Narrensicher.

Weiterhin von Nachteil ist die auf den ersten Blick geringe Verbreitung. Schätzungsweise 100.000 Systeme sind in Deutschland installiert, wie viele Millionen Windows- und Unix-Systeme installiert sind, kann man nicht genau sagen. So gering ist die Verbreitung allerdings nicht, bedenkt man die Anzahl an Benutzern, die eine AS/400 bedient.

Letztlich sei gesagt, dass es für den Entscheider oft recht schwierig erscheint, qualifiziertes AS/400-Personal aufzutreiben. Mit PCs kennt sich jeder aus, die Modesprache Java oder Visual Basic ist auch recht geläufig. Aber RPG und OS/400 ruft (leider inzwischen auch bei sehr vielen Uni-Absolventen) nur noch ein Fragezeichen auf die Stirn. Gegen diese Unkenntnis soll dieses Buch etwas Hilfe leisten. (Und Java geht auf der AS/400 natürlich auch.)

Die Hardware

Die Hardware der AS/400 hoppla eServer iSeries™ ist heutzutage recht konventionell. Der RAM-Speicher wird mit 10Bit Breite angesprochen. Neben den 8 Datenbits gibt es selbstverständlich das 9. Bit für die Fehlerkorrektur. Das 10. Bit wird (vereinfacht ausgedrückt) für Sicherheitszwecke verwendet. Es kann nur von der CPU in bestimmten Betriebssystemroutinen gesetzt werden. Sollte es also einem Hacker oder einem Amok laufenden Programm (beides recht unwahrscheinlich) gelingen, einfach einen Speicherbereich ohne Berechtigung zu verändern, wird er nie diese Prozessorbefehle benutzen können. Somit kann dieses 10. Bit auch nicht gesetzt werden (dies ist nur dem Softwareteil zwischen Betriebssystem und Hardware möglich). Beim nächsten Zugriff bemerkt die CPU, dass dieses Bit fehlt und erklärt die Speicherzelle sowie alle dazu gehörigen Datenbereiche für ungültig. Manche sprechen bei den verwendeten PowerPC-CPUs auch vom so genannten 65-Bit-Prozessor[I]. Stimmt zwar nicht ganz, verdeutlicht aber die Funktion des „Tag“-Bits, welches den Schutz der Pointer ermöglicht.

Die AS/400 unterscheidet nicht zwischen Festplatten und RAM. (Natürlich tut sie dies aus Gründen der Performance, aber nicht in der Speicherverwaltung). OS/400 unterscheidet bei der Ablage der Objekte (alles im System ist ein Objekt, sei es eine Datei, ein Benutzerprofil oder ein Gerät; ähnlich wie bei Unix, allerdings prüft Unix nicht objektbasierend den Zugriff) nicht, wo es abgelegt wird, da das Betriebssystem nur die Objekte nach ihrem Namen kennt, nicht den Speicherort.

Dieses so genannte *einstufige Speicherkonzept* bringt diverse Vorteile bei der Erweiterbarkeit. Der gesamte verfügbare Speicher wie RAM und alle Festplatten wird in einem einzigen Adressraum verwaltet, wobei die Obergrenze durch eine 64Bit-Adresse definiert wird. Tatsächlich sind die verwendeten Pointer 128Bit groß, von denen derzeit 64Bit für die Adresse, einige für Objektbeschreibungen sowie andere für Reservezwecke verwendet werden. Sollte der Speicher voll werden (alle Festplatten belegt), reicht es, im laufenden Betrieb dem System Festplatten hinzuzufügen und dem Betriebssystem dies mitzuteilen. Alternativ kann man auch nicht benötigte Objekte löschen. Schon ist mehr Speicher für neue oder größere Objekte verfügbar. Ebenso kann eine Festplatte entfernt und gegen eine größere Platte ausgetauscht werden.

OS/400 kennt keine Laufwerke wie Windows, es kennt nur die Objekte, mit denen es arbeitet. Ein System mit 1GB RAM und 40GB Festplatten hat somit einfach 41GB Speicher, in dem Objekte aller Art abgelegt werden. Der Benutzer entscheidet nicht, wo das Objekt abgelegt wird, dies ist Sache des unter dem Betriebssystem liegenden Microcode.

Man kann sich das Ganze ungefähr wie einen PC vorstellen, der 1GB RAM und 40GB Festplatte hat, und die Festplatte wird nur für die Auslagerungsdatei verwendet. Bei einem Neustart wird diese Auslagerungsdatei nicht geleert, sondern nur erneut geöffnet und weiterverwendet, der Inhalt vor dem Herunterfahren ist noch vorhanden. Dateien werden nicht in einem Dateisystem abgelegt, sondern in dieser Auslagerungsdatei. Ach ja, eigentlich gibt es gar keine Dateien, sondern nur Objekte. Davon haben einige das Attribut „Datei“. Das RAM kann man sich somit wie einen Festplattencache vorstellen.

Noch mal zur Verdeutlichung: Das Betriebssystem hat keine Ahnung über RAM, Festplatten und Speicher, es kennt nur Objekte. Die Adressierung wird vom MI (Machine Interface – die Schnittstelle zur Hardware) erledigt.

Nachteil dieses Prinzips ist: Sollte eine Festplatte kaputt gehen, sind alle Daten im System inkonsistent, da keine weitere Verwaltung existiert, die Buch führt, welche Daten auf welchen Platten liegen. Es hilft nur ein Booten vom letzten Sicherungsband und die Restaurierung der kompletten (und hoffentlich funktionsfähigen) Sicherung. Aber wer lässt schon ein solches System im Produktionsbetrieb ohne Festplattenredundanz (RAID) laufen? ☺ Auf der AS/400 wird oft ein spezieller RAID5-Controller verwendet, die allerdings (wie vieles bei IBM) sündhaft teuer sind. Größere Systeme sind oft mit der

[I]Hiermit ist gemeint, dass die CPU neben den 64 Datenbits auch jeweils ein weiteres, das so genannte „Tag“-Bit verwaltet. Dieses Tag-Bit zeigt an, ob der in den 64Bit enthaltenen Pointer gültig ist, also vom System geprüft.

luxuriösen Variante ausgestattet: Hier ist jede Festplatte gespiegelt (RAID 1). Da die AS/400 alle Platten wie einen Speicher betrachtet, kann man hier auch von RAID10 reden. Daneben kann man dann pro Plattensatz im Notfall bis 1.6GB Cache einbauen...

Ganz große Systeme verwenden oft ein externes Speichersystem (SAN – Storage Area Network). Hier hat man ein eigenes Gehäuse mit vielen Festplatten, die virtuell aufgeteilt sind und gleichzeitig mehreren Systemen (AS/400, Unix, PC, Grossrechner) zur Verfügung gestellt sind. Intern sind die Platten natürlich redundant aufgebaut, oder man spiegelt diese Plattenschränke gleich komplett.

Aus der besonderen Speicherverwaltung ergibt sich eine weitere Eigenart der AS/400. Das System läuft nur mit speziellen IBM (jetzt: Hitachi) - Festplatten bzw. von IBM freigegebenen Speichersystemen wie IBMs eigene (Shark), oder von EMC etc. Grund hierfür ist, dass jeder Sektor auf der Festplatte nicht mit 512 Bytes wie im PC oder Unix-Bereich, sondern mit 520 Bytes formatiert ist. Diese zusätzlichen 8 Byte dienen der Verwaltung der Speicheradressierung und dem Schutz der Pointer. Technisch gesehen sind die Festplatten gleich, nur die Formatierung der Platte ist anders. (Es gehen immer wieder Gerüchte um, man könnte normale PC-Platten auf 520 Byte/Sektor neu formatieren und in die AS/400 einbauen. Dann müsste man aber auch eindeutige Seriennummern auf die Platten schreiben und spezielle IBM-Teilenummern, mit denen sich die Platte meldet. Wer das schafft, dürfte viele Freunde haben ;-) Meine eMail-Adresse kennst Du ja.

Generell ist angemerkt, dass IBM im Bereich Zusatzhardware zur AS/400 sehr viel restriktiver als zum Beispiel bei den RS/6000 Unix-Servern (heutzutage pSeries genannt) oder den PC-Servern (xSeries) ist. Dort darf man jede Festplatte oder RAM einbauen, die man bei eBay oder wo auch immer kaufen kann. Sollte allerdings das System einmal ausfallen, wird der IBM-Techniker sagen: „Da steht nicht IBM drauf, das ist also nicht IBMs Problem.“ Entsprechend teuer könnte dann ein Servicefall werden. Hat man in seiner AS/400 nur Teile von IBM und einen Wartungsvertrag, kann man beruhigt schlafen (der Autor zahlt selbst für seine kleine AS/400 einen Wartungsbetrag und hatte schon davon profitiert. Eine neue Ersatzplatine hätte ohne Wartung ein Heidengeld gekostet.) Ausserdem garantiert IBM die Ersatzteilversorgung bis 10 Jahre nachdem ein Gerät nicht mehr hergestellt wird.

Skalierbar

Skalierbarkeit ist ein Begriff, der zwar heute gerne in der Intel-Welt verwendet wird, um zu beweisen, dass man aus einem Single-CPU-System auch ein Dual-CPU-System machen kann. Erfunden wurde aber die Bandbreite der nutzbaren Hardware von IBM mit den verschiedenen Modellen der AS/400-Vorgänger. Es gab mehrere Modelle mit größtenteils gleichen oder ähnlichen Hardwarekomponenten, allerdings war die maximale Ausbaustufe in Speicher und Systemleistung unterschiedlich. Kunden konnten mit einem kleinen System anfangen und bei Bedarf dieses aufrüsten oder ein größeres System anschaffen. Die Software des Kunden lief auch auf dem größeren System, mit entsprechend erhöhter Performance. Und das Betriebssystem skaliert auch nach oben recht offen. Die derzeitigen Modelle mit 32 CPUs haben im Schnitt eine 30fache Leistung als ein System mit nur einer CPU gleichen Typs. Allerdings baut man in solche Hobel auch gleich etwas mehr RAM und weitere Festplatten ein.

Dies ist heute bei AS/400 Rechnern nicht anders. Das System des Autors (Typ 9406 Modell 170), auf dem Ihr auch arbeiten könnt, wird leider nicht mehr hergestellt, es ist ein gemütlicher Rechner Baujahr 1999 mit einer CPU, 64 bis 1024MB RAM und zwei bis 80GB Festplattenplatz. In der Grundausstattung gab es das Gerät mit 64MB RAM und 4GB Festplatte (war aber nicht wirklich flott). Die aktuellsten Systeme der AS/400-Reihe können bis 32CPUs unterstützen, 512GB RAM enthalten und 144TB Festplatten in bis zu 45 Systemgehäusen verwalten, ohne auf teure Storage-Systeme von IBM oder Fremdherstellern angewiesen zu sein.
Alle Systeme laufen mit der gleichen System- und Anwendungssoftware. So kann der Kunde frei entscheiden, wie viel Leistung er benutzt. Natürlich muss auch der Geldbeutel entscheiden, die größten Systeme kosten in der Grundausbaustufe (Gehäuse, CPU-Boards, eine gespiegelte Platte zum Booten und etwas RAM) schon diverse Millionen Euro.

Objekt oder kein Objekt?

Das Betriebssystem OS/400 bekommt von der Hardware nichts mit. Zwischen Betriebssystem und Hardware liegt der sogenannte LIC (Licensed internal code), sowie eine Software namens MI (Machine Interface). Es ist etwas kompliziert, zu definieren, was das Betriebssystem der AS/400 ist. Vereinfacht ist es diese Kombination aus OS/400, LIC und MI. Der Hauptteil des Betriebssystems, hier OS/400 genannt, greift mehrstufig über MI/LIC auf grundlegende LIC-Routinen zu, und ist somit vollständig von der Hardware entkoppelt. Umgekehrt kann die Kombination LIC/MI dem System unterschiedliche Prozessmodelle zur Verfügung stellen.
Ergebnis ist, dass OS/400 auf unterschiedlichen CPUs laufen kann, umgekehrt kann IBM derzeit AS/400 Modelle ausliefern, auf denen gleichzeitig OS/400, Linux und AIX läuft (im Notfall auch auf der gleichen CPU).

Das MI vermittelt zwischen System und Hardware mit dem Ergebnis, dass IBM Anfang der 90er die Hardware von 48Bit CPUs auf 64Bit-CPUs umstellen konnte. Der Anwender bekam außer einer einmalig nötigen Umwandlung der Programme und der danach folgenden Leistungssteigerung nichts mit.
Es existieren von den derzeit verwendeten 64Bit-RISC-CPUs mindestens 6 verschiedene Typen. Für jedes Hardwaremodell gibt es eine Variante des LIC, aber das Betriebssystem OS/400 und alle Softwareprogramme sind stets gleich.

Grund hierfür ist die konsequente *objektbasiertheit* des Betriebssystems. Objektbasiert ist hier etwas anderes als objektorientiert, in dem vorhandene Objekte abgeleitet und modifiziert werden können. Unter OS/400 kann man mit Objekten arbeiten, die auf einer Vorlage basieren, aber diese Objekte nicht nach Belieben verändern, sondern nur kopieren. Man kann also nicht das Objekt „Müller“ vom Typ „Benutzerprofil“ kopieren und der Kopie eine Eigenschaft „Datendatei“ zuordnen. (würde auch keinen wirklichen Sinn machen). Jedes Objekt hat definierte Rechte, Eigenschaften und Möglichkeiten, mit denen man arbeiten kann. Somit kann eine Sicherheit und Integrität gewährleistet werden.
Um zu demonstrieren, wie hardwareunabhängig dies ist: Wenn ich auf meinem kleinen Systemchen eine Komplettsicherung auf Band mache, kann ich ein 32CPU-System von diesem Band booten und das neue System mit allen Benutzerdaten installieren. Bitte mal mit einem Intelbasierten Server probieren: Zieh ein Image von Deinem alten VLB-486er und spiele das auf einen Compaq-8-Wege-Server.[I]

Um einigen Fragen vorwegzugehen: Man kann nicht einfach einen Emulator der MI-Befehle schreiben und damit OS/400 auf einer anderen Plattform verwenden. Viele Teil von Betriebssystem und anderen Routinen sind inzwischen direkt PowerPC-optimiert, so dass das Unterfangen nicht funktionieren dürfte.

[I]Der Ehrlichkeit halber eine Erwähnung: Beim Rückspielen der Sicherung auf ein anderes System werden viele Systemobjekte über die üblichen Installationsroutinen zurückgelesen, so dass es im Grunde einer Neuinstallation mit Konfigurations-Restore entspricht.

Eine Datenbank ohne Datenbank?

Die Datenbank ist Grundbestandteil des Betriebssystems OS/400, bzw. umgekehrt. Das Betriebssystem kann ohne Datenbank ebenso wenig existieren wie andersrum. Ursprünglich hatte die Datenbank-Software auch keinen Namen, da sie nicht extra vermarktet wurde, sondern es Mitte der Achtziger Jahre als „ganz normal“ betrachtet wurde, dass die Datenbank im System integriert ist. Ein Zitat von Frank Soltis soll dieses Verständnis sowie die Denkweise vieler Anwender verdeutlichen [Quelle 2]:

> *„Ich war immer dagegen, der AS/400-Datenbank einen Namen zu geben. Nach einem Treffen mit einem sehr großen IBM-Kunden, der von einem anderen System auf die AS/400 umstieg, habe ich meine Meinung geändert. In diesem Meeting wurde ich gefragt, welche Datenbank ich für die AS/400 empfehlen würde. Ich platzte mit ‚DB2/400’ heraus. Wir benutzen diesen Namen intern. Sofort begannen alle im Raum zu nicken und zu bestätigen, dass das eine sehr gute Datenbank sei. Wenn man bedenkt, dass diese Leute in ganz anderen IBM-Abteilungen arbeiteten und von der AS/400-Datenbank keine Ahnung hatten, dann wird klar, dass ich selbst ab diesem Moment überzeugt war, dass wir einen Namen brauchten.“*

So kam das Kind zum Namen. Selbstverständlich unterstützt die DB2/400 alle Eigenschaften einer modernen relationalen Datenbank, sowie SQL und einige Besonderheiten der DB2, die auch auf anderen Plattformen existiert.

Als letzter Teil dieser Theorie sei nur noch gesagt: Vergiss' die organisatorischen Eigenarten von Windows oder Unix, unter OS/400 ist alles noch viel eigenartiger. Aber wenn man sich damit beschäftigt, wird man es (teilweise) lieben lernen, denn trotz (oder wegen) seiner Obskuritäten hat dieses System hervorragende Steuerungsmöglichkeiten.

So, nun genug der trockenen Theorie, der nächste Teil widmet sich dem Anmelden und der ersten Schritte.

First Contact

Wollen wir uns nun darauf vorbereiten, die erste Bekanntschaft mit einer AS/400 zu machen. Wir werden uns anmelden, die ersten Befehle kennen lernen sowie ein Gefühl für die Bedienung bekommen.

Das Terminal

Da die AS/400 ein Multiuser-System ist, benötigt man ein entsprechendes Terminal, um auf das System zuzugreifen. Es gibt nicht wie bei einem PC einen Bildschirm- und Tastaturanschluss direkt am System. Früher üblich waren Twinax-Terminals. Das sind "dumme" Bildschirme mit einem Anschlusskabel nach Twinax-Norm. Dies ist ein serieller Bus mit einem bis zwei MBit und einer Reichweite von 2km (und mehr mit Zwischen-Verstärkern). Diese Anschlussart ist inzwischen überholt, diese Terminals werden durch Terminalemulationen auf den vernetzten PCs ersetzt.

Mit dem Standardprotokoll TCP/IP kann theoretisch mit einem üblichen Telnet-Programm nach VT100-Norm auf die AS/400 zugegriffen werden. Allerdings werden hierbei nicht alle Funktionstasten (F1-F24) sowie einige Sondertasten unterstützt.

Bezugsquellen

Das Protokoll der auf der AS/400 verwendeten Terminals ist Telnet mit der so genannten 5250-Emulation. Somit benötigt man ein TN5250-kompatibles Terminal-Programm, wie z.B. http://www.mochasoft.dk erhältlich.

Von Andreas kam noch der Hinweis auf: http://tn5250j.sourceforge.net
Dies ist ein Java-Basierter Client und benötigt entsprechend etwas mehr Rechnerleistung (konnte ich mangels Zeit noch nicht ausprobieren).

Ansonsten kann man im Internet mit einer Suchmaschine nach „Telnet 5250“ suchen und wird bestimmt fündig.

Hat man ein solches Telnet-Programm installiert, sollte man ein paar Grundeinstellungen vornehmen, dann kann man sich anmelden.

Am Beispiel des Mochasoft-Terminals erkläre ich diese Parameter:
(Version W32 TN5250 Version 7.0, deutsch)

Einrichten des Telnet-Clients

Nach der Installation des Mochasoft-TN5250 und dem ersten Starten sollten folgende Einstellungen vorgenommen werden:

Sprachenauswahl:

Im Menü "Einstellungen", Unterpunkt „Mappingdateien (Sprache)..." sollte man sich zuerst die deutsche Tastatur- und Zeichenbelegung auswählen.

Im Fenster "Namen der Mappingdateien auswählen" befindet sich auf der rechten Seite eine Liste der möglichen Belegungsdateien. Hier einmal auf den Eintrag "Austria/Germany (273)" klicken, damit im Feld "Mapping Datei für EBCDIC<->ASCII" der Eintrag "`<Pfad zum Programm>ebcdic52.ger`" erscheint. Die Auswahl mit OK bestätigen. Somit ist die Tastatur schon mal korrekt belegt.

Terminaltyp:

Im Menü "Einstellungen", Unterpunkt "Terminaltyp" kann man einstellen, ob der Bildschirm 24x80 oder 27x132 Zeichen unterstützt. Beide angegebene IBM-Typen sind Bildschirme mit Standard-Darstellungsgrößen von OS/400. Da das gesamte System Bildschirm-orientiert und nicht Zeichenorientiert arbeitet, ist der Benutzer auf diese Bildschirmformate festgelegt, da alle Programme mit festgelegten Positionen arbeiten. Für diese Option empfehle ich 27x132 (IBM 3477). Sollte Dein Bildschirm nur eine Auflösung von 800x600 Punkten erlauben, empfehle ich 24x80 Zeichen (IBM3179).

Im Feld "`Gerätename:`" kann man einen Namen für die Terminalsitzung einstellen, sofern der Administrator einen vergeben hat. Lässt man dieses Feld leer, generiert die AS/400 automatisch einen Namen. Es wird empfohlen, dieses Feld vorerst leer zu lassen. Wichtig: Einige Benutzer wählen sich hier selbst einen Namen aus. Dann bitte folgendes Beachten: Bildschirmnamen dürften max. 10 Zeichen lang sein.

> **Eine Bitte für Nutzer von `ipublic.online`**: Die ersten drei Buchstaben sollten „`DSP`“ sein, danach kann der Anfang Deines Benutzernamens folgen. Willst Du mehrere Bildschirme gleichzeitig nutzen, hänge Ziffern hinten dran. Beispiel: `DSPKLAUS1`, `DSPKLAUS2` etc...

Wenn Du ein Programm zur Emulation eines Druckers geladen hast, bitte hier unbedingt einen Gerätenamen angeben. Dieser sollte mit „`PRT`“ anfangen. Somit kann ich die Geräte genauer zuordnen.

Mit dem letzten Auswahlfeld "Display (LTTI)..." kann man festlegen, dass in der Statuszeile angezeigt wird, wie lange das System beschäftigt ist, nachdem man einen Bildschirm freigegeben hat (mehr dazu später). Hier kann man ruhig ein Häckchen machen, ist manchmal interessant.
Die Eingaben wieder mit OK bestätigen.

Tastaturbelegung:

Zuletzt sollte man noch einige Tasten umdefinieren. Die AS/400 arbeitet wie beschrieben Bildschirm-orientiert, im Gegensatz zur Zeichenorientierung unter normalem Telnet auf Unix etc. Wenn Du dort eine Taste drückst, wird sie sofort vom Rechner bearbeitet, gibst Du einen Befehl in der Shell ein und drückst Return, passiert etwas. Auf der AS/400 läuft das anders:

Wenn man mit einem 5250-Terminal einen Bildschirm sieht, besteht dieser aus vielen Feldern (zur Darstellung von Text) und Eingabefeldern (meist unterstrichen). Die Eingabefelder kann man beliebig bearbeiten, zwischen ihnen mit der Tabulatortaste oder den Pfeiltasten wechseln etc. Erst wenn man auf eine spezielle Taste drückt, wird der gesamte Bildschirminhalt wieder an die AS/400 zurückgesendet, die sich dann um die Bearbeitung kümmert, und die Ergebnisse an Dein Terminal zurücksendet.

Hierbei ergibt sich eine gewisse Begriffsverwirrung, die auch bei IBM wohl bei einigen Handbuchautoren und Übersetzern für Verwirrung gestiftet hat. Um einen Bildschirm an die AS/400 abzusenden, gibt es die so genannte "Datenfreigabe"-Taste, auch "Eingabetaste" oder englisch "Enter" genannt.
Daneben gibt es die "Feldeingabe" oder "FieldExit" Taste, die die Eingabe in einem Eingabefeld abschließt und in das nächste Feld springt. Davon bekommt die AS/400 aber nichts mit. (Verwirrung komplett? :-)

In der Standardkonfiguration bei Mochasoft TN5250 ist die *Datenfreigabe* auf der Return-Taste Deines PCs gelegt, dies ist nicht ganz optimal. Es wird empfohlen, die "Feldeingabe"-Taste auf die Return-Taste zu legen, und Datenfreigabe auf die rechte STRG-Taste zu legen (so wie Du es vielleicht von den alten IBM InfoWindow® Terminals kennst. Wie das geht?

Im Menüpunkt "Einstellungen", Unterpunkt "Tastaturlayout" kann man die Tasten umbelegen. In dem nun geöffneten Fenster ist links eine Liste der PC-Tasten und rechts eine Liste der 5250-Funktionen. Bitte suche links den Eintrag "ENTER" aus und klicke ihn einmal an. Damit hast Du Deine PC-Taste gewählt (der Autor von Mochasoft meint hiermit die RETURN-Taste).
Nun in der rechten Liste den Eintrag "FIELDEXIT" suchen und doppelklicken. Nun müsste unter dem oberen Anzeigefeld "Aktuelle 5250 Funktion:" der Wert "FIELDEXIT" erscheinen, und die Funktion ist der entsprechenden Taste zugeordnet.

Ebenso muss der linke Eintrag "CTRL" mit dem rechten Eintrag "ENTER" verbunden werden.
Danach diese Angaben mit „Sichern in Datei“ abspeichern.

Das wäre es auch schon, sorry, wenn es zu ausführlich oder kompliziert beschrieben ist, man muss es ja nur einmal machen. Und einige Neulinge in der AS/400-Welt tun sich verständlicherweise damit schwer.

Nun im Menü "Datei" Unterpunkt "Verbinden" auswählen. Es erscheint ein Fenster, in dem der Host anzugeben ist. Hier "ipublic.online" (oder der Name des gewünschten Systems) eintragen und mit OK bestätigen.

Anmelden:

Es sollte nun das Anmeldebild erscheinen (wenn nicht, bitte die Einstellungen prüfen):

Abbildung 1 : Das ist nicht der Standard-Anmeldebildschirm einer AS/400, ich habe ihn auf meinem System geändert, damit es nicht so langweilig ist :-)

Das Feld für den Benutzernamen ist unterstrichen, da hier eine Eingabe möglich ist, das Kennwortfeld ist nicht unterstrichen, da die Eingaben hier nicht angezeigt werden. Selbstverständlich kannst Du dort Dein Kennwort aber eintippen.
Unter DEIN BENUTZERNAME tippe bitte den Dir zugeteilten Benutzernamen ein (hier im Handbuch wähle ich HACKER, das Benutzerprofil für anonyme Anwender). Durch Verwenden der Tabulator-Taste kommst Du in das Eingabefeld PASSWORT (hier musst Du blind Dein Kennwort eintippen). Ich tippe hier auch HACKER. Wenn Du nun auf RETURN drückst, wechselt der Cursor wieder ins nächste Eingabefeld, also zum Feld ALT. STARTMENÜ; es passiert sonst nichts. Erst wenn Du auf die rechte STRG-Taste drückst, kümmert sich die AS/400 um Deine Eingabe. Die so belegte Taste nennt man „Datenfreigabe“.

Drin!

Nun kommt aller Wahrscheinlichkeit ein Bildschirm mit Nachrichten (wenn nicht, ignorieren, aber diesen Abschnitt trotzdem lesen, beim Anmelden bekommt der Nutzer immer angezeigt, ob Nachrichten für ihn vorliegen).

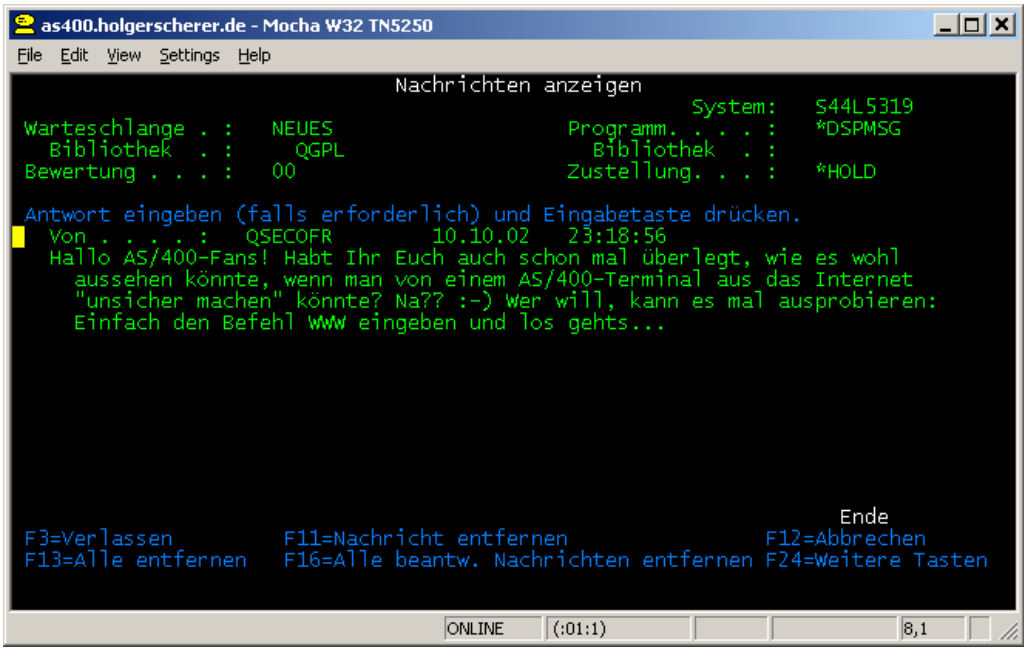


Abbildung 2 : Du bist angemeldet!

Der erste AS/400-Bildschirm

Dieser Bildschirm ist typisch für eine AS/400 Anwendung. In der Mitte der obersten Zeile steht "`Nachrichten anzeigen`", dies ist die Bildschirmüberschrift. Darunter in grün einige Zeilen mit Angaben zu der Warteschlange, deren Nachrichten anzeigt werden (dazu später mehr). Darunter steht in blauer Schrift "`Antwort eingeben (falls erforderlich)...`"
An dieser Stelle wird in vielen Bildschirmen angezeigt, was man mit den Einträgen anfangen kann.

Nun folgen in Grün die wichtigen Informationen, in diesem Falle einige Nachrichten aus der Nachrichtengruppe „`NEUES`“, die ich für alle Benutzer anlege.

Unten, am Fuß des Bildschirms, sind einige Funktionstasten beschrieben. Diese Funktionstasten gelten immer für den gerade angezeigten Bildschirm und sind für jeden Bildschirm eigens definiert. Weiterhin sind meistens für gewisse Funktionen die Tasten immer gleich. Merken kann man sich immer: F3=Verlassen, F12=Abbrechen und F24=Weitere Tasten. Nicht angezeigt wird die Taste F1, diese gilt zu 99% bei allen IBM-Bildschirmen, und sollte in selbst geschriebenen Programmen auch eine Hilfefunktion bieten.

Die Taste F3 beendet die aktuelle Funktion bzw. das aktuelle Programm. F12 bricht die aktuelle Funktion ab. Dies hört sich am Anfang nach der gleichen Funktion an, hat aber manchmal unterschiedliche Wirkung, F12 springt einen Bildschirm zurück, mit F3 wird manchmal ein Programm komplett beendet (abhängig von der Laune des Programmierers).
Die Taste F24 zeigt am Bildschirmfuß weitere Funktionstasten an, da nicht immer alle Beschreibungen dort Platz finden. Mal kommt es vor, dass eine Funktionstaste angezeigt wird, die gar nicht funktioniert. Da hat entweder der Programmierer geschlampt, oder es war zu kompliziert, die unteren Zeilen anzupassen (aber das ist keine gute Ausrede).

Drücken wir an dieser Stelle F3, um den Bildschirm "Nachrichten anzeigen" zu verlassen. (Oft kann man einen Bildschirm auch einfach mit Datenfreigabe verlassen).
Aus Faulheit schreibe ich ab sofort anstelle `Datenfreigabe` nur noch DF oder „Bestätigen“, ok? :-)

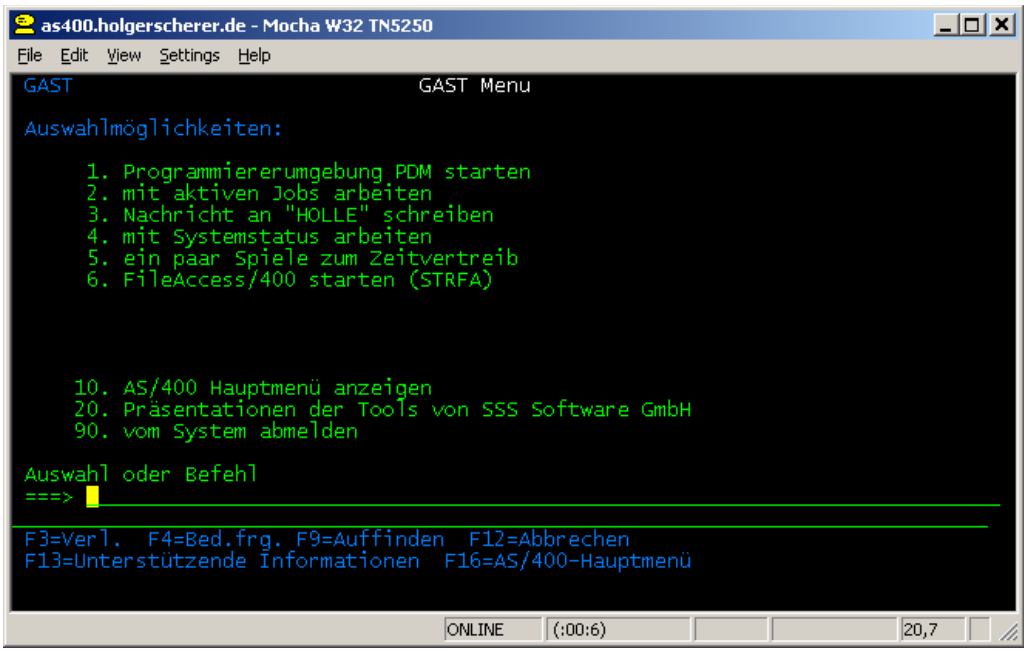


Abbildung 3 : Das ist Startmenü für Gäste auf meiner AS/400

Nun kommt das für den Benutzer definierte Anfangsmenü, beim Account "HACKER" und vielen der Benutzeraccounts auf meinem System ist dies das "GAST Menu". Solltest Du dieses Menü *nicht* sehen, gib bitte folgendes ein:

```
GO GAST DF
```

Bitte wählen Sie – Das Menü

Hier kann man schon etwas herum schnuppern. Auf dem Bildschirm stehen einige Einträge mit einer Zahl davor. Gibt man diese Zahl gefolgt von DF ein, wird das gewünschte Programm gestartet.

Schauen wir uns beispielsweise den Systemstatus an. Gib ein:
4
und drücke auf DF.

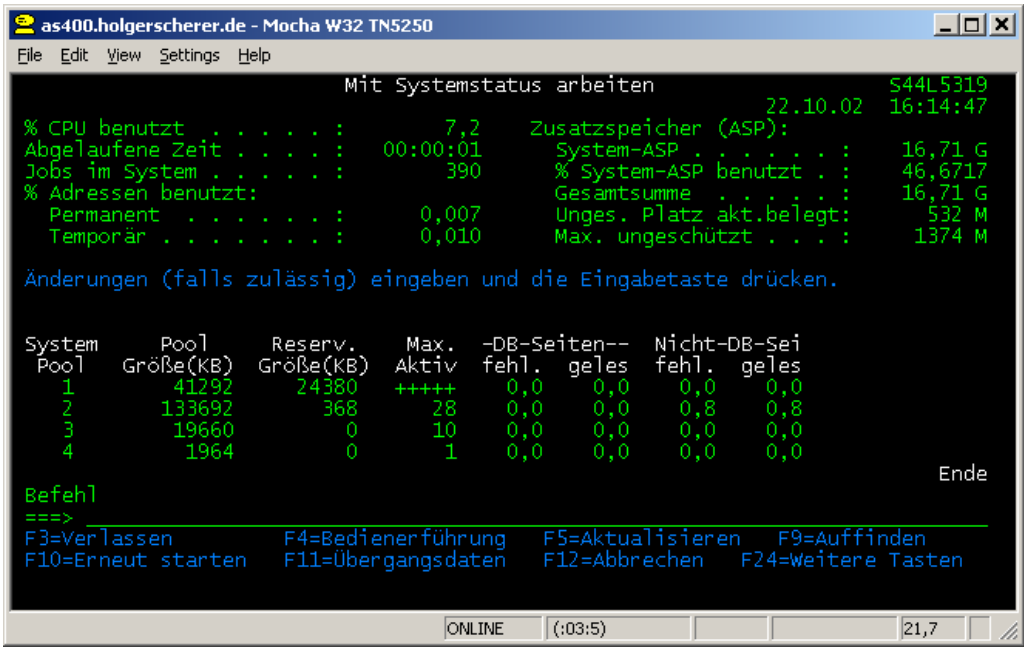


Abbildung 4 : Die Anzeige von WRKSYSSTS

Ein typischer Bildschirm

Es erscheint ein Bildschirm "`Mit Systemstatus arbeiten`". Auf dieser Anzeige erhält man eine ganze Menge Informationen zum System.
Als da wären zum Beispiel: aktuelle Auslastung der CPU (oder aller Prozessoren im Schnitt, sofern man sich mehrere leisten kann), Auslastung des Speichers und viele weitere interessante Informationen.

Unten stehen wieder die Funktionstasten.

Wenn man nun etwas mehr über einen angezeigten Wert wissen will, kann man gleich eine der nützlichsten Tasten im System benutzen: Die Taste F1. Diese Taste funktioniert fast immer und überall, nicht nur kontextsensitiv, sondern auch Feldsensitiv.

Auf Deutsch: Der Cursor befindet sich nach Anzeige des Bildschirms in der Befehlszeile (siehe unten, rechts vom ===> Symbol). Drückt man nun F1, erhält man allgemeine Informationen zu diesem Bildschirm.

Bewege nun aber mal den Cursor auf das Wort %CPU BENUTZT (oben links) und drücke F1. Nun erscheint in einem Fenster ein Hilfetext zu genau diesem Systemwert.

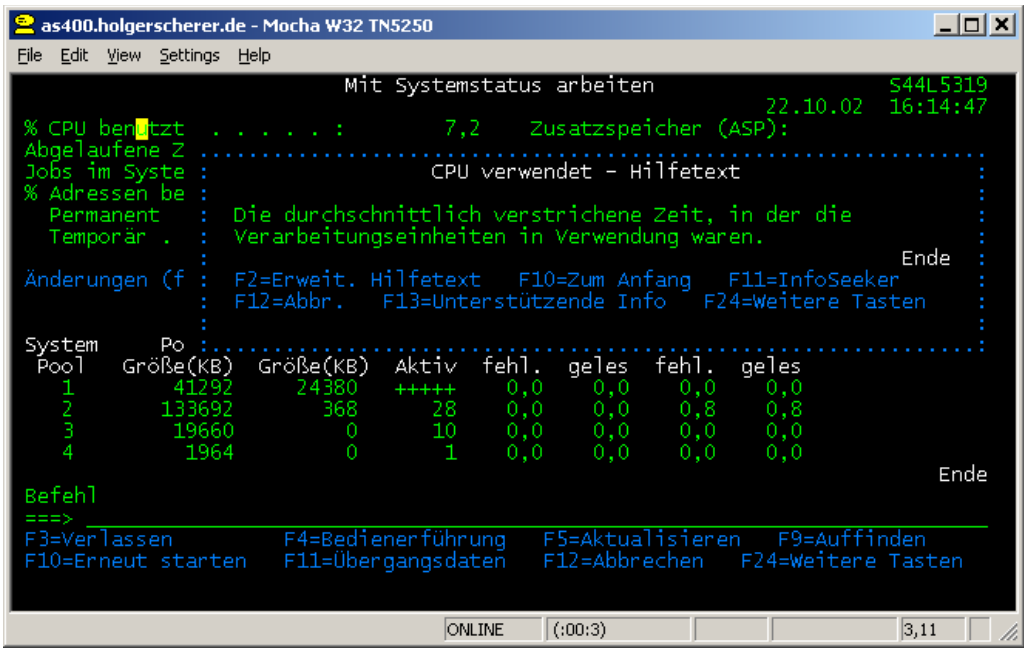


Abbildung 5 : Auch die AS/400 kennt Fenster, hier mit einem Hilfetext

Hilfe!

Im Hilfefenster sind auch Funktionstasten angegeben. Unten rechts steht „Weitere...“, dies bedeutet, dass mehr Informationen vorhanden sind, als in das Fenster passen. Blättere einfach mit Bild hoch und Bild ab. Mit F20 kann man das Hilfefenster vergrößern, mit F12 wieder schließen.

Praktisch, nicht wahr? ☺ Dies geht mit jedem Feld auf diesem Bildschirm, probier es einfach mal aus.
Um diesen Bildschirm MIT SYSTEMSTATUS ARBEITEN zu verlassen, wie bekannt F12 oder F3 drücken. Nun befindest Du Dich wieder im GAST Menu.
Dies war mal die erste Begegnung mit der Arbeitsoberfläche der AS/400.

Kommunikation ist alles

Auf der AS/400 kann jeder Benutzer einem anderen eine lokale Nachricht senden. (Später lernen wir auch, dass Programme untereinander Nachrichten senden können)

Du willst sehen, ob für Dich Nachrichten vorhanden sind; also die für Dich vorhandenen Nachrichten anzeigen.

Englisch übersetzt könnte man sagen: „Display Messages“.
Nun streicht man alle Selbstlaute, es wird daraus:

```
DSPLY MSSGS
```

Das ganze kürzen wir zu Dreiergruppen, deren Abkürzung einen gewissen Sinn ergibt, und fügt alles zu einem Wort zusammen. Das ergibt nun:

```
DSPMSG
```

Schon haben wir den ersten Befehl kennen gelernt!

Also den Cursor in die Befehlszeile bewegen (das ist immer ganz unten eine unterstrichene Linie, meist hinter den Zeichen "===>"). Sollte Dein Cursor nicht da sein, drücke einfach mal die TAB-Taste. Nun eingeben:

dspmsg DF

(Die GROSS/Kleinschreibung eines Befehls ist völlig egal, die AS/400 ist in dieser Beziehung viel toleranter als Unix oder Linux).

Nun kommt der Bildschirm NACHRICHTEN ANZEIGEN, den Du vielleicht noch vom Anmelden in Erinnerung hast. Normalerweise wird dieser Bildschirm immer nach dem Anmelden angezeigt. Auf diesem Bildschirm stehen diverse System-Informationen und in der Mitte die für Dich vorhandenen Nachrichten. Möglicherweise steht dort "(KEINE NACHRICHTEN VERFÜGBAR)", das muss ich dann wohl nicht kommentieren :-)

Auch hier kann man mal wieder mit dem Cursor rumfahren und mit der F1 Taste spielen.
Mit F3 den Bildschirm bitte verlassen.

Nachrichten senden

Damit wir mal eine Nachricht haben, schicken wir uns selbst eine. (Ich weiß, nicht sehr aufregend. Wie man mit anderen Benutzern Nachrichten tauscht, erklären wir gleich danach).

Um eine Nachricht zu schreiben oder zu senden, braucht es wieder einen Befehl.
"Nachricht senden" oder "Sende Nachricht" heißt auf englisch:
`SEND MESSAGE`

SEND kürzt man mit SND ab (einfach das E streichen), das Kürzel für MESSAGE(s) kennen wir schon: MSG

Also lautet der Befehl: `SNDMSG` (halt, noch nicht eintippen!)

`SNDMSG` ist ein Befehl, der dringend mindestens einen Parameter verlangt. (warum? Schon mal einen Brief ohne Adressat und ohne Inhalt in den Briefkasten geworfen?) Nur, welche Parameter brauchst Du nun, wie heißen die und wie und überhaupt?

Die AS/400 macht es dem Anwender recht einfach. Befehle, die dringend einen Parameter benötigen, fragen nach diesen Parametern, wenn man sie nicht angegeben hat. Nun darfst Du also eingeben:

sndmsg DF

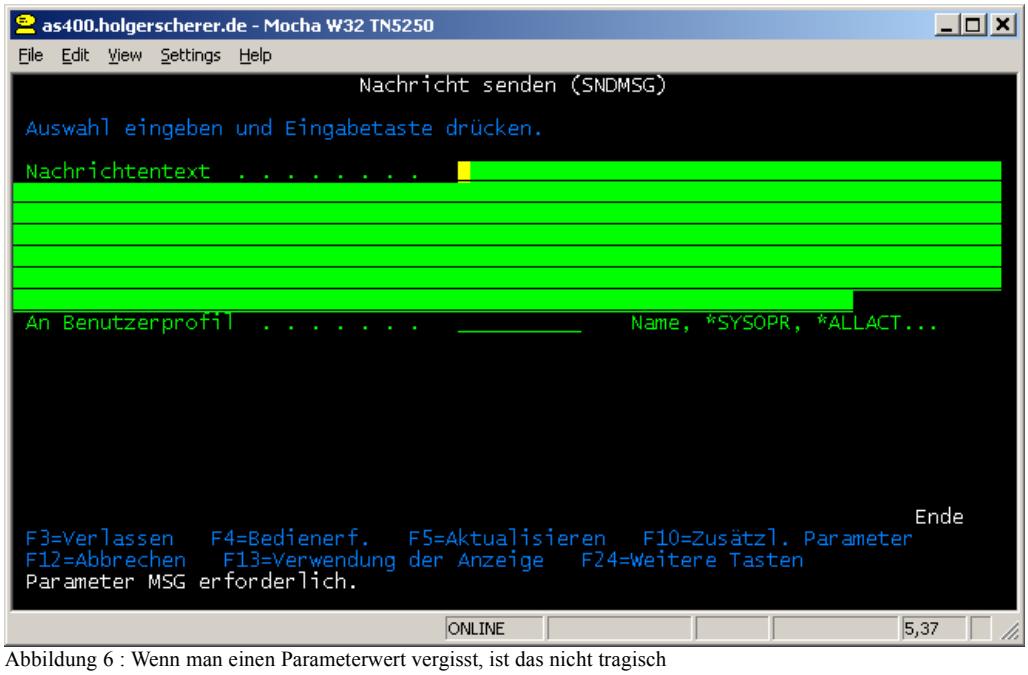


Abbildung 6 : Wenn man einen Parameterwert vergisst, ist das nicht tragisch

Es erscheint das Bild "Nachricht senden (sndmsg)“, und es werden zwei Eingabefelder angezeigt: Nachrichtentext und An Benutzerprofil. Das erste Feld ist invertiert angezeigt, und am Fuß des Bildschirms steht „`Parameter MSG erforderlich.`“

Dies bedeutet: Mindestens die Angabe für den Nachrichtentext ist nötig (klar, eine leere Nachricht macht nicht sehr viel Sinn, Du hast doch was zu sagen, oder?)

Tippe nun in das Feld Nachrichtentext folgendes ein:
Hallo Du da, wie geht es Dir?
(verlasse das Eingabefeld mit der Tabulator-Taste, das ist die Taste links vom Q)

Nun musst Du noch etwas in das Feld An Benutzerprofil eintippen, damit das System weiß, wohin es den Text senden soll. Nehmen wir der Einfachheit halber Deinen eigenen Benutzernamen. Ich verwende hier HACKER (für alle unter uns, die kein eigenes Profil haben).

Nun drückst Du DF, um die Daten an die AS/400 zu senden.

Erst jetzt wird der Bildschirm mit den ausgefüllten Feldern wieder an die AS/400 geschickt, die sich um die Bearbeitung der Parameter kümmert. Solange wir rumtippen, ist das System nicht belastet, die Arbeit der Verwaltung aller Felder erledigt das 5250-Terminalprogramm.

Nachricht gesendet

Das Bild Nachricht senden verschwindet, und erscheint der von vorhin bekannte Bildschirm Nachrichten anzeigen mit einer uns sehr bekannten Meldung :) Da Du die Nachricht an Dich selbst gesendet hast, wird sie auch direkt angezeigt.

Es wird gezeigt, wer wann welche Meldung gesendet hat.

(Sollte der Bildschirm nicht erscheinen, bist Du möglicherweise als HACKER angemeldet und momentan nicht der einzige mit diesem Benutzernamen im System. Warum das so ist, erkläre ich später. In diesem Fall gib bitte den Befehl DSPMSG ein. Dass Du nach einem Befehl immer die Taste DF drücken musst, lasse ich nun aus.)

Nun nutzen wir mal eine andere Funktionstaste. Wir haben die für uns bestimmte Nachricht erhalten, gelesen und gelacht :) Nun möchten wir diese Nachricht löschen, da der Inhalt nicht wirklich für ewig der Nachwelt erhalten bleiben muss.

Stelle sicher, dass der Cursor irgendwo auf der Nachricht steht, also innerhalb der beiden Zeilen "Von..." und "Hallo,..." die diese Nachricht bilden. Nun genügt ein vorsichtiger Druck auf F11, um die Nachricht aus dem System zu entfernen (siehe auch am Fuß des Bildschirms).

Da es nun keine weiteren Nachrichten mehr anzuzeigen gibt, springt die AS/400 auch wieder zu unserem letzten Bild (in dem Falle das GAST Menu) zurück.

Auch hier gilt: Sollte das Löschen mit der Funktionstaste nicht erlaubt sein (Betrachte eventuelle Meldungen am Bildschirmfuß), ist derzeit jemand anderes mit dem Benutzernamen HACKER angemeldet. Dies ist der richtige Zeitpunkt, um über ein eigenes Benutzerprofil nachzudenken ☺

Da dies mit den Nachrichten so schön geklappt hat, probieren wir das gleich noch einmal und lernen dabei noch einiges interessantes:

Wir könnten wieder SNDMSG eingeben, auf DF drücken und warten, dass uns das System wieder nach den Parametern fragt. Aber wir wollen doch Profis werden und wissen, wie die Parameter lauten, oder?

Prompt

Man kann in der Befehlszeile sich die zuletzt eingegebenen Befehle zurückholen.
Drücke mal auf die Taste F9! In der Befehlszeile sollte nun der verwendete Befehl:

```
SNDMSG MSG('Hallo, wie geht es Dir?') TOUSR(HACKER)
```

erscheinen. Wenn nicht, drücke bitte noch mal auf F9, bis diese Zeile erscheint.

Was bedeutet das ganze nun? Ist eigentlich nicht schwer zu erraten. Die AS/400 gibt Dir den zuletzt verwendeten Befehl vor und zeigt alle verwendeten Parameter mit an (sofern das in die Eingabezeile passt).

SNDMSG ist der Befehlsname.

MSG('Hallo, wie geht es Dir?') ist der Parameter MSG mit dem Wert 'Hallo, wie geht es Dir?'
(Werte für Parameter werden stets direkt nach dem Parameternamen -- ohne Leerschritt -- in Klammern angegeben.)
TOUSR ist der Parameter mit dem Wert HACKER für den Empfänger der Nachricht.

Die Namen der Parameter halten sich ebenso an die Dreier-Regel wie die Befehlsnamen, damit man sich nicht zu Tode tippt :-) Bei TOUSR klappt die Regel des Auslassens der Selbstlaute und Beschränkung auf drei Buchstaben nicht ganz, daher verwendet IBM hier eine Variante, die recht einfach lesbar ist.

Nun könntest Du direkt wieder DF drücken, um den Befehl mit den gleichen Parameterwerten abzusenden, oder Du nutzt das Prompting (mit F4), um die Parameter in einer Maske zu bearbeiten (Du könntest die Werte auch direkt in der Befehlszeile bearbeiten, ist aber bei mehreren Parametern nicht einfach, besonders wenn nicht alles in die Zeile passt).

Also einmal F4 drücken (dieses Prompting ist sehr vorteilhaft, Ihr werdet es noch lieben lernen, spätestens, wenn es ans Programmieren geht).

Es kommt wieder das bekannte Bild Nachricht senden, diesmal mit den beiden Parametern ausgefüllt. (Wer Lust hat, kann ja mal in einem Parameterfeld die F1 Taste drücken).

Also ändere den Nachrichtentext etwas ab und sende ihn erneut an Dich. Bitte beachte, dass beim Prompten die Textfelder immer in die für die Parameterübergabe notwendigen Hochkommata (') eingeklammert werden, so wie sie auch dem Parameter übergeben werden müssen. Daher solltest Du beim Ändern der Parameterwerte darauf aufpassen, diese Hochkommas entsprechend zu übernehmen oder ganz zu löschen.

Gut, nun haben wir uns selbst genug mit Nachrichten gequält, nun sind andere Benutzer dran.

Ein Hinweis zu diesen Nachrichten: Man kann sie nicht direkt mit einer eMail vergleichen, da mit `SNDMSG` verschickte Nachrichten nur auf dem lokalen AS/400 System versendet werden können. Der Empfänger muss nicht unbedingt angemeldet sein, die Nachrichten werden gespeichert, bis der Empfänger sie gelöscht hat, oder das System sie nach einer einstellbaren Zeit (bei mir 30 Tage) löscht.

Nun müssen wir natürlich wissen, *wem* wir Nachrichten senden können. (Griff in die Zukunft: Es ist theoretisch möglich, sich alle im System angelegten Benutzerprofile direkt anzuzeigen, das habe ich aber aus Sicherheitsgründen deaktiviert. Eine andere Möglichkeit, die gerade aktiven Benutzer zu ermitteln, wird später aufgezeigt :-)

Kommen wir nun zu etwas ganz anderem... Eine Möglichkeit ist, sich anzeigen zu lassen, wer gerade im System angemeldet ist. Diesen Leuten kann man eine Nachricht schreiben, und sie können dann auch direkt zurück schreiben.

Was ist denn hier los?

Auf der AS/400 sind alle Programme, die sich derzeit so im System rumtummeln, so genannte Jobs. Auch ein Benutzer, der angemeldet ist, erzeugt einen Job, auch wenn er nur das Hauptmenü betrachtet und Kaffee trinkt. Es läuft ein Programm, welches sich um die Aktivitäten des Benutzers kümmert, und das ist ein Job. Auch eine von einem Benutzer erstellte Druckausgabe, die noch nicht gelöscht ist, erzeugt einen eigenen Job. Somit kommt es schnell vor, dass auf einer AS/400 mehrere Hundert oder Tausende Jobs laufen, aber das ist für das System kein Problem.[I]

Diese aktiven Programme kann man sich anzeigen lassen. Da man (bei entsprechender Berechtigung) auch mit diesen Jobs auch arbeiten kann (verändern, anhalten, beenden etc.), nennt man dies "mit aktiven Jobs arbeiten", englisch "Work with active Jobs".
Nach der Dreier-Regel kürzt man das so ab:

`WRKACTJOB` (einfach, oder?)

[I]In aktuellen Systemversionen ist die Anzahl der maximal im System vorhandenen Jobs auf 999.999 beschränkt, aber ein solches System habe ich noch nicht gesehen.

Also den Befehl eintippen, DF drücken und schauen :)

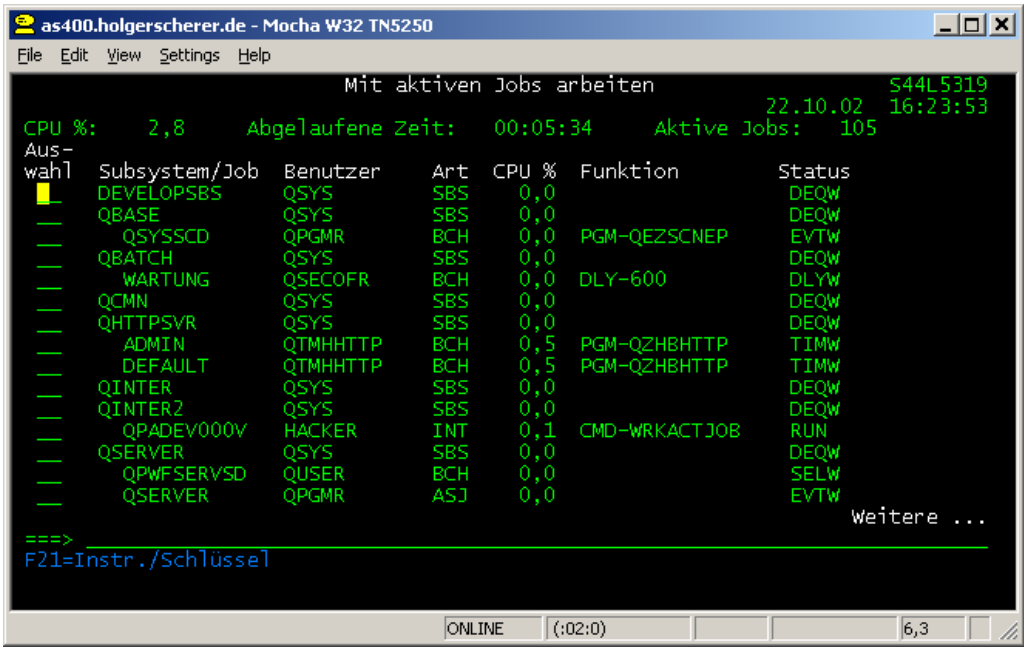


Abbildung 7 : Die Ausgabe von WRKACTJOB

Der Bildschirm MIT AKTIVEN JOBS ARBEITEN zeigt eine ganze Menge an. Ich erspare mir mal die Erklärung für die Angaben "CPU, Abgelaufene Zeit und Aktive Jobs", wie man das herausfindet, solltet Ihr nun wissen ☺

Schwer was los da.

Auf dem Bildschirm seht Ihr eine Menge Einträge. Betrachtet die Spalte "SUBSYSTEM/JOB". Was bedeutet dies? Hier seht Ihr die Namen der laufenden Jobs, sowie der Subsysteme (eine Stelle nach Links gerückt). Ein Subsystem ist eine logische Gruppe von Programmen. In so einem Subsystem sammelt man Programme mit ähnlicher Funktion, oder nach Abteilung sortiert. Jedem Subsystem kann man gewisse Systemressourcen zuteilen, sowie es beenden (und somit alle darin gekapselten Jobs), ohne dass es andere Subsysteme stört. Damit hat der Administrator des Systems nette Möglichkeiten, die Performance zu regeln etc. Halten wir uns damit aber im Moment nicht auf.

Administratoren an der lokalen Konsole haben ihre Jobs üblicherweise im Subsystem "QBASE" (das ist für administrative Zwecke gedacht), alle anderen Benutzer arbeiten meist in "QINTER". Große Systeme haben oft mehrere Subsysteme für die Benutzer.

Wie erkennt man nun, was ein Benutzer ist und was ein anderes Programm? Die Spalte ART zeigt an, was für ein Typ von Job angezeigt ist. "SBS" deutet auf ein Subsystem hin. Wir suchen aber normale Benutzer, die gerade interaktiv mit dem System arbeiten. Logischerweise ist die Abkürzung in der Spalte Art dafür "INT". (Ich verzichte ab nun, die Taste F1 anzupreisen. Kommt Eurem Forschungsdrang nach!)

Blättere also nun einmal mit Bild ab nach unten. Es sollte nun spätestens auf der zweiten oder dritten Seite irgendwo ein Eintrag für das Subsystem "QINTER" erscheinen, und darunter ein Job mit einem Namen wie "QPADEV...", einem Benutzernamen und der Art "INT". Mindestens ein interaktiver Job wird Dir immer angezeigt, das bist Du selbst :)

Um das Blättern zu sparen, kannst Du auch eingeben:

```
wrkactjob sbs(qinter qinter2)
```

Dies zeigt nur die Jobs in den beiden Subsystem QINTER und QINTER2 an, also die angemeldeten Benutzer. (Wirklich Neugierige Leute probieren hier gleich das Prompting aus. Bitte die möglichen Parameter vorerst ignorieren).

Hinweis am Rande: Im oberen Rand des Bildschirms werden sogenannte „Auswahlen“ angezeigt. Diese Zahlen kannst Du in die Spalte "Ausw" vor jedem Job eintippen und mit DF bestätigen (inzwischen dürfte klar sein, dass man der AS/400 meist mit dieser Taste sagen muss, dass sie etwas tun soll. Weiterhin gibt es dafür Funktionstasten oder die Blätter-Tasten. Ich erwähne das Betätigen der Datenfreigabe nun nicht immer wieder.)
Mit diesen „Auswahlen“ kann man den Job bearbeiten. Als normaler Benutzer hast Du üblicherweise nicht viele Rechte, also hat es keinen Sinn, großartig damit herumzuspielen, wenn man nicht weiß, was man damit anstellen kann. (besonders die Auswahl 4 vor dem eigenen Job kann sehr effektiv sein :-)) Wir wollten ja nur feststellen, wer angemeldet ist...

Dieses Prinzip der Auswahlen existiert bei vielen Bildschirmen.

Was ist aktiv?

Sollte ausser Dir kein "INT" Job anwesend sein, hast Du leider Pech gehabt. Solltest Du als HACKER angemeldet sein und weitere Benutzer mit HACKER angemeldet sein (dieses Benutzerprofil ist ja für alle gedacht), wirst Du auch keinen großartigen Erfolg mit dem Senden von Nachrichten haben. Nachrichten haben die Eigenschaften, dass sie im Normalfall nur demjenigen Benutzer sofort auf dem Bildschirm gezeigt werden, der sich als erster mit dem entsprechenden Benutzernamen angemeldet hat.

Du kannst ja spaßeshalber eine Nachricht an HOLLE (das ist mein Standard-Account zum Programmieren) oder an QSECOFR (das ist der Administrator, wie Root unter Unix) senden.
Hinweis am Rande (sollte man sich gleich merken): Alles, was mit Q anfängt, gehört entweder dem System, oder wurde von IBM eingerichtet oder hat Sonderfunktionen. Daher sollte man, wenn man ein neues Objekt erzeugt, vom Anfangsbuchstaben Q absehen. Ausnahme: Die Ablage für Quelldateien, dazu später mehr.

Mach mal Pause

Zwischendrin ein Befehl, den jeder kennen sollte:

```
SIGNOFF
```

Der Befehl hat die gleiche Funktion wie 90 im Menü, aber diese Auswahl ist nicht immer vorhanden. Einfach eintippen, DF und schon ist man abgemeldet.

Nun ist Zeit für eine kleine Lesepause, dann machen wir mit den Konventionen der Befehlsnamen weiter.

Wie sag ich's meinem Kinde?

Alle Funktionen auf der AS/400 werden mit Befehlen der Sprache CL (Command Language) aufgerufen. Einige dieser Befehle hast Du bereits kennen gelernt.
Damit man sich nicht jeden einzelnen der vielen Tausend Befehle merken muss, hat IBM eine gewisse Namenskonvention eingeführt, um von der gewünschten Aktion auf den Befehlsnamen schließen zu können. Befehle sind Objekte vom Typ `*CMD` und befinden sich in der Bibliothek `QSYS`.

Ein Befehlsname besteht aus einem bis drei Teilen:

Das Verb	(was soll getan werden?)
Das Objekt	(womit soll die Aktion ausgeführt werden?)
Die Parameter	(weitere Angaben zur Ausführung der Aktion)

Die einzelnen Teile eines Befehls bestehen im Allgemeinen aus einer Drei-Buchstaben-Gruppe und leiten sich aus dem dazugehörigen englischen Wort ohne Selbstlaute ab. Da diese Regel nicht immer funktioniert, gibt es natürlich auch Ausnahmen (keine Regel ohne Ausnahme).

Hier ein paar Beispiele:

Verben

Deutsches Wort	*Englisches Wort*	*OS/400 Kürzel*
Anzeigen	Display	`DSP`
Erstellen / Anlegen	Create	`CRT`
Editieren / Verändern / Bearbeiten	Edit	`EDT`
Hinzufügen zu einem Objekt/Liste	Add	`ADD`
Entfernen von einem Objekt/Liste	Remove	`RMV`
Löschen	Delete	`DLT`
Arbeiten mit Objekt / Liste	Work with	`WRK`
Starten einer Routine / Anwendung	Start	`STR`
Erteilen (Berechtigung)	Grant	`GRT`
Entziehen (Berechtigung)	Revoke	`RVK`
Senden	Send	`SND`

Objekte

Deutsches Wort	*Englisches Wort*	*OS/400 Kürzel*
Bibliothek	Library	LIB
Datei	File	F
Programm	Program	PGM
Befehl	Command	CMD
Nachrichtenwarteschlange	Message Queue	MSGQ
Ausgabewarteschlange	Output Queue	OUTQ
Aktive Programme / Jobs	Active Jobs	ACTJOB
Nachricht	Message	MSG
Teildatei	Member	MBR
Dateien in der Druckerschlange	Spooled Files	SPLF
Objekt allgemein	Object	OBJ

Besonders bei Dateien muss man oft genauer angeben, um was für eine Datei es sich handelt, mit der Du arbeiten willst. Daher gibt es für das Kürzel „F“ entsprechende Ersatzworte, die eine Datei genauer beschreiben:

Dateiarten

Deutsches Wort	*Englisches Wort*	*OS/400 Kürzel*
Physikalische Datendatei	Physical File	PF
Logische Datei (Index / Sicht)	Logical File	LF
Bildschrimdatei	Display File	DSPF
Druckerdatei	Printer File	PRTF
Teildatei (als Kürzelerweiterung)	Member	M

Ausserdem gehören gewisse Befehlskürzel zusammen. Wenn Du mit CRTXXX ein Objekt erstellst, kannst Du es stets mit DLTXXX wieder löschen. Was man einer Liste oder einem Objekt mit ADDXXX hinzufügt, kann man mit RMVXXX wieder entfernen. Du siehst, es ist gar nicht so schwer. Man muss sich nicht alle Befehle merken, nur grob die Art und Weise, wie die Tätigkeits- und Objektworte zusammengesetzt werden.

Anhand der Liste oben wirst Du Dich nicht wundern, dass Du für die gewünschte Funktion den nebenstehenden Befehl verwenden musst:

Du willst welche Aktion durchführen?	*nötiger Befehl*
Eine Bibliothek löschen	`DLTLIB`
Eine Bildschirmdatei erstellen	`CRTDSPF`
Mit einem Objekt arbeiten	`WRKOBJ`
Inhalt einer Teildatei (einer Datei) anzeigen	`DSPPFM`

Ein Tipp noch am Rande: Wenn Du Dir sicher bist, Du willst mit etwas arbeiten (`WRK`), aber nicht genau, womit, oder wie man das Kürzel schreibst, gib einfach mal einmal folgendes ein:

```
WRK*
```

Das Studium der folgenden Liste dürfte Dich eine Zeit lang beschäftigen und Du weißt, wie man sich den passenden Befehl auswählt.

So, nun lasse ich Dich das erst einmal verdauen. Mach eine kurze Pause, dann geht es weiter.

Ordnung im System

Ziel dieses Manuals ist, dass Du Dich etwas mit der AS/400 auskennst, Datenbanken erstellen und bearbeiten kannst, und später auch Anwendungsprogramme erstellen kannst. Dies ist ein weiter Weg, für mich zum Schreiben und für Dich zum Lesen, aber lassen wir uns nicht abhalten.
Bevor wir uns weiter in die Tiefen der Befehle und Möglichkeiten stürzen, muss ich noch ein paar Worte zur Organisation der Objekte verlieren, damit Du nicht verwirrt am Bildschirm sitzt.

Kein Dateisystem?

Im Gegensatz zu PCs oder Unix-Rechnern kennt OS/400 kein Dateisystem nach üblichem Verständnis. Das MI (Machine Interface, der Vermittler zwischen Hardware und Betriebssystem) ist der einzige Teil des Systems, der etwas von Festplatten und physikalisch angeordnetem Speicher weiß. Das Betriebssystem, die Anwendungsprogramme, Compiler und vor allem der Anwender kennen und sehen nur *Objekte*. Diese Objekte können unterschiedlichster Art sein und bilden das gesamte System. Es gibt Objekte vom Typ „Datei“, „Ordner“, „Benutzerprofil“, „Gerät“ etc. Jedes Objekt hat seine eigenen Eigenschaften, Vorgehensweisen und Möglichkeiten. Durch das MI geschützt, kann man Objekte nur mit den vorgesehenen Werkzeugen bearbeiten und nur die gewünschten Operationen daran vornehmen; somit ist die Sicherheit des Systems gewährleistet. Direkten Zugriff auf diese Objekte gibt es nicht (z.B. byteweises Auslesen eines Objektes vom Typ „Benutzerprofil“), sondern nur über die Befehle und Schnittstellen. Hierbei wird selbstverständlich geprüft, ob der aktuelle Benutzer tun darf, was er tun will. Diese Objektart kann man sich wie die Dateiendung auf einem Windows-Rechner vorstellen. Objektart „Programm“ entspricht „.EXE“, Objektart „TXT“ entspricht „.TXT“ etc. Allerdings kann man die AS/400 nicht austricksen, in dem man die Objektart einer Textdatei von „TXT“ nach „Programm“ ändert, im Gegensatz zu Windows ;-)

Bibliotheken

Alle Objekte im System werden in so genannten Bibliotheken (engl. „Libraries“) organisiert. Ich spreche bewusst nicht von „Ordnern“, dieser Begriff steht auf der AS/400 für die Verwaltung von PC-Daten. Diese Bibliotheken kann man sich aber wie die Ordner in einem Hauptverzeichnis eines PC-Laufwerkes vorstellen. Mit einem kleinen, jedoch gewichtigen Unterschied: Innerhalb einer Bibliothek kann man *keine* weitere Bibliothek anlegen, sondern nur Objekte. Dies mag auf den ersten Blick schädlich für die Ordnungswut der Anwender sein, ist es aber nicht. Bedenke, wir reden hier von einem System für kaufmännische Anwendungen, nicht, um tausende MP3-Dateien abzulegen :) Ausser dem sind Bibliotheken nicht alles; für das Netzwerk kann die AS/400 jedes beliebige Dateisystem und Protokoll anbieten (IPX/SPX, TCP/IP, NetBios, NFS, IFS, SMB etc.).

Bleiben wir bei den Bibliotheken. Ihre Anzahl ist nicht begrenzt (ausser durch den verfügbaren Speicher), und ihr Nutzen ist bei sinnvoller Nutzung sehr hoch. Der Anwender kann diese Bibliotheken einrichten und benutzen, wie es ihm beliebt, mit einer gewollten Ausnahme: Wie bereits vorher in diesem Manual besprochen, sind viele Objekte, deren Name mit Q beginnt, etwas besonderes. Man *kann* Objekte mit einem mit Q beginnenden Namen anlegen, aber man *sollte* es vermeiden (Ausnahme: Dateien für Programmquellen).

Es gibt auf jeder AS/400 einige Bibliotheken, deren Name mit Q beginnt, und diese wurden von IBM eingerichtet. Davon sollte man die Finger lassen, wenn man nicht weiß, was man tut!

Die wichtigste Bibliothek im System ist QSYS, für sie gelten einige Ausnahmen. Sie enthält nicht nur die meisten Objekte des Betriebssystems, sondern auch Objekte vom Typ „Benutzer“, „Gerät“ und „Bibliothek“. Du hast richtig gelesen, auch eine Bibliothek ist ein Objekt, das in QSYS gespeichert wird. Für den Benutzer sieht es allerdings so aus, dass alle Bibliotheken auf einer Ebene stehen.

Weiterhin gibt es noch eine ganz besondere Bibliothek namens QTEMP. Diese ist für temporäre Daten gedacht und existiert mehrfach. Wie das geht? Nun, QTEMP existiert für jeden Job in einer eigenen Instanz. Jeder Benutzer, jedes im Hintergrund laufende Programm hat seine eigene Bibliothek QTEMP, sie wird vom Betriebssystem beim Start des Jobs erstellt. Nur dieser Job kann auf seine QTEMP zugreifen, nicht auf die zu anderen Jobs gehörenden. Wenn der Job beendet wird, werden alle Objekte in der dazugehörigen QTEMP gelöscht. Bist Du also zwei mal angemeldet, hat jeder Deiner Jobs eine QTEMP, mit jeweils eigenem (am Anfang leeren) Inhalt.

Objekte

Nun kann man innerhalb dieser Bibliotheken seine Objekte einrichten. Innerhalb einer Bibliothek kann man viele Arten von Objekten anlegen, das wichtigste Objekt dürfte das Objekt vom Typ „Datei“ sein. Innerhalb einer solchen Datei kann man zum Beispiel Adressdaten einer Kundendatenbank ablegen. Um die Verwirrung etwas weiter zu treiben, kann eine Datei aus mehreren so genannten „Teildateien“ bestehen. Mindestens eine Teildatei ist immer vorhanden. Eine Teildatei enthält die wirklichen Daten, legt man weitere Teildateien an, müssen diese den gleichen Aufbau wie die erste haben. Somit kann man dreidimensionale Tabellen anlegen. Beispielsweise hat man eine Datei namens BUCHUNGEN, darin die erste und einzige Teildatei mit gleichem Namen (also BUCHUNGEN). In diesem Fall muss man den Namen der Teildatei nicht explizit angeben. Man kann aber auch in der Datei BUCHUNGEN die Teildateien BUCH200101, BUCH200102, BUCH200103 etc. anlegen, um für jeden Monat die Buchungen eigens abzulegen. Dann muss man aber zum Ansprechen der einzelnen Teildateien etwas mehr Arbeit betreiben.

Ein Objekt auf der AS/400 spricht man somit über den Namen der Bibliothek, dem Namen der Datei, und evtl. dem Namen der Teildatei an (sofern der Name der Teildatei vom Namen der Datei abweicht). Dies nennt man einen „qualifizierten“ Namen. In der Regel lässt man die Bibliothek aber aus, damit erreicht man eine sehr hohe Flexibilität. Dazu später mehr, wollen wir uns wieder der Praxis zuwenden.

In einer Baumstruktur würde das ungefähr so aussehen:

Objektname	Objekttyp	Sonderwert Objekttyp
➢ MEINELIB	(Bibliothek)	*LIB
➢ DATEI1	(Datei)	*FILE
➢ DATEI1	(Teildatei)	*FILE / Member
➢ DATEI2	(Datei)	*FILE
➢ DATEN123	(Teildatei)	*FILE / Member
➢ DATEN124	(Teildatei)	*FILE / Member
➢ MEINELIB2	(Bibliothek)	*LIB
➢ PROG01	(Programm)	*PGM
➢ QSYS	(Bibliothek)	*LIB
➢ WRKACTJOB	(Programm)	*PGM
➢ MUELLER	(Benutzerprofil)	*USRPRF

Um den Blick für praktische Anwendungen nicht zu verlieren, baue ich dieses Buch in Projekten auf: Wir wollen uns jeweils eine kleine Anwendung schreiben. Während dieser Arbeiten werden wir mit den wichtigsten Komponenten des OS/400 in Berührung kommen, Platz für eigene Experimente ist natürlich vorhanden.

Die Grundlage für jede Datenbankanwendung ist natürlich eine passende Datenbank. Ich setze etwas Grundwissen über eine Datenbank voraus, was eine Tabelle, eine Zeile und eine Spalte ist. Sollte hier Erklärungsbedarf bei den Lesern vorliegen, bitte ich um Meldung, ich werde dann ein entsprechendes Kapitel einbauen.

Während der Arbeit mit den Projekten kommen auch immer mehr Feinheiten über die Bedienung und Programmierung hinzu. Es gibt auf der AS/400 so viele Möglichkeiten, dass man gleich zu Anfang es nicht übertreiben sollte.

Das erste Projekt: RPG-Grundlagen

Zum Üben nehmen wir uns zuerst eine klitzekleine Adressverwaltung vor, später kommt etwas komplizierteres, wie wäre es mit einem einfachen Buchhaltungsprogramm?
Zunächst werden wir eine Tabelle mit Adressdaten erstellen. Das Erstellen dieser Tabelle wird Grundlage für erste Experimente mit dem Editor, dem Datenbanksystem und einigen kleinen Tools. Danach werden wir ein Programm für die Batchverarbeitung der Adressen erstellen, als nächsten Schritt ein Programm zur Bearbeitung und zum Drucken der Adressen.

Um den theoretischen Teil nicht zu lang werden zu lassen, gehe ich Schritt für Schritt vor, beschreibe die Vorgehensweise und erkläre, was passiert. Somit kannst Du in aller Ruhe jeden Schritt nachvollziehen. Obwohl die AS/400 selbstverständlich SQL von Grund auf beherrscht, hebe ich mir dieses Thema für später auf. Zum einen, um das Verständnis für die Vorgehensweise der AS/400 zu bilden, zum anderen, um von den Grundlagen bis zur Eleganz des Nutzens verschiedenster Techniken aufzusteigen ☺

Die Bibliothek des Wissens

Als erstes brauchen wir einen Ort, an dem wir unsere Tabelle ablegen. Wie angesprochen, legen wir alle Objekte in Bibliotheken ab. Alle Benutzer mit einem festen Account haben mindestens eine eigene Bibliothek namens „LIB<+7Stellen vom Namen>“ oder „<Benutzername>1“ sowie „<Benutzername>2“. Diese sollte auch jetzt verwendet werden. Benutzer, die mit „HACKER“ angemeldet sind, sollten die Bibliothek „LIBHACKER“ verwenden. Da hier andere Leute Deine Daten ändern oder löschen können, wäre ein fester Account von Nutzen, oder Du verwendest andere Datei- und Programmnamen. Um alle Befehle nachvollziehen zu können, arbeite ich nun mit einem fiktiven Benutzer, der sich erst eine eigene Bibliothek anlegen muss. Ihr dürft natürlich die Befehle nachvollziehen, aber bitte! Löscht später unbenutzte Bibliotheken und Objekte wieder, ich möchte nicht ständig Aufräumarbeiten vornehmen, Danke! *Und wer schon eine Bibliothek hat, sollte diese natürlich verwenden.*

Eine neue Bibliothek anlegen

(Hinweis: Der hier beschriebene Befehl zum Anlegen von Bibliotheken ist für alle Benutzer auf meinem System gesperrt, da ich zu viele Müllbibliotheken im System hatte. Daher ist dieser Teil als Theorie zu betrachten. Danke für das Verständnis!)
Um eine neue Bibliothek anzulegen, braucht es natürlich einen Befehl mit Parametern. Neue Objekte werden immer erzeugt, englisch „create“. Eine Bibliothek heißt auf Englisch „library“. Daraus folgert sich der Befehlsname:

```
CRTLIB
```

Natürlich benötigt dieser Befehl mindestens einen Parameter, den Namen der zu erzeugenden Bibliothek. Wird er ausgelassen, fragt OS/400 wie bekannt nach. Der Name des Pflichtparameters ist `LIB(...)`. Bei vielen Befehlen, bei denen nur ein Parameter Pflicht ist, kann der Name des Parameters ausgelassen, natürlich nicht der Wert. Folgend wäre die komplette Befehlszeile für das Anlegen der Bibliothek „Meintest“ wie folgt: (bitte nicht alle Befehlsbeispiele abtippen)

```
CRTLIB LIB(MEINTEST)
```

Abkürzen kann man diesen Befehl, wenn man nur einen Bibliotheksnamen angibt:

```
CRTLIB MEINTEST
```

Dieses Auslassen der Parameternamen funktioniert aber nur, wenn man alle Parameter in der gleichen Reihenfolge angibt, wie OS/400 sie bei dem entsprechenden Befehl erwartet. Tippe nun mal:

CRTLIB F4

Mit dem Prompten siehst Du, welche Parameter OS/400 für diesen Befehl haben will. Neben dem Namen der Bibliothek gibt es einen Parameter Bibliotheksart sowie Text ´Beschreibung´. Rechts vom Eingabefeld für die Bibliotheksart wird Dir angezeigt, welche Einträge man hier vornehmen kann. Es existieren zwei Werte namens „*PROD“ und „*TEST“. Wieder eine Besonderheit. Die beiden Angaben mit einem Stern davor sind so genannte Sonderwerte und stellen vom System vordefinierte Parameterwerte dar. Wir werden diese Sonderwerte noch näher kennen lernen. Wenn Du keine Lust hast, die Hilfefunktion zu nutzen: *PROD bedeutet, dass diese Bibliothek Produktionsdaten enthält, *TEST bedeutet „Testdaten“. Das kann beim Sichern und Zurückspeichern (und bei anderen Gelegenheiten) von Nutzen sein. Derzeit ist für uns egal, was Du angibst.

Diese Sonderwerte sind keine frei wählbaren Werte für einen Parameter, sondern fest definierte. Es gibt Parameter, die nur solche Sonderwerte akzeptieren, andere Parameter nehmen auch frei eingebbaren Text. Das System erkennt anhand des Sternchens, dass die Wertangabe ein Sonderwert ist. Welche Werte erlaubt sind, erfährst Du ganz einfach. Wenn Du Dich mit dem Cursor auf dem fraglichen Eingabefeld befindest und F1 drückst, bekommst Du ein Fenster mit einer Beschreibung zu diesem Feld. Drückst Du aber statt dessen die Taste F4, so bekommst Du eine Liste der erlaubten Eingaben. Probier's einfach mal aus!
Der dritte Parameter dient dazu, eine kurze Beschreibung für Deine Bibliothek aufzunehmen. Ist keine schlechte Idee, oder?

Du könntest also wie gewohnt die nötigen Felder ausfüllen und dann DF tippen, oder Du tippst in die Befehlszeile den Befehl mit allen Parametern ein. Aber Halt! Wie heißen denn die beiden anderen Parameter für die Bibliotheksart und die Beschreibung? Sie werden doch gar nicht angezeigt! Keine Panik auf der Titanic, schau bitte mal auf den Fuß des Bildschirms. Dort stehen die Funktionstasten, unter anderem steht dort F24=Weitere Tasten. Dies bedeutet, dass Du nach Druck auf F24 an dieser Stelle weitere Funktionstasten angezeigt bekommst, da nicht genug Platz für die Beschreibung aller Tasten vorhanden ist.

Drücke also nun einmal auf F24 und der Bildschirmfuss ändert sich. Nun interessieren wir uns für die Taste F11=Schlüsselwörter. Drücke einmal hierauf, und die Anzeige der Parameterfelder ändert sich. Mit dieser Taste kann man zwischen der Anzeige der Felder und der möglichen Werte, sowie der Anzeige der Parameter und deren Namen umschalten.

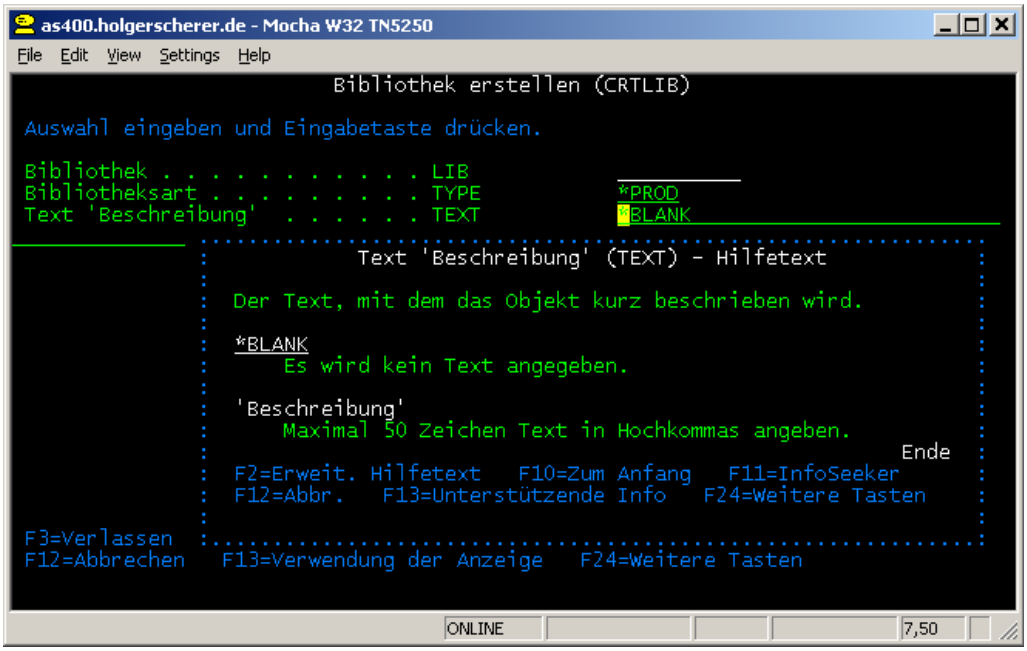


Abbildung 8 : Alle Parameter zum Befehl haben einen Hilfstext

Diese Beschreibung über die Oberfläche bei Befehlsparametern gilt generell für alle Systembildschirme mit Parametern oder Eingabefeldern. Ich denke, das Konzept ist verständlich, ich erspare mir ab nun die genauere Beschreibung, wie man mehr über einen Parameter erfährt.

Nach diesem Exkurs über die Oberfläche eines Befehls wollen wir nun eine Beispielbibliothek mit den drei besprochenen Parametern erstellen. Wie Du Dir vorstellen kannst, sieht der Befehl so aus:

```
Crtlib lib(test01) Text('Ich bin ein Test') DF
```

Den Parameter TYPE(...) haben wir ausgelassen, so dass das System den Standardwert *PROD verwendet.

Somit kannst Du also eine Bibliothek anlegen. Sollte nun eine Meldung am Bildschirmfuss erscheinen, dass die gewünschte Bibliothek bereits existiert, war ein anderer Leser dieses Manuals schneller. Ändere einfach den Namen der zu erstellenden Bibliothek. Für die Faulen unter uns bleibt der eingegebene Befehl in diesem Falle in der Befehlszeile stehen, so dass Du nicht einmal die F9-Taste bemühen musst. Einfach mit dem Cursor nach rechts gehen, den Namen ändern und ab geht's.

Ich habe vorhin schon geschrieben, dass nach Möglichkeit nicht hunderte verwaister Bibliotheken auf dem System rumliegen sollen, und Du für die Experimente Deine eigene Bibliothek verwenden sollst (Du kennst ihren Namen?).

Aufräumarbeiten

(Auch diesen Befehl habe ich für alle gesperrt.)
Bitte lösche nun die von Dir erstelle Testbibliothek. Dies ist ganz einfach. Löschen heißt auf Englisch „delete“. Abgekürzt auf drei Buchstaben lautet dies „`dlt`“. Damit fangen alle Löschbefehle an. Für eine Bibliothek lautet der Befehl logischerweise `DLTLIB`. Auch hier ist der Parameter `lib(…)` zur Angabe der betreffenden Bibliothek zwingend nötig. Die oben erzeugte Bibliothek wird somit wie folgt aus dem System entfernt:

```
dltlib test01
```

(Die Angabe des Parameternamens `lib(…)` kann man sich sparen, es gibt bei diesem Befehl nur diesen einen Parameter.)

Es gibt noch einige weitere Befehle, die sich mit Bibliotheken beschäftigen. Diese kommen zu einem späteren Zeitpunkt. Um aber das Publikum nicht zu sehr zu ermüden, mache ich jetzt mit Objekten in unserer Bibliothek weiter. Es sei angemerkt, dass Du ab nun mit Deiner eigenen Bibliothek arbeiten musst. Schreibe ich also in einem Befehl oder Parameter „`<DEINELIB>`“ so ersetzt Du dies bitte durch den Namen Deiner Bibliothek.

Ich hab doch Recht!

Auf meinem System ist eingestellt, dass neu erstellte Objekte zunächst für alle Benutzer zugänglich sind (warum, erzähle ich in einer anderen Geschichte).
Damit nun nicht jeder in Deinen Objekten rumfummeln kann, musst Du die Rechte darauf etwas anpassen. Auf der AS/400 kann man für jedes Objekt (auf das man natürlich selbst entsprechende Rechte haben muss), die Rechte für andere Benutzer ändern. Das kann man wie folgt machen:
Zunächst zeigt man sich das entsprechende Objekt an. Angenommen, Du hast eine Bibliothek LIBABC123. Dann gibst Du ein:

```
wrkobj libabc123
```

Es erscheint folgender Bildschirm:

```
                            Mit Objekten arbeiten

Auswahl eingeben und Eingabetaste drücken.
  2=Berechtigung editieren   3=Kopieren   4=Löschen   5=Berechtigung anzeigen
  7=Umbenennen   8=Beschreibung anzeigen   13=Beschreibung ändern

Opt  Objekt      Art       Bibliothek  Attribut    Text
 _   LIBABC123   *LIB      QSYS        PROD

                                                                        Ende
Parameter für Auswahlmöglichkeiten 5, 7 und 13 oder Befehl
===>
F3=Verlassen   F5=Aktualisieren   F11=Namen und Arten   F12=Abbrechen
F16=Neuer Listenanfang   F17=Listenanfang bei           F24=Weitere Tasten
```

An der markierten Stelle gibst Du die Auswahl `2` ein, um die Berechtigung zu editieren (der Befehl heißt `EDTOBJAUT` - Edit Object Authority):

```
                          Objektberechtigung editieren

Objekt . . . . . :   LIBABC123       Eigner . . . . . :   HOLLE
  Bibliothek . . . :     QSYS        Primärgruppe . . . :   *NONE
Objektart  . . . . :   *LIB

Aktuelle Berechtigungen ändern und Eingabetaste drücken.

  Objekt durch Berechtigungsliste geschützt . . . . . .   *NONE

                          Objekt-
Benutzer    Gruppe      berechtg.
HOLLE                   *ALL
*PUBLIC                 *EXCLUDE


                                                                        Ende
F3=Verlassen   F6=Benutzer hinzufügen   F12=Abbrechen   F24=Weitere Tasten

(C) COPYRIGHT IBM CORP. 1980, 1998.
```

Nun änderst Du den vorhandenen Eintrag *CHANGE für den Benutzer *PUBLIC (also alle, die sonst nicht genannt sind) auf *EXCLUDE, und schon darf keiner mehr ausser Dir an Deine Daten :-)

Datei oder nicht Datei?

Nachdem wir nun also eine Bibliothek haben, in der wir unsere Objekte unterbringen können, brauchen wir noch Dateien, in dem wir unsere Quellcodes unterbringen. Zur Verwirrung: Als Programmierer sortiert man seine Quellcodes, am besten nach dessen Typ. Die AS/400 kennt viele Programmiersprachen und weitere Typen von Quelldateien. Also erstellt man für jeden benötigten Typ von Quellcode eine Datei, und legt die Quellen für die einzelnen Programme in Teildateien ab. Klingt kompliziert, isses aber nicht ☺ Das hat nur den Vorteil, dass man etwas Ordnung schafft und die einzelnen Arten von Dateien sortiert. Man muss ja nicht immer alles in einem Haufen sammeln (es gibt Leute, die tausende Quelldateien an einem Platz haben...)

Quelldateien

Fangen wir mit einer ungewöhnlichen Art von Quelldateien an. Dass ein Programmierer den Text für ein Programm in eine Quelldatei schreibt, dürfte bekannt sein. Die AS/400 kennt aber auch Quelldateien für Datentabellen, Bildschirmdefinitionen und Druckausgaben. All diese Sachen werden in einem Quelltext beschrieben und dann in das entsprechende Objekt umgewandelt. Sicher könnte man eine neue Tabelle schnell mit einem passenden SQL-Statement erstellen, aber wie angedroht – das kommt später! ;-) Und auch die SQL-Statements kann man in Quelldateien ablegen.

Der Vorteil einer Quelldatei für Datentabellen ist, dass man im Nachhinein fröhlich in diesem Quelltext Änderungen vornehmen kann, ohne gleich die Tabelle zu ruinieren. Ausserdem ist dieser Quelltext sehr „plakativ“, das bedeutet, er beschreibt alles, was für die Erstellung dieser Datei nötig ist und ist somit übertragbar. Weiterhin kann man im Falle eines Falles anhand der Quellendatei die Datendatei schnell neu erstellen.

Die Beschreibung für eine Datenbankdatei nennt man DDS (engl. Data Definition Specification), oder auch Datendefinitions-Spezifikationen (tolles Wort, reinstes IBM-Deutsch).

In einer solchen Teildatei legt man die Spezifikationen für eine Datei ab. Datei sei hier ein generelles Wort für Datenbanktabelle, Bildschirmbeschreibung oder Druckausgabe-Beschreibung.

(Ermunterung für alle, die an PCs gewöhnt und nun total verwirrt sind: Es wird noch viel verwirrender, aber bleibt dran, es lohnt sich!)

Wie schon besprochen, rede ich hier von Quelldateien. In diesen lege ich viele Teildateien an, jede einzelne enthält die Beschreibung für eine Datendatei, ein Programm, ein Bildschirm oder was auch immer. Bedenke die Struktur: Eine Bibliothek enthält viele Dateien. Eine Datei kann eine oder mehrere Teildateien gleichen Typs enthalten.
Wir legen also in einer Datei unsere Teildateien für die Quellcodes an. Wem das nicht ganz geläufig ist, stelle sich eine Datei als Ordner vor, in dem die Teildateien wie z.B. Textdateien auf einem PC enthalten sind.

Deine eigene Quelldatei

Um nun eine Datei für diese Quellen-Teildateien anzulegen, gibt es den Befehl:

```
CRTSRCPF
```

Ausgeschrieben bedeutet dies: „create Source physical File“. „Create“ ist bekannt, „Source“ steht für „Quelle“ und „physical File“ bedeutet nur, dass wir eine Datei erstellen wollen, die wirklich Daten enthält.

Bitte diesen Befehl mit F4 prompten (oder DF drücken, hat hier den gleichen Effekt, OS/400 ist wissbegierig).

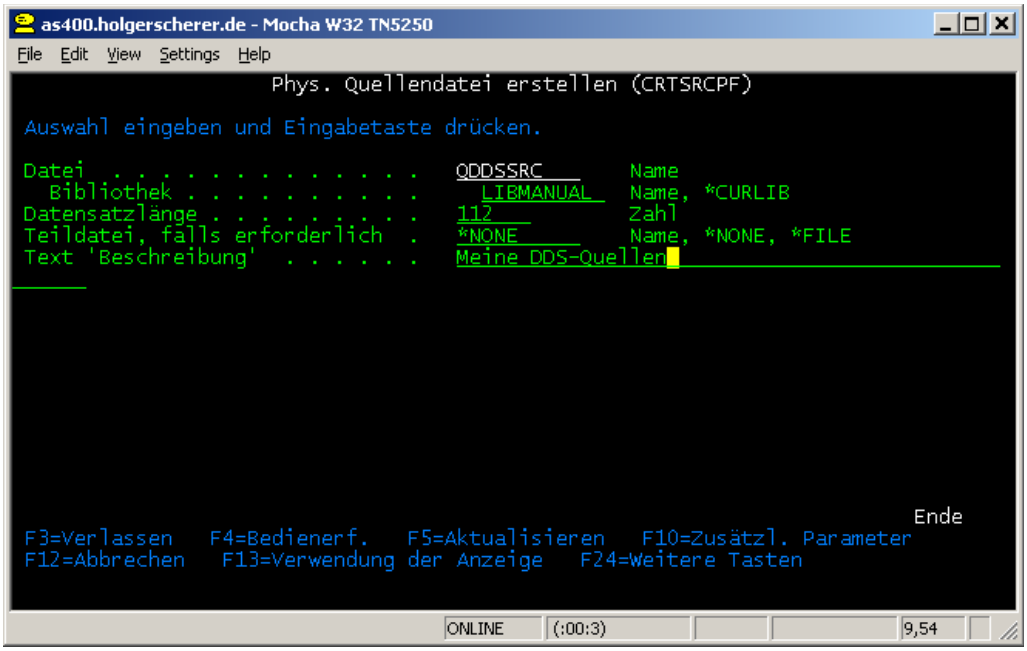


Abbildung 9 : Erstellen einer Datei, in der wir unsere Quellen ablegen

Ich habe in diesem Beispiel bereits die nötigen Parameter für die DDS-Datei eingetragen. Der Name der Datei lautet hier „QDDSSRC“, dies ist ein Standardname von IBM. Warum lege ich ein Objekt an, dessen Name mit Q beginnt? Nun, das hat sich irgendwie eingebürgert, fast jeder Programmierer benennt seine Dateien für Programmquellen nach den Namen, die IBM einmal vorgeschlagen hat. Das sind aber auch die einzigen Benutzer-Objekte, die mit dem Buchstaben Q anfangen. Natürlich könntest Du als Dateiname auch „MEINEDDS“ verwenden.

Als Bibliothek für meine Datei habe ich hier LIBMANUAL eingetragen, bitte wähle hier den Namen Deiner eigenen Bibliothek. Auf LIBMANUAL hast Du auf meinem System keinen Zugriff, da ich hier für das Handbuch Experimente durchführe.

Die Datensatzlänge von 112 bedeutet, dass jede Zeile max. 112 Zeichen lang sein kann. Da fast alle Quelldateien spaltenorientiert sind, hat diese Länge ihren Sinn. Überschreibe die vorgeschlagenen 92 durch `112` (damit haben wir später mehr Platz für Kommentare), und gib unter Text eine kurze Beschreibung für Deine Datei ein. „Meine DDS-Quellen“ ist eine gute Beschreibung.

Drücke nun auf DF, und Deine Quelldatei wird angelegt, eine Meldung am Fuß bestätigt dies.

Jetzt werden wir mit dieser Datei arbeiten und dort eine Teildatei anlegen. Wir lernen jetzt ein sehr leistungsfähiges Werkzeug auf der AS/400 kennen: Die Entwicklungsumgebung. Sie besteht aus mehreren Teilen, dem PDM (Program Development Manager) und dem SEU (Source Entry Utility).
Aber der Reihe nach…

Der PDM

Ein ehemaliger Kollege von mir (*Hallo, Bäähaarnd!*) bezeichnete den PDM mal als „den Norton Commander der AS/400“. Dies ist eigentlich gar nicht so falsch, denn er erleichtert uns den Umgang mit Dateien und Teildateien ungemein. Benutzen wir ihn aber zunächst einmal, um die Objekte auf der AS/400 zu durchforsten. Starte den PDM einfach mal mit:

```
strpdm
```

Der Befehl ist eigentlich nicht erklärungsbedürftig. Es erscheint ein Menu, in dem Du wählen kannst, mit welchen Objekten Du arbeiten willst, Du erkennst automatisch die Hierarchie der Objekte auf der AS/400. Beginnen wir damit, uns die vorhandenen Bibliotheken anzeigen zu lassen. Wähle den Menüpunkt „`1`“ aus. Im nachfolgenden Bildschirm wirst Du gefragt, mit welchen Bibliotheken Du arbeiten willst. Bitte gib den Sonderwert „`*ALL`“ ein und bestätige den Bildschirm. (Du könntest auch „`LIB*`“ eingeben und siehst alle Bibliotheken, deren Name mit LIB anfängt, und auf die Du zugreifen darfst.)

Nun kommt der Bildschirm „`Mit Bibliotheken arbeiten (mittels PDM)`“, der Dir alle Bibliotheken anzeigt. Natürlich nicht wirklich alle, sondern nur die Bibliotheken, auf die Du auch zugreifen darfst. Dazu gehören viele Systembibliotheken, einige Bibliotheken für alle, sowie Deine Bibliotheken. Die Objekte von anderen Benutzern sind in der Regel nicht für Dich erreichbar. Ausnahme: Andere Benutzer haben den Zugriff für alle anderen Benutzer erlaubt, unabsichtlich oder mit Absicht :)

Mit den Tasten BildAuf und BildAb kannst Du in der Liste blättern, wie Dir das Wort „`Weitere…`“ rechts unten am Bildschirm anzeigt. Du siehst jetzt eine übliche AS/400-Liste, in der Du blättern kannst. Wenn weitere Einträge vorhanden sind, steht entweder dieses Wort oder ein Pluszeichen unten rechts. Wenn Du am Ende der Liste bist und blätterst weiter, erscheint eine entsprechende Meldung. Auf diesen Hinweis verzichte ich in Zukunft, es ist ja auch leicht zu merken und gilt für alle Listen.

Nun wollen wir mit der vorhin erzeugten Quellendatei arbeiten. Du hast Dir ja hoffentlich gemerkt, in welcher Bibliothek Du Deine Datei angelegt hast. Blättere bitte nun in der Liste der Bibliotheken, bis Du einen Eintrag für Deine Bibliothek findest. Wir wollen nun sehen, welche Objekte sich in dieser Bibliothek verstecken. Um mit dem Inhalt dieser Bibliothek zu arbeiten, geben wir die Auswahl „12“ in das Auswahlfeld vor dem Bibliotheksnamen ein. Warum Auswahl 12? Schau mal auf den Fuß des Bildschirms; mit der Taste F24 kann man sich wie so oft weitere F-Tasten anzeigen lassen, wenn aus Platzgründen nicht alle angezeigt werden können. Analog dazu kann man mit der Taste F23 weitere *Auswahlen* ansehen, wenn sie nicht alle entsprechenden Platz finden. Diese beiden Tasten existieren in vielen Bildschirmen.

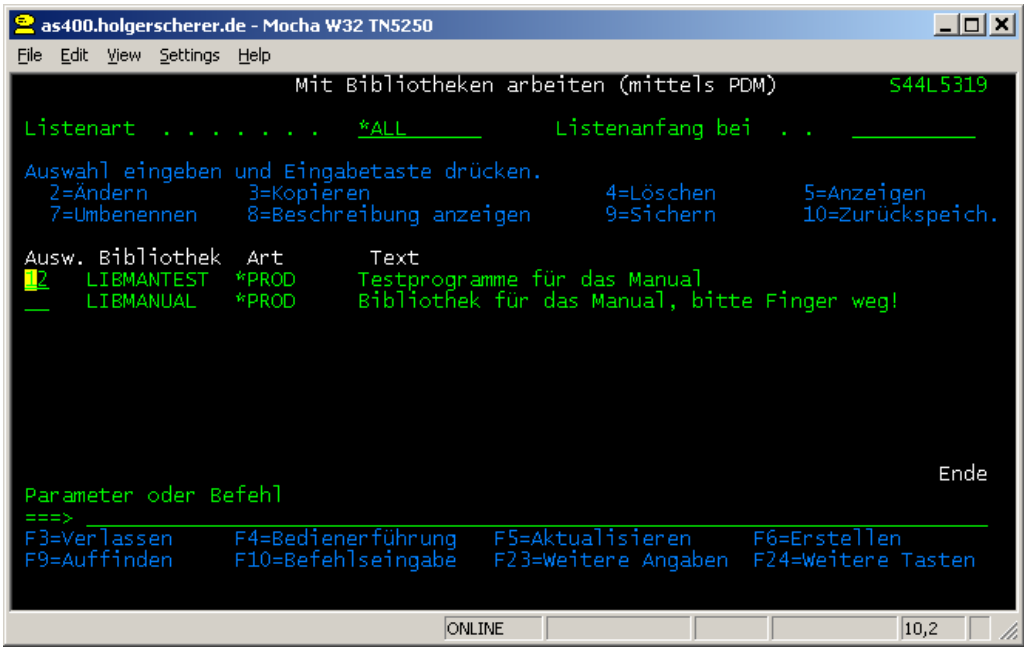


Abbildung 10 : Im PDM kann man mit Auswahl 12 sehen, was in einem Objekt enthalten ist

Also gib' – wie im obigen Bild gezeigt – die Auswahl 12 vor Deinem Bibliotheksnamen ein und drücke DF, das Bild wechselt sofort auf „`Mit Objekten arbeiten (mittels PDM)`“. Dieser Bildschirm sieht fast genauso aus wie der eben angezeigte, die Funktionen sind gleich.

Du siehst nun, dass in Deiner Bibliothek genau ein Objekt existiert, und zwar die von Dir angelegte Quellendatei. (Sollten dort mehrere Objekte existieren, so sind das die Objekte, die ich Dir beim Erstellen Deines Accounts angelegt habe, damit Du weniger Arbeit hast). Was man mit diesen Objekten alles anstellen kann, werden wir später sehen. Nun habe ich vorhin geschrieben, dass man in einer Datei so genannte Teildateien anlegen kann; und bei Quellen für Programme und Dateien dies genau so macht. In der Datei `QDDSSRC` legt man verschiedene Quellen für Datendateien an. Um mit den Teildateien innerhalb dieser Datei zu arbeiten, geht man genauso vor wie eben bei der Bibliothek. Also einfach vor den Eintrag unserer Quellendatei wieder die Auswahl 12 eingeben und bestätigen. Wir sehen nun den Bildschirm „`Mit Teildateien arbeiten (mittels PDM)`“, dessen Layout uns sofort bekannt vorkommt, mit der Ausnahme, dass in der Objektliste nur lapidar steht „`(Keine Teildateien in Datei)`“. Dies ist auch logisch, da wir diese Datei erst angelegt haben und sie somit leer ist.

Nun geht es Schritt für Schritt, Du machst jetzt Bekanntschaft mit dem SEU, dem Source Entry Utility. Dies ist ein gar merkwürdiger, aber leistungsfähiger Editor, Unix-Leute, die auf den Editor „vi“ stehen, werden Verständnis für „SEU“ haben. Alle anderen: lasst nun nicht alle Hoffnung fahren. Der Editor ist für Quelldateien besser geeignet als jedes Word mit Mausbedienung <grins> und wenn man sich damit etwas beschäftigt, ist er ein tolles Werkzeug. Zuerst eine Übung.

(dieser Raum ist absichtlich leer :-)

SEU – Die Eier legende Wollmilchsau

Nun wollen wir also eine neue Teildatei erstellen und dort was reintippen, um uns mit dem Editor vertraut zu machen. Am Fuß des Bildschirms siehst Du die Angabe „`F6=Erstellen`“. Dies hört sich für unsere Zwecke ganz gut an, also benutzen wir diese Funktionstaste. Es erscheint das Bild „`SEU starten`“, und es werden ein paar Parameter gefragt. SEU muss nun wissen, wie die Teildatei heißen soll, die wir anlegen wollen, und welchen Typs sie ist. Bitte fülle die Parameter wie angezeigt aus.

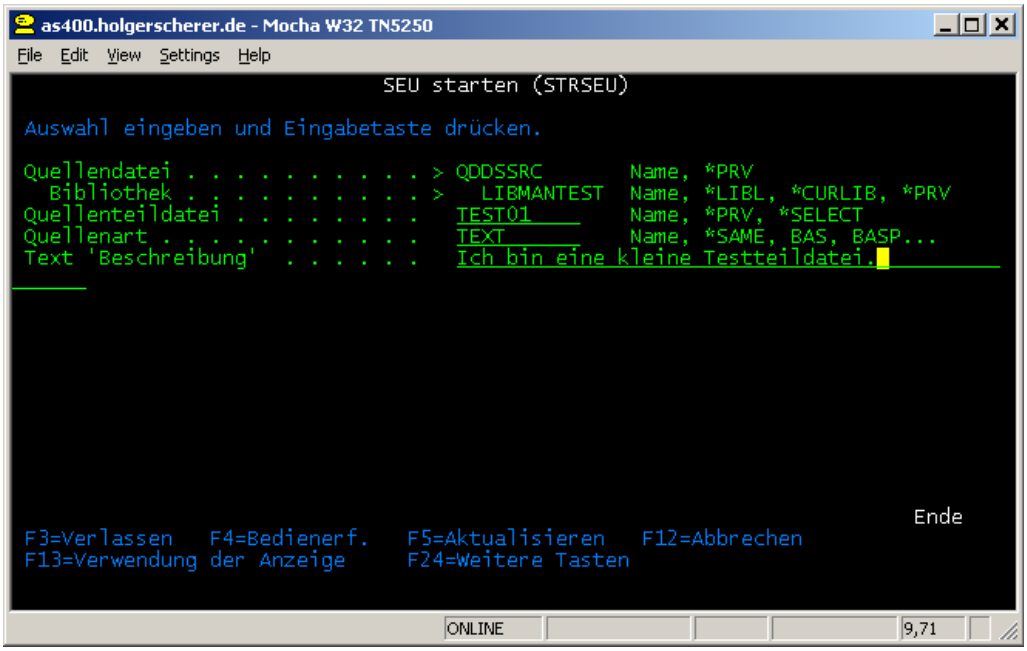


Abbildung 11 : Starten des SEU mit dem Wunsch, eine neue Teildatei zu erstellen

Wir erstellen also eine Teildatei namens „`test01`“. Da SEU ein Editor ist, der automatisch eine Syntax- und Formatprüfung vornimmt, wir aber erst mal einen kleinen Schreibkurs vornehmen wollen, geben wir als Quellenart „`TEXT`“ an. Diesen Dateityp prüft SEU nicht und nervt somit auch nicht mit Fehlermeldungen. Die Eingaben wie oben angezeigt bestätigen, und Du landest im Editorbildschirm.

Damit Du nicht ganz verloren bist, erst mal eine Orientierungshilfe:

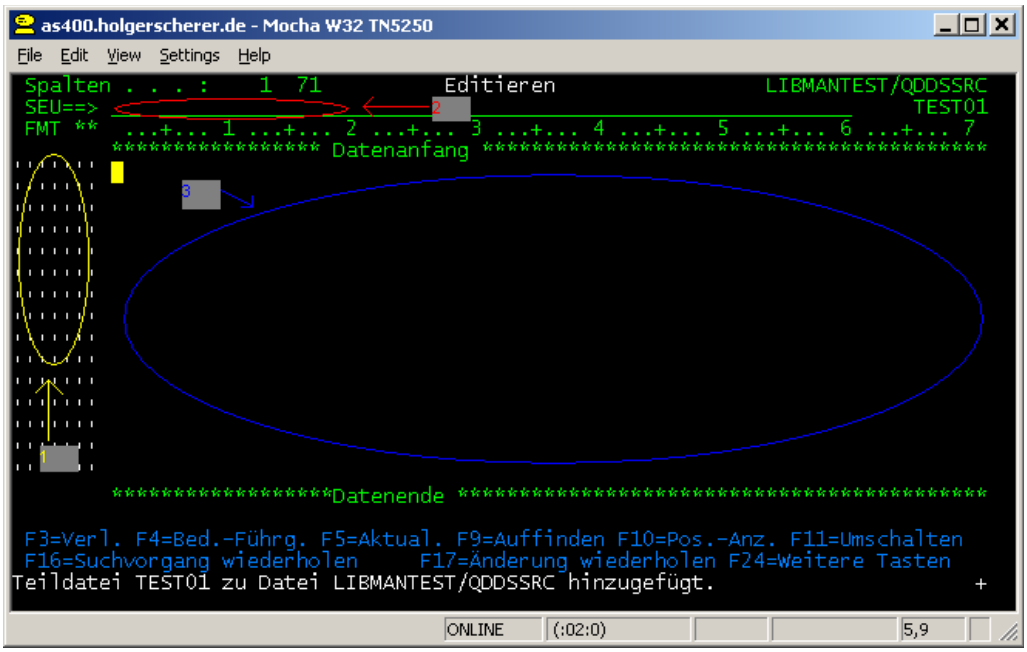


Abbildung 12 : Aufteilung des Editierbildschirms

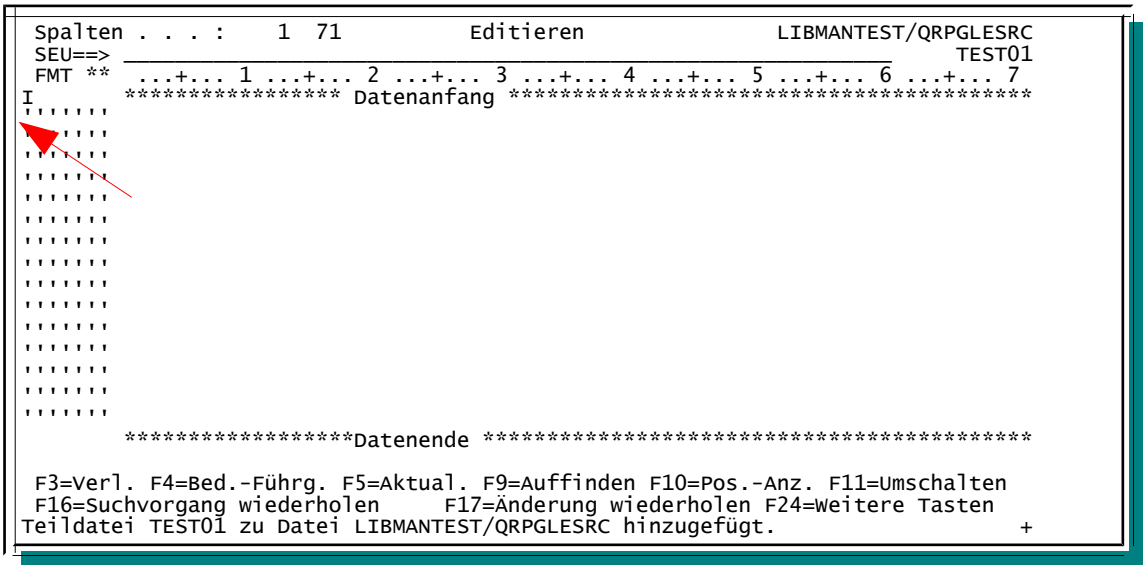
Der Bildschirm von SEU ist in mehrere Bereiche unterteilt. Links ist der Bereich für Zeilennummern und Zeilenbefehle (markiert mit einer gelben „1“). Oben ist die Befehlszeile („2“ rot markiert). Den Hauptteil des Bildschirms nimmt der Bereich für die Zeilendaten (blau mit „3“ markiert) ein. (Man möge mir das „Kunstwerk“ verzeihen :)

Der Cursor befindet sich im Zeilennummernbereich der ersten Zeile und wartet darauf, dass wir etwas tun Dies wollen wir auch sogleich erledigen, können dies aber noch nicht.

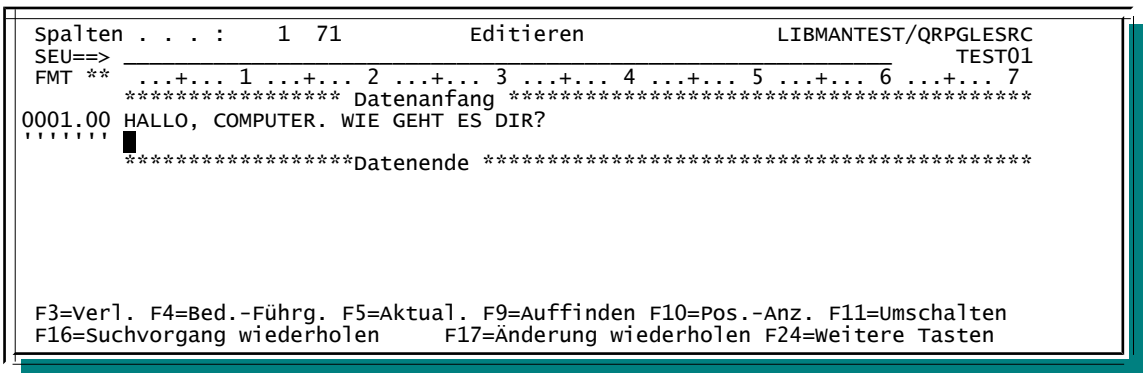
Zeile für Zeile

Da SEU zeilenorientiert arbeitet und nach dem Starten nicht im Einfügemodus läuft, müssen wir dem Programm sagen, dass wir weitere Zeilen einfügen wollen. Hiermit kommen wir zu den leistungsfähigen Zeilenbefehlen. Diese werden links im Bereich der Zeilennummern eingegeben. Dieser ist bei uns noch leer, da SEU keine Zeilennummern bisher vergeben hat.

Springe mit dem Cursor auf die erste Spalte, also in den Bereich der Zeilennummern (derzeit befindet sich der Cursor wohl in der ersten Spalte im Datenbereich). Gib dort ein „I“ (für Insert) ein und drücke auf DF. Damit sagst Du SEU, dass Du nach der aktuellen Zeile weitere Zeilen eingeben willst (siehe Pfeil):



SEU verändert die Darstellung. Der leere Bereich wird entfernt und eine leere Zeile eingefügt. Der Cursor wird an den Anfang des Datenbereichs gesetzt. Dort schreiben wir „Hallo, Computer. Wie geht es Dir?“ (ohne eine Eingabetaste zu drücken). Sollten Deine Eingaben nur in Großbuchstaben erscheinen, nicht verzagen. Wir kommen später dazu. Nun wollen wir etwas mehr Text eingeben. Wenn Du nun auf DF drückst, fügt SEU automatisch die nächste leere Zeile ein und setzt den Cursor wieder an den Zeilenanfang. Beachte: Der Zeilenanfang ist nicht in der Zeilennummer zu finden :-) Eine Syntaxprüfung findet bei Teildateien vom Typ `TEXT` nicht statt, also interessiert es den Editor nicht, was wir eingegeben haben. Nur wenn Du nun mit der `Enter`-Taste (oder Return auf deinem Computer, also nicht DF) in die nächste Zeile springen willst, landest Du im Zeilennummernbereich, wo „`***Datenende***`“ steht. Dort könntest Du auch Zeilenbefehle eintippen.
Dein Bildschirm müsste nun so aussehen:



Wir üben tippen!

Nun kannst Du weitere Zeilen eingeben. Damit wir jetzt mit dem gleichen Text arbeiten, tippe bitte folgende Zeilen ab und drücke nach jeder Zeile die DF-Taste. Solltest Du eine falsche Taste getippt haben und der Einfügemodus ist nicht aktiv, einfach mit einem `I` im zeilennummernbereich wieder aktivieren.

```
Ich bin eine Testzeile
Dieser Editor ist sehr merkwürdig.
Aber vielleicht kapiere ich das Programm doch noch.
The quick brown fox jumps over the lazy dog.
Jetzt schreibe ich in Zeile 6.
Und hier endet unser Texttext!
```

Dein Bildschirm sieht nun so aus:

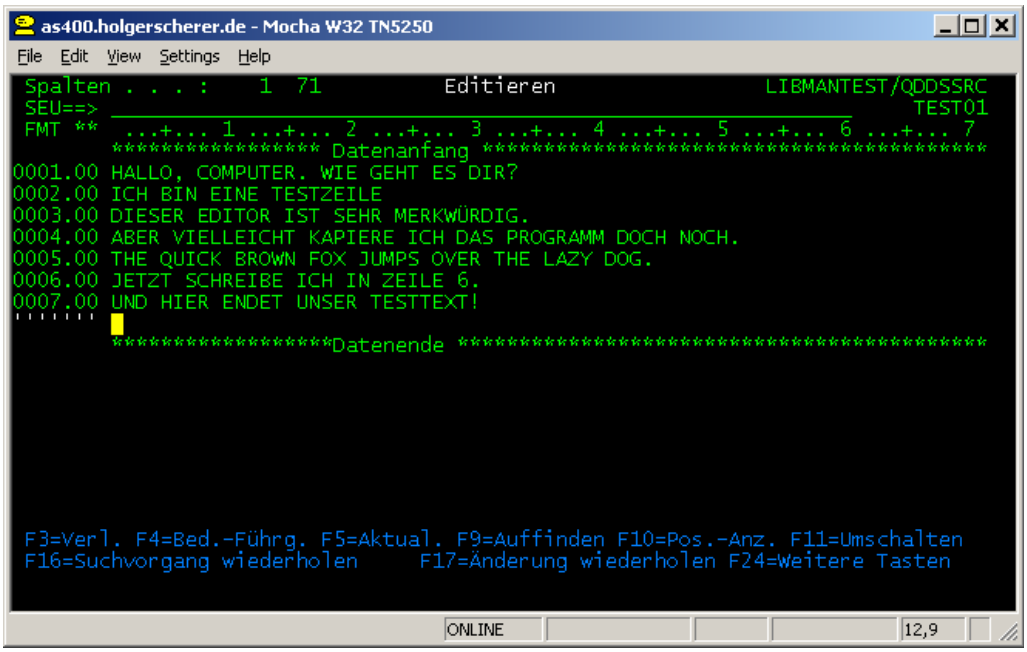


Abbildung 13 : Du hast nun ein paar Testzeilen eingetippt

Nun bist Du immer noch im Einfügemodus, dies erkennst Du an den weißen Punkten unterhalb von Zeile `0007.00`, außerdem steht der Cursor noch im Datenbereich.

Wenn Du nun noch einmal auf DF drückst, wird der Einfügemodus beendet und die leere Zeile gelöscht. Nun füge bitte unterhalb der Zeile `0003.00` eine Zeile ein. Dazu wie bekannt auf die Zeilennummer gehen und ein „`I`“ tippen.

Dies muss nicht unbedingt am Zeilenanfang passieren, aber Du musst innerhalb der Zeilennummer, also den ersten 7 Spalten des Bildschirms bleiben. Den Zeilenbefehl mit DF bestätigen und folgende Zeile tippen:

```
Aber der Editor kann tolle Sachen machen! :-)
```

Und die Zeile wieder mit DF absenden, sowie mit einem weiteren DF den Einfügemodus beenden. Bitte betrachte die Zeilennummerierung, es fällt auf, dass die eingefügte Zeile hinter dem Punkt hoch gezählt wird.

Weg damit!

So einfach wie das Einfügen geht das Löschen einer Zeile. Wir wollen nun Zeile `4.00` (ich spare mir die führenden Nullen) löschen, da sie uns nicht gefällt. Einfach in die Zeilennummer den Befehl „`D`“ für „`DELETE`“ tippen und bestätigen (ich gehe nun davon aus, Du weißt, dass man einen Befehl oder eine Zeile mit DF bestätigt). Schon ist die Zeile weg! Übrigens aufpassen: Wenn Du einen Zeilenbefehl eingegeben hast, wird dieser auch mit den Blättern-Tasten bestätigt! Solltest Du aus Versehen mal einen Befehl eingegeben, aber noch nicht DF gedrückt haben, hilft Dir die Taste F5.

Dies waren zwei einfache Zeilenbefehle, die aus nur einem Buchstaben in einer Zeile bestanden. Ich konzentriere mich zunächst auf diese Befehle.

Innerhalb einer Zeile kann man wie gewohnt Buchstaben löschen und einfügen (mit der entsprechenden Taste). Wenn Du ab einer bestimmten Stelle einer Zeile den Rest löschen willst, einfach den Cursor auf das erste zu löschende Zeichen setzen und Enter, also die Returntaste drücken. Gehe in Zeile `0005.00` hinter das Wort „`jumps`“ und drücke Enter, schon ist der Rest weg. Sollte Dir dies versehentlich passieren, keine Sorge. Du kannst Deine Änderungen rückgängig machen, solange Du Deine Änderungen (auch über mehrere Zeilen) noch nicht bestätigt oder die Blättertasten benutzt hast. Drücke nun F5, und die ursprünglichen Worte hinter „`jumps`“ erscheinen wieder. Probiere es aus!

Mehrzeilige Befehle

Kommen wir nun zu den mehrzeiligen Befehlen, die den Reiz des Editors ausmachen. Ich hoffe, Du hast beim Experimentieren den Text nicht zu sehr modifiziert, daher hier noch mal das Bild, von dem ich ausgehe:

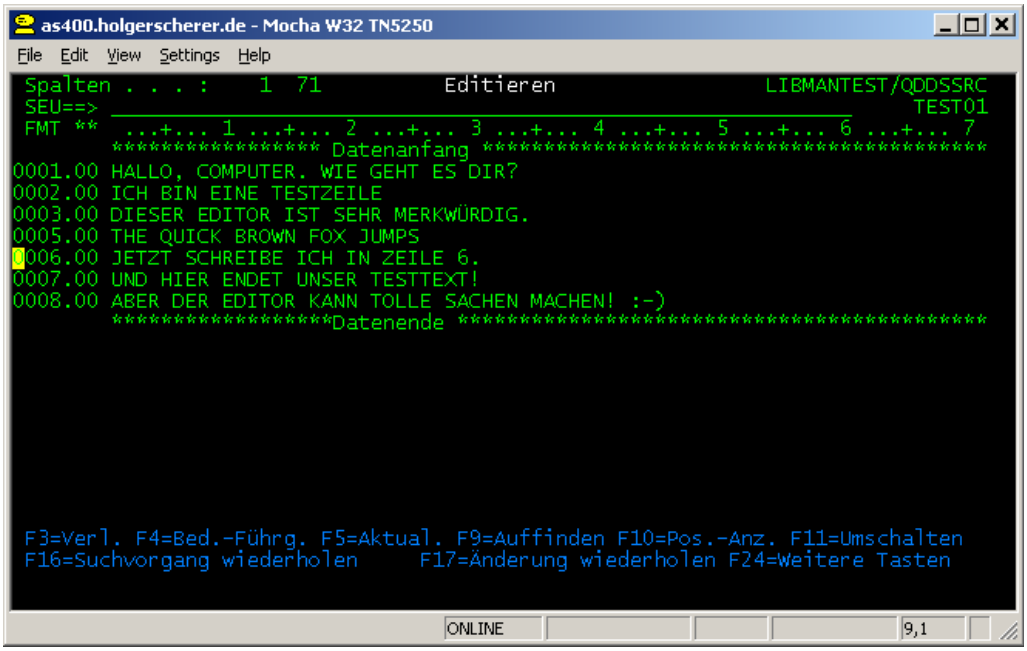


Abbildung 14 : Von diesen Zeilen gehe ich nun aus

Wir wollen nun die Zeile `2.00` hinter die Zeile `6.00` kopieren. Dazu springst Du mit dem Cursor in die Zeilennummer der zu kopierenden Zeile (also `2.00`), schreibst ein „*C*“ für „copy“ (ohne zu bestätigen), springst dann in die Zeilennummer, *nach* der die Zeile eingefügt werden soll (also `6.00`) und schreibst ein „*A*“ (für After) und bestätigst dann die beiden Befehle.
Bei Zeilenbefehlen, die in mehreren Zeilen eingegeben werden, kannst Du ruhig blättern, solange Du den zweiten Zeilenbefehl noch nicht eingegeben hast. Somit kann man auch größere Blöcke bearbeiten. Oben rechts steht, welcher Zeilenbefehl in Arbeit ist. Bearbeitet wird erst, wenn Du DF drückst.

Üben wir das mal...

So gibst Du den mehrzeiligen Befehl ein:

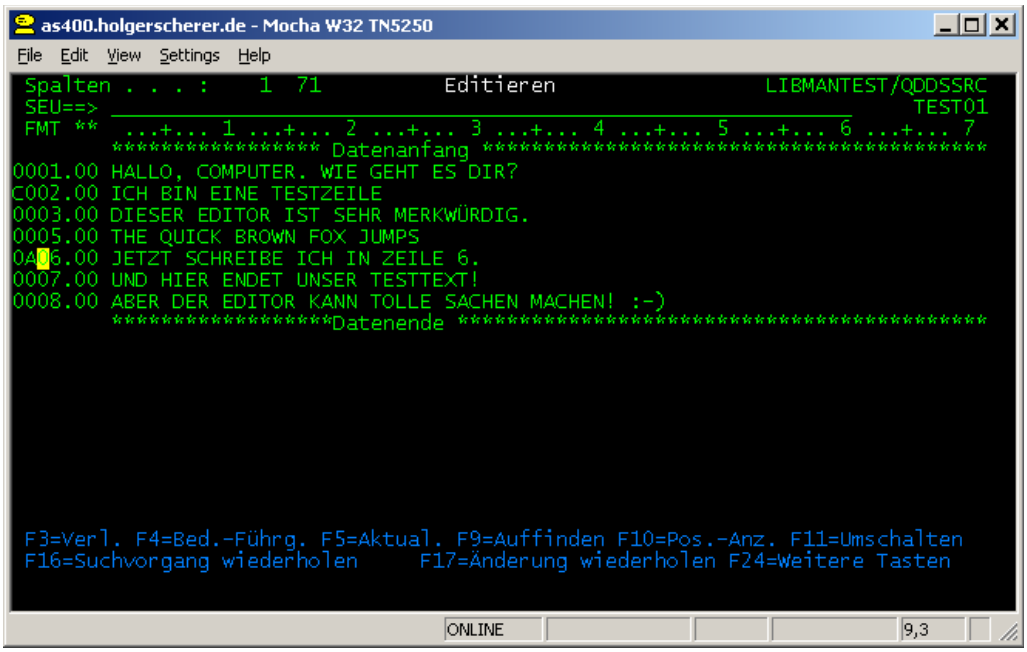


Abbildung 15 : Mehrzeiliger Befehl in Zeile 2 und 6

Schon haben wir eine zweite Testzeile. Und wenn Du das `A` durch ein `B` ersetzt, wird die zu kopierende Zeile *vor* (before) der Zielzeile eingefügt. Einfach, nicht?

Dieses gilt auch, wenn Du mehrere Zeilen auf einmal kopieren willst. Lösche bitte die zweite Testzeile (`6.01`) wieder, so dass obiges Bild entsteht. Nun wollen wir die Zeilen `2.00` HALLO… bis `5.00` THE… hinter die Zeile `7.00` UND HIER… kopieren. Diesen zu kopierenden Block markieren wir mit jeweils zwei „CC“ in der ersten und letzten Zeile des Blocks. Also springe in die Zeilennummer `2.00`, schreibe `CC` rein, springe in `5.00` und gib dort auch `CC` ein. Danach springe in `7.00`, gib dort das `A` für „After“ ein und bestätige Deine Befehle. Schon haben wir ein paar zusätzliche Zeilen. Die Zielzeile muss nicht mit zwei AA gekennzeichnet werden.

Löschen kannst Du mehrere Zeilen, in dem Du in der ersten und letzten zu löschenden Zeilennummer den Befehl `DD` eingibst und mit `DF` bestätigst. (IBM spricht hier von „Folgenummern“, da sie die Ablauffolge der Datei angeben, ausser dem liefen die Programme auf Lochkarten früher in dieser Reihenfolge ab.)

Es gibt noch einige weitere Zeilenbefehle. Am besten, Du drückst, wenn der Cursor innerhalb einer Folgenummer steht, einfach mal die Hilfetaste, also `F1`.

Bevor Du gleich richtig losexperimentieren darfst, noch ein paar Worte zur Befehlszeile oben. Dort kannst Du weitere Befehle eintippen. Ein Beispiel: Du suchst in Deinem Text die Zeichen „OVER“. Dazu gibst Du einfach in die Befehlszeile oben ein:

```
F over
```

und bestätigst dies mit `DF`. Der Befehl lautet „FIND“ gefolgt vom zu suchenden Text, und kann mit dem Buchstaben `F` abgekürzt werden.

Ebenso kann man einen Text ersetzen, in dem man eingibt:

```
c quick slow
```

Damit wird das erste vorkommende „quick“ durch „slow“ ersetzt (Change). Im Fuß findest Du eine Funktionstaste, mit der Du das Suchen oder Ersetzen wiederholen kannst. Wenn Du eine genaue Beschreibung zu einem Befehl lesen willst, tippe nur den Befehl (z.B. „C“) in die Befehlszeile und drücke die Hilfetaste, anstelle zu bestätigen. Ich empfehle die Lektüre der Hilfetexte. Sie sind recht ausführlich beschrieben, so dass ich dies nicht hier im Manual wiederholen muss (ich bin ja auch faul.)

Ich überlasse Dich nun ein wenig Deinem Experimentier- und Lesedrang. Wenn Du fertig bist, kannst Du den Editor mit F3 verlassen. Ich warte solange ein wenig. Wenn Du weitere Testdateien anlegst, lösche diese nachher wieder, bevor die Datei QDDSSRC unübersichtlich wird ☺

Wenn Du fertig geübt hast, wollen wir langsam mit dem Ernst des Lebens beginnen, äh, mit den Dateien und dem Programmieren und allem drumherum.

Noch ein Hinweis: Ein paar Seiten früher habe ich den Befehl CRTSRCPF beschrieben, mit dem man eine physische Datei für Quellen erstellt. Hier gibt es eine Angabe der Datensatzlänge. Von der dort angegebenen Zahl musst Du 12 abziehen, dann erhälst Du die max. Anzahl Zeichen, die Du im SEU pro Zeile eingeben kannst. Sechs stellen werden für die Zeilennummern links abgezogen, und 6 Stellen für das Datum.
Welches Datum? Drücke mal die Funktionstasten F20 / F19 (rechts / links blättern) aus! :-)

Die Tabelle

Da bist Du ja wieder. :-)

Strukturierte Daten

Erstellen wir nun die Beschreibung für eine Datendatei, die Adressen enthalten soll.

Spätestens nun überlegen wir uns, wie die Tabelle der Adressen aussehen soll. Wegen der Übersicht bleiben wir bei einer sehr einfachen Struktur, bei späteren Experimenten darf der Anwender natürlich versuchen, bis an die Grenzen zu gehen. Bevor wir also eine Teildatei für die Quelle der Adressdatei anlegen, notieren wir uns kurz, wie unsere erste Übungsdatei aussehen soll. Ich wähle mal folgendes Darstellungsbeispiel, das keinen Anspruch auf Vollständigkeit und hervorragendes Datenbankdesign erhebt.

Dateiname: `ADRESSEN`

Feldname	*Feldtyp*	*Beschreibung*
`ADRNR`	Num 10/0	Adressnummer, ganzzahlig, eindeutig (UNIQUE)
`ADRANRE`	Text 20	Anredetext
`ADRVNAME`	Text 35	Vorname
`ADRNNAME`	Text 35	Nachname
`ADRGEB`	Num 8/0	Geburtstag, Datumsformat als Ganzzahl
`ADRSTR`	Text 35	Straße
`ADRPLZ`	Text 10	Postleitzahl
`ADRORT`	Text 40	Ort
`ADRTEL`	Text 25	Telefonnummer
`ADRMAIL`	Text 50	e-Mail-Adresse

Tabelle 1 - Aufbau unserer ersten Testtabelle

So, ein paar kurze Worte zur Notation der Feldtypen. Der Typ „Num“ gibt eine Zahl an, die Zahl vor dem Schrägstrich die Anzahl der Stellen, die Zahl nach dem Schrägstrich gibt an, wie viele Stellen von der Gesamtzahl als Dezimalstellen verwendet werden sollen. Das UNIQUE kennzeichnet, dass dieses Feld zu einem eindeutigen (unique) Schlüssel gehört.
Textfelder werden mit ihrer Länge angegeben. Das Datum habe ich als Zahl definiert, dies hat ein paar Vorteile für einfache Sortierungen und Vergleiche. Normalerweise definiert man aber ein solches Feld als Datumsfeld, und später werden wir uns auch ansehen, was diese Methode für schöne Vorteile bietet.

Nun wollen wir die Tabelle mit dieser Struktur erstellen. Dazu ist es nötig, diese Beschreibung in eine Quellenteildatei einzugeben. Diese Teildatei wird später verwendet, damit die AS/400 die daraus resultierende Datendatei erstellen kann. (Dies ist das Grundprinzip der inzwischen als „veraltet“ angesehenen Vorgehensweise via DDS. Man erstellt auch für eine Datendatei eine Quelle, diese wird übersetzt und das System erstellt daraus die Datendatei, bzw. ändert eine vorhandene. Man kann die DDS-Quellendatei sich ähnlich eines SQL-Befehls vorstellen.)

Zunächst erstellen wir nun mit der der uns bereits bekannten Funktionstaste F6 eine neue Teildatei. Diesmal vom Typ „PF“. PF steht für „physical File“, also eine Datei, die physikalische Daten enthält. Das Gegenstück dazu ist eine „Logical File“, eine logische Datei, oder in SQL-Notation ein View. Ab hier kommt erst mal ein platzsparenderes Format für Bildschirmfotos, bevor das Manual viele MB gross wird.

```
                                SEU starten (STRSEU)

Auswahl eingeben und Eingabetaste drücken.

Quellendatei . . . . . > QDDSSRC          Name, *PRV
  Bibliothek . . . . . >   LIBMANUAL      Name, *LIBL, *CURLIB, *PRV
Quellenteildatei . . . . . > ADREPF        Name, *PRV, *SELECT
Quellenart . . . . . > PF          Name, *SAME, BAS, BASP…
Text `Beschreibung`. . . . . .   ADRESSEN -physisch_____________________________
__________




Ende
F3=Verlassen   F4=Bedienerf.   F5=Aktualisieren   F12=Abbrechen
F13=Verwendung der Anzeige     F24=Weitere Tasten
```

Start des SEU für eine Datendatei

Für den Namen der Teildatei, der auch später Name der Datendatei wird, nehmen wir ein Kürzel, bestehend aus vier Buchstaben (ADRE), die die Datei kennzeichnen, sowie der Endung „PF“, die eine physische Datei symbolisiert. Natürlich könntest Du auch als Namen „ADRESSEN“ vergeben, allerdings sollte man sich ein Schema überlegen, das bei vielen physikalischen und logischen Dateien ein Auffinden vereinfacht.

Die Quellenart ist jetzt nicht TEXT, wie im vorigen Übungsbeispiel, sondern PF für eine physische Datei. Die Beschreibung kannst Du wie immer frei wählen.

Wenn Du nun diesen Bildschirm wie üblich bestätigst, startet SEU wie gewohnt mit einer leeren Datei. Wenn Du allerdings genau hinschaust, fällt Dir oben am Bildschirm, unterhalb der Befehlszeile, eine Zeile mit vielen Kürzeln auf. Diese Zeile ist links mit „FMT PF“ gekennzeichnet. Damit zeigt SEU Dir an, dass er weiß, welchen Dateityp Du eingeben willst und bietet entsprechende Eingabehilfen an. Ebenso wird geprüft, ob Deine Eingaben Sinn machen. Wenn nicht, oder bei einem Tippfehler, wird die falsche Zeile invers angezeigt.

Du wirst nun die Erfahrung machen, dass alle Quellen, die man mit dem SEU eingibt, spaltenorientiert formatiert werden. Dies hat seinen Ursprung aus der Zeit der Lochkarten, die ebenso spaltenorientiert aufgebaut waren. Man muss sich etwas daran gewöhnen, allerdings ist dies stellenweise der Übersicht recht förderlich.

Dateitypen

Jeder Dateityp, den man eingibt, hat seine eigenen Formatierungen und spezielle Spalten und Zeilenarten (auch Bestimmungsarten genannt). Damit man nicht schon am Anfang entnervt aufgibt, hält SEU einige Formatierungs- und Eingabehilfen bereit.

Schau Dir mal das unten stehende Beispiel an:

```
 Spalten . . . :   1  71          Editieren                  LIBMANUAL/QDDSSRC
 SEU==> _____________________________________________________________    ADREPF
 FMT PF .....A.........A.Name++++++RLängeDDsF.....Funktionen+++++++++++++++++
0001.00      A                                    UNIQUE
0002.00      A          R ADRFMT
0003.00      A            ADRNR          10S 0    TEXT(´Adressnummer´)
0004.00      A            ADRANRE        20A      TEXT(´Anrede´)
0005.00      A            ADRVNAME     0035A
0006.00      A            ADRNNAME       35A
0007.00      A            ADRGEB          8S 0
0008.00      A            ADRSTR         35A
0009.00      A            ADRPLZ         10A
0010.00      A            ADRORT         40A
0011.00      A            ADRTEL         25A
0012.00      A            ADRMAIL        50A
0013.00      A          K ADRNR
       ******************Datenende*****************************************


F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

So sieht eine Tabellendefinition als DDS-Liste aus

Du siehst, wie unsere oben beschriebene Tabelle als DDS auf der AS/400 definiert wird. Zunächst möchte ich kurz die Spalten darstellen, wie sie in der Formatzeile „FMT PF“ angezeigt sind.

Das bedeutet, dass das erste Definitionsfeld A in Spalte 6 steht (zähle die Punkte in der dritten Zeile!) Dieses A steht für die Spalte der Spezifikationsart. Diese ist bei einer Datendatei immer „A“ (macht die Sache einfacher).

Das nächste Feld A in Spalte 17 kennzeichnet, um was für einen Namen es sich bei der eingegebenen Zeile handelt (sofern ein Name definiert wurde). Man nennt diese Spalte auch „Namensart“.

Der NAME steht in den Spalten 19 bis 28, also zehn Stellen. Hier definiert man einen Feldnamen, oder – wenn ein der Spalte „Namensart“ ein R eingegeben wurde, den Namen eines Satzformates.

Das Feld in Spalte 29, mit einem R überschrieben, kann analog mit einem „R“ belegt werden, wenn die Definition des Namens von einer anderen Datei übernommen wird (wird später erläutert). Man spricht von der Referenz.

Nun folgt die Angabe der „Länge“, wenn Du ein Feld definierst (Spalten 30-34). Innerhalb dieser Spalte musst Du die Längenangabe *rechtsbündig* eintragen, wie im obigen Beispiel ersichtlich. Im Feld ADRVNAME habe ich die Länge mit vorangestellten Nullen eingegeben, damit Du den Feldbereich erkennst. Sie müssen aber nicht eingegeben werden.

In Spalte 35 steht das Feld „D“. Dies kennzeichnet die Datenart. Ich habe im Beispiel ein „A“ für alphanumerisch, oder ein „S“ für gezonte dezimale Daten verwendet. Gezonte Dezimale Daten ist eine Variante, Zahlen darzustellen. Hierbei wird definiert, dass von den angegebenen Stellen ein Teil als Nachkommastellen zu betrachten ist.

Solltest Du numerische Daten definieren, musst Du in der folgenden Spalte „Ds“ die Zahl der Dezimalstellen für die Definition angeben (im Beispiel jeweils Null, *rechtsbündig* eingetragen).

Nun folgt das Funktionsfeld „F“, welches wir zunächst leer lassen. Als letztes findest Du rechts den Funktionsbereich. Hier kannst Du eine oder mehrere Zusatzangaben zum Feld oder zur Definition machen.

Spezifische Spalten

Puh… Fangen wir oben an. Ich erkläre Zeile für Zeile.

Die Spezifikation in Zeile 1 beinhaltet nur die Funktion „UNIQUE“, damit erklären wir dem System, dass wir eine Datei mit einem eindeutigen Schlüssel erstellen wollen. Dabei übernimmt OS/400 die Duplikatprüfung und kann uns auf die Finger klopfen, wenn wir versuchen, beim Schreiben in die Datei doppelte Schlüssel zu erzeugen.

In Zeile 2 definiere ich als Namensart ein „R“. Damit kennzeichne ich den Beginn eines Satzformates, verständlicher Beschrieben als Record. Mit DDS kann man für ein und die selbe Datendatei mehrere Satzformate definieren. Damit kann man zum Beispiel in einem bestimmten Satzformat nur einige Felder aus der Datendatei dem Programm unter diesem Namen zugänglich machen. Man kann dies mit Views unter SQL vergleichen. Ich benenne dieses Satzformat „ADRFMT“.

In der dritten Zeile beginne ich mit dem ersten Feld. Die Namensart ist wieder leer. Im Namensfeld steht der Feldname „ADRNR“. Darauf folgt in der Spalte „Länge“ die Angabe, dass dieses Feld 10 Stellen lang sein soll. Es ist hier unbedingt darauf zu achten, dass das Feld „Länge“ unterhalb des Buchstabens „L“ anfängt, ich trage allerdings die Zahl rechtsbündig ein. Alternativ hätte ich auch „00010“ schreiben können, dies ist aber der Lesbarkeit etwas abträglich.

Im Feld für die Datenart habe ich ein „S“ eingetragen, dies kennzeichnet gezonte Dezimalzahlen. Über den Aufbau der Zahlenarten werde ich später berichten. Damit ich ein ganzzahliges Feld erhalte, gebe ich bei den Dezimalstellen eine Null rechtsbündig ein. Wenn ich ein Zahlenfeld definiere, muss ich zwingend die Anzahl der Dezimalstellen eingeben.

Letztlich folgt als Funktion eine lesbare Beschreibung für dieses Feld. Dieser Parameter TEXT() wird vom System später als generelle Feldbeschreibung verwendet.

In der vierten Zeile definiere ich das Feld „ADRANRE“, es handelt sich hierbei um ein 20stelliges Alpha-Feld für die Anrede. Es wird wie das erste Feld definiert, beachte hierbei, dass keine Angabe für Dezimalstellen erfolgt, was bei Alpha-Feldern auch keinen Sinn machen würde.

Bis Zeile 12 folgen weitere Felddefinitionen, die keiner weiteren Erklärung benötigen.

Abschließend folgt in Zeile 13 eine neue Namensart. „K“ bedeutet hier, dass in dieser Zeile ein Schlüsselfeld für diese Datei folgt. Ich definiere hier das bereits vorhandene Feld „ADRNR“ als Schlüsselfeld. Es können natürlich weitere Felder mit der Namensart K eingegeben werden. Beachte aber, dass der entstehende Schlüssel eindeutig wird. Ich habe in der ersten Zeile, noch vor der Angabe der Namensart „R“, also eines Records oder Satzformates, die Funktion UNIQUE angegeben. Man nennt diese Position dann eine Definition auf Datei-ebene, während eine Feldangabe eine Definition auf Satz-ebene ist. Da auf Datei-ebene mit UNIQUE die Eindeutigkeit der Schlüssel definiert ist, gehören alle in dieser Dateidefinition mit K angegebenen Schlüsselfelder zu einem eindeutigen Schlüssel.
Dies ist bei einem einzigen Schlüsselfeld noch übersichtlich ☺

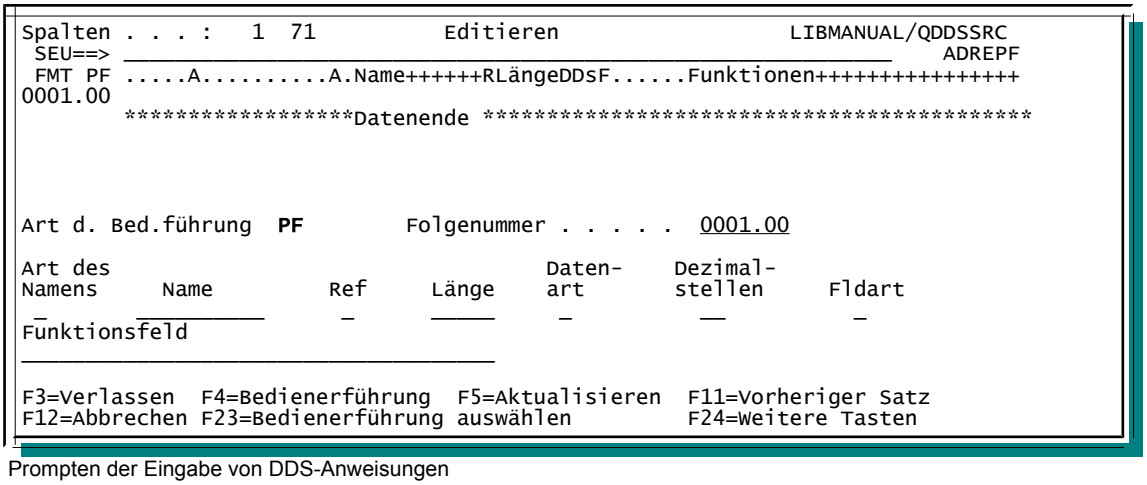
Nun weißt Du also über den spalten orientierten Aufbau der DDS-Definition Bescheid. Am besten, Du tippst das Beispiel nun ab. Achte besonders darauf, dass Du jede Angabe in die richtige Spalte eingibst.

Schwierig?

Vertippt?
…
Ich warte so lange.

Prompt gibt es Hilfe

Ok, es soll ja niemandem unnötig schwer gemacht werden. Außerdem haben wir ja eine durchdachte Benutzeroberfläche. Nehmen wir an, Du hast noch nichts eingegeben, und befindest Dich in der leeren Zeile `0001.00`. Nun drücke mal auf die Taste F4, also auf Bedienerführung.

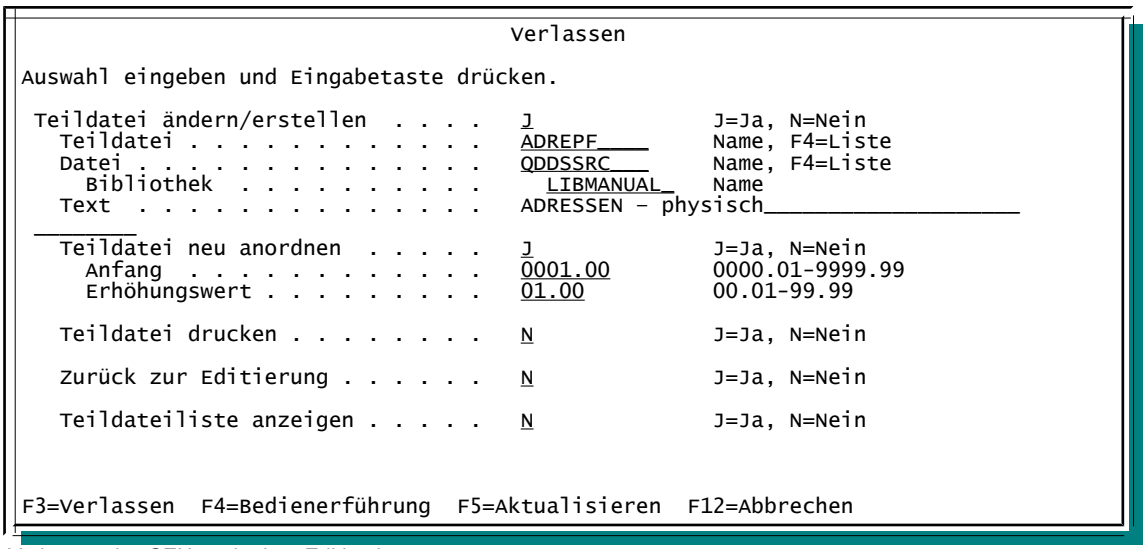


Prompten der Eingabe von DDS-Anweisungen

Im unteren Teil des Bildschirms werden die Zeilen durch eine Bedienerführung mit mehreren Feldern ersetzt. Diese Felder repräsentieren die einzelnen Spalten. Wenn Du nun also mit dem Tabulator zum Eingabefeld unter „FUNKTIONSFELD“ wechselst, dort UNIQUE eintippst und mit DF bestätigst, wird Deine Angabe an der richtigen Stelle in der Zeile platziert. Probiere dies auch mit den anderen Zeilen. Anmerkung: Wenn Du mehrere Felder ausfüllst, verlasse jedes einzelne Feld mit der Return-Taste, nicht mit der Tabulatortaste. Das System stellt somit sicher, dass Felder, die rechtsbündig gefüllt werden müssen, auch rechtsbündig eingetragen werden. Und wenn Du weiteres über die möglichen Eingaben in einem Feld wissen willst, die F1-Taste ist Dein Freund.

Das Prompting verlässt Du mit F12.

Gut, nun hast Du die Spezifikationen für die Adressentabelle korrekt abgetippt. Solltest Du falsche Spalten belegt haben oder ungültige Kombinationen eingetippt haben, wird Dir die AS/400 die fehlerhafte Zeile hinterlegt anzeigen und eine Meldung ausgeben. Wenn alles fehlerfrei ist, verlasse den SEU mit der Taste F3.



Verlassen des SEU nach einer Editiersitzung

Der Bildschirm „Verlassen“ stellt noch einige Fragen, die mit dem Abspeichern der Teildatei zu tun haben. Wichtig ist, dass das Feld „Teildatei ändern/erstellen“ auf J gesetzt ist.
Die Optionen zum neu anordnen ermöglichen Dir es, die Numerierung der Zeilen zu erneuern. Wie Du bestimmt bemerkt hast, passt SEU die Numerierung automatisch Deinen Eingaben an. Manchmal, wenn Du sehr viele große Blöcke einfügst, kann es passieren, dass Du viele Zeilen mit Angaben hinter dem Punkt hast. Das ist zwar mehr oder weniger nur ein optisches Problem, aber es kann stören. Dann benutze diese Funktion.

Sollte unten das Feld „Zurück zur Editierung“ ebenfalls auf J stehen, existieren möglicherweise noch Fehler in Deinem Quelltext, die Du bearbeiten solltest.

Ansonsten bestätige das Verlassen und Sichern der Teildatei.

Umwandeln und prüfen

Hast Du die Quelle eingegeben, machen wir uns auch direkt daran, der AS/400 den Befehl zu geben, die Spezifikationen zu prüfen und im Erfolgsfalle die Datendatei zu erzeugen.

Da Du dich wieder im Bildschirm „Mit Teildateien arbeiten (mittels PDM)“ befindest, kannst Du vor den Eintrag für ADREPF die Auswahl 14 (=Umwandeln, siehe Auswahlenzeile oben) eintragen und bestätigen.

Das System arbeitet kurz, prüft dabei alle Zeilen, die Du eingegeben hast und erstellt einen Bericht über alle Aktionen. Hast Du die Spezifikationen fehlerfrei abgetippt, dürfte im Bildschirmfuß eine Meldung wie „Datei ADREPF in Bibliothek <DeineBib> erstellt.“ erscheinen. Sollte dies nicht der Fall sein, ändere bitte die Quelldatei mit der Auswahl 2.

Für alle, die nicht nur die Oberfläche benutzen wollen, sondern auch Systembefehle kennenlernen wollen, hier ein Tipp. Die PDM-Oberfläche setzt auch nur einen Systembefehl ab, wenn Du eine Auswahl vor eine Quellenteildatei eingibst. Drücke nach der Umwandlung Deiner Datei auf die Taste F9. Das System sollte Dir nun in der Befehlzeile folgenden Befehl anzeigen:

```
CRTPF FILE(DeineBib/ADREPF) SRCFILE(DeineBib/QDDSSRC) SRCMBR(ADREPF)
```

Dieser Befehl ist zum erstellen einer physikalischen Datei da (Create physical File). Die Parameter im Einzelnen:

FILE(Bibliothek/Zieldatei)	gibt an, an welchem Ort unter welchem Namen die physikalische Datei erstellt werden soll.
SRCFILE(Bibliothek/Quelldatei)	gibt an, wo das System die Teildatei mit den Bestimmungen über das Aussehen der Datei findet.
SRCMBR(Teildateiname)	gibt den Namen der Teildatei innerhalb der Quellendatei an.

Beachte, dass die gesamte Bezeichnung Bibliothek/Datei/Teildatei in zwei Parameter aufgeteilt ist.

Somit weißt Du, wie eine physikalische Datei erstellt wird. Sollte nach F9 dieser Befehl nicht angezeigt werden, musst Du die Einstellungen für Dein PDM ändern. Dafür gibt es hier die Taste F18 (Standard-Werte ändern). Es erscheint ein Bildschirm mit vielen möglichen Einstellungen. Uns interessiert aber nur auf der zweiten Seite (blättern mit BildAb) der Wert „Befehle für Auswahl protokoll.“. Wenn Du dort J eingibst, steht die Taste F9 (Auffinden) auch für Befehle zur Verfügung, die über eine Auswahlzahl erzeugt wurden. Praktisch, gelle? ☺

Nun kannst Du Dir zur Prüfung die Ausgabe des Systems zur Umwandlung Deiner Quelldatei ansehen. Der Compiler produziert üblicherweise für jede Aktion eine Druckausgabe, so wie unter OS/400 fast jeder Job protokolliert wird. Da OS/400 alle Druckausgaben in so genannte Druckwarteschlangen (engl. Spools) schreibt, lautet der Befehl „work with spooled files“ abgekürzt:

```
wrksplf
```

Gib diesen Befehl bitte in die Befehlszeile (unten, rechts vom Symbol ==>) ein und bestätige.

Es erscheint der Bildschirm „`Mit allen SPOOL-Dateien arbeiten`“. Dieser wird hier nicht weiter beschrieben, es ist eine übliche Liste. Mit der Auswahl `5` kannst Du einen Ausdruck am Bildschirm betrachten, mit der Auswahl `4` eine Spooldatei löschen. Es wird empfohlen, nicht mehr benötigte Spools bei Gelegenheit zu entfernen, da diese mit der Zeit Festplattenplatz benötigen und die Liste lang und unübersichtlich machen. Besonders wenn viel programmiert wird, wird diese Liste sehr lang, da OS/400 sehr geschwätzig über die eigenen Aktionen werden kann. Stell Dir vor, Du probierst viel aus und wandelst häufig um. Da kommen an einem langen Abend schon mal über hundert Spools zusammen. Wie man die Angaben in diesem Protokoll liest, findest Du im Anhang B auf Seite 215.

Dateneingabe auf die einfache Art

Kommen wir nun zu einem interessanten Teil, und zu einem interessanten Programm. Wir haben jetzt eine schöne, neue Datei erstellt. Da sie jetzt recht leer ist, müssen wir sie auch mit Leben erfüllen, sprich, Daten eingeben. Später werden wir uns dafür eigene Programme schreiben, aber bis es so weit ist, helfen wir uns mit einem netten Tool von OS/400 aus, welches uns unter die Arme greift. Dieses Tool heißt `DFU` (das ist eine Abkürzung für „Data File Utility“), und zum Starten gib bitte in die Befehlszeile (die wie immer unten über den F-Tasten zu finden ist) ein:

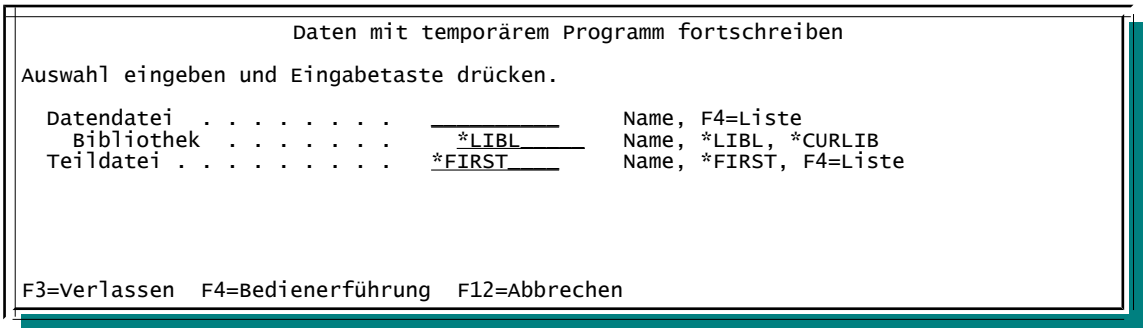
```
STRDFU
```

und bestätige. Es erscheint ein neues Menü, das die groben Vorzüge von DFU anpreist. Mit DFU kann man nicht nur auf die schnelle jede vorhandene Datendatei bearbeiten, sondern DFU kann uns auch Programme generieren, mit denen ein unbedarfter Benutzer eine Datendatei bearbeiten kann. Stelle Dir vor, Du erstellst eine Applikation mit vielen Dateien. Ein Großteil dieser Dateien wird nur gelegentlich bearbeitet, und die Logik der Bearbeitungsprogramme ist nicht aufwändig.

Da Du keine Zeit hast, für jede Datei ein Programm zu schreiben, erstellst Du Dir erst mal mit DFU die Bearbeitungsprogramme und kümmerst Dich um die komplexeren Programme. Die so erstellten DFU-Programme kann man per Menü den Benutzern zugänglich machen.
Ausserdem kannst Du mit DFU jede Dir zugängliche Datendatei bearbeiten, da DFU die Informationen über die Dateien aus diesen auslesen kann. Nutzen wir also zunächst diese Funktion, die mit der Menüoption

```
5. Daten mit temporärem Programm fortschreiben
```

gestartet wird. Nach Auswahl dieser Menüoption will DFU wissen, welche Datei bearbeitet werden soll.



DFU benötigt weitere Angaben

DFU behält sich Deine letzten Eingaben, das wirst Du merken, wenn Du es öfter startest. Beachte daher später bitte, daß unter TEILDATEI meist der gleiche Eintrag wie unter DATENDATEI steht. Wenn Du nun den Namen der Datendatei änderst, paßt der Teildateiname nicht dazu, wie DFU Dir mitteilt. Lösche in diesem Falle den Eintrag für Teildatei oder überschreibe ihn mit dem Sonderwert „*FIRST“. (Dies gilt besonders für den Account HACKER)

Bitte trage nun den Namen Deiner Datendatei ein, sowie den Bibliotheksnamen. Solltest Du den Bibliotheksnamen nicht eingeben, wird DFU sich möglicherweise beschweren, dass es die Datei nicht finden kann. Warum, dazu nachher.

DFU gibt Dir jetzt einen einfach aufgebauten Bildschirm mit allen Feldern Deiner Datendatei. Dort kannst Du nun Werte für alle Felder eintragen und abspeichern.

Eventuell mußt Du in den Funktionstasten schauen, damit Du Daten eingeben oder ändern kannst, F10 wird Dir helfen.

Gib mir Daten!

Bitte fülle nun die Felder wie folgt aus:

```
MIT DATEN IN EINER DATEI ARBEITEN                Modus  . . . :   EINGEBEN
Format . . . . :   ADRFMT                        Datei  . . . :   ADREPF

ADRNR:      1
ADRANRE:    Herr
ADRVNAME:   Klaus
ADRNNAME:   Müller
ADRGEB:     19630825
ADRSTR:     Hintertupfinger Weg 9
ADRPLZ:     12345
ADRORT:     Testort
ADRTEL:     0123/456789
ADRMAIL:    klaustest@muellerklaus.test


F3=Verlassen  F4=Bedienerführung  F5=Aktualisieren  F12=Abbrechen
```

Ein Beispieldatensatz

Bestätige Deine Eingaben mit Datenfreigabe. Die Feldinhalte werden gespeichert, der Bildschirm gelöscht, und Du kannst weitere Daten eingeben, wie es Dir beliebt. Achte darauf, wie ich das Datum 25.08.1963 eingegeben habe, es ist eine „Unsitte“ aus DV-technischen Welten, ermöglicht aber sehr einfache und schnelle Sortierroutinen.[I] Daher wurde das Feld **ADRGEB** als 10stellige Dezimalzahl definiert. Wenn wir später unsere Programme schreiben, speichern wir ein Datum in diesem Format, werden dem Benutzer aber ein „schöneres“ Format präsentieren, vielleicht sollten wir gleich ein richtiges Datenformat verwenden? ☺

Tippe also nun ein paar Datensätze nach Lust und Laune ein, lass Dir ein paar abwechslungsreiche Daten einfallen und verwende möglichst alle Felder. Da wir beim Definieren der Datei faul waren, müssen wir uns jetzt darum kümmern, dass wir richtige Werte für `ADRNR` eingeben. Doppelte Werte moniert DFU an, ansonsten zähle einfach hoch.

Wenn Du einige Sätze (10 oder mehr sollten es schon sein) eingetippt hast, lies weiter. Viel Spaß! Ich mache so lange eine Zigaretten- und Denkpause.

[I] Ja, liebe Profis... keine Sorge, diese Unsitte werden wir nicht auf Dauer übernehmen. Es folgt ein Abschnitt über die richtigen Datumsfelder, und was man damit so alles anstellen kann.

Der Pfad zu den Daten

Kommen wir zu einer sehr interessanten Sache, die uns etwas mehr Verständnis für die Organisation der Daten in Bibliotheken bringt. Wie wir gelernt haben, können wir unsere Daten in Bibliotheken ordnen. Zum Beispiel Projekt A in Bibliothek „PROJEKTA“, Testdaten in „LIBTEST“, das Betriebssystem liegt in „QSYS“ und so fort. Allerdings ist man bei seiner Ordnungswut nie auf eine einzelne Bibliothek beschränkt. Man kann sich durch geschicktes Verteilen der Daten auf mehrere Bibliotheken auch einiges an Arbeit ersparen.

Vom PC her wirst Du das Prinzip des Suchpfades kennen. Wenn Du unter DOS oder einem Eingabeaufforderungs-Fenster einen Befehl angibst, sucht der Rechner alle Verzeichnisse in diesem Pfad durch, bis er den Befehl gefunden hat, und führt ihn aus. Mit der Systemvariablen „$PATH“ kann man definieren, wo die Programme zu suchen sind.

Dieses Prinzip ist aber schon älter als der PC, und die AS/400 kann dies nicht nur mit Programmen, sondern mit allen Objekten, also auch Datendateien. Der Suchpfad wird hier „library list“ oder „Bibliotheksliste“ genannt. Diese Liste wird jedes Mal von oben nach unten durchsucht, bis das gesuchte Objekt gefunden wurde. Ausnahme: Man gibt ganz spezifisch an, aus welcher Bibliothek man ein Objekt haben will (was aber nicht immer Sinn macht).

Ein Beispiel: Du gibst einen Befehl ein (wie `strpdm`) und drückst auf DF. Das Betriebssystem sucht nun in der Bibliotheksliste nach einem Objekt mit dem Namen „STRPDM“ und dem Attribut „*CMD“. In der Bibliothek „QSYS“ wird das System fündig und führt den dort abgelegten Befehl aus.

Dies funktioniert auch mit Datendateien. Wenn Du also (in einem Programm, mit SQL oder wie auch immer) die Datei „ADREPF“ ansprichst, ohne explizit eine Bibliothek anzugeben, sucht das Betriebssystem in der Bibliotheksliste, ob in einer der dort stehenden Bibliotheken das Objekt gefunden wird. Befindet sich die Bibliothek nicht in dieser Liste, wird das Objekt auch nicht gefunden und das System meckert.

Probiere es mal aus. Es gibt den Befehl „work with objects“ (Mit Objekten arbeiten), mit dem man einige Operationen mit allen möglichen Objekten anstellen kann. Wir nutzen diesen Befehl an dieser Stelle nur, um die Funktionsweise der Bibliotheksliste zu verstehen.

Gib ein:

```
wrkobj adrepf
```

Auf dem folgenden Bild wird Dir angezeigt, ob und wo ein Objekt dieses Namens gefunden wird. Wenn Deine Bibliothek nicht in Deiner Bibliotheksliste steht, wirst Du die Meldung „(Zu dem angegebenen Namen kann kein Objekt gefunden werden.)“ sehen. Nun probiere mal den Befehl „display library list“ (Bibliotheksliste anzeigen) aus:

`dsplibl` (die Namensgebung dürfte verständlich sein?)

Du siehst eine Liste mit Bibliotheksnamen. Die Spalte „Art“ gibt an, um was für einen Art Eintrag es sich hier handelt. Einträge der Art „SYS“ sind vom System bzw. dem Administrator vorgegeben und können nicht geändert werden, Einträge der Art „USR“ sind für alle Benutzer gedacht und können getauscht oder gelöscht werden. Nun ist vermutlich Deine Bibliothek mit der Datendatei nicht enthalten. Wir müssen also die Bibliothekenliste verändern / editieren. Dies können wir mit dem Befehl „edit library list“ (Bibliotheksliste bearbeiten) erledigen:

```
edtlibl
```

Zeigt uns einen ähnlichen Bildschirm, wir sehen allerdings nur die Benutzereinträge der Bibliotheksliste. Weiterhin gibt es eine Menge an leeren Feldern, die wir mit Bibliotheksnamen füllen können. Die Spalte „Folgenummer“ sagt dem System, in welcher Reihenfolge die Bibliotheken im Benutzerteil organisiert werden sollen. Du kannst also an beliebiger Stelle auf dem Bildschirm den Namen Deiner Bibliothek eintragen. Da wir aber wollen, dass unsere Bibliothek oberste Priorität hat, sollte sie einen Eintrag mit niedriger Folgenummer haben. Am besten, Du trägst den Namen rechts vom Eintrag mit der Folgenummer „010“ ein und bestätigst den Bildschirm. Dieser erste Eintrag wird vom System leer gelassen, damit Du schnell arbeiten kannst.

Wenn Du nun den Befehl „wrkobj adrepf“ wiederholst (probiere die Taste F9), sollte Deine Datei sichtbar sein. Das ist doch einfach, nicht wahr? ☺

Probiere nun mal den Befehl

```
wrkobj qddssrc
```

Du wirst mindestens zwei Einträge sehen. Einen in Deiner Bibliothek, und einen in der Bibliothek „QGPL“. Letztere Bibliothek ist von IBM für „allgemeine Zwecke“ vorgesehen, aber bitte: lass die Finger davon und arbeite in Deiner Bibliothek. Sonst gibt es nur Streitereien mit anderen Benutzern, und vielleicht komme ich ja auf die Idee und lösche die Datei in QGPL, da sie keinen Sinn für mein System hat.

Auf jeden Fall siehst Du an diesem Beispiel, dass Du auf mindestens zwei Objekte des Namens „QDDSSRC“ zugreifen kannst, wenn Du bei einem Befehl den Bibliotheksnamen nicht angibst. Dies kann zu ungewünschten Effekten führen, andererseits aber auch gewünscht und vorteilhaft sein.

Stell Dir vor, Du arbeitest an einem großen Projekt. In der Bibliothek „LIBDAT01“ hast Du alle Datendateien mit produktivem Inhalt. Nun möchtest Du ein paar Programme testen, aber keinen Datenverlust provozieren, wenn die Programme Amok laufen. Da Du beim Programmieren sowieso kaum eine Bibliothek, sondern nur einen Dateinamen angeben kannst, hast Du es einfach. Du kopierst die gesamte Bibliothek inklusive Inhalt in eine Bibliothek namens „LIBTEST01“. Dann änderst Du Deine Bibliotheksliste so, dass „LIBTEST01“ vor „LIBDAT01“ steht, oder entfernst „LIBDAT01“ ganz. Nun rufst Dein Programm auf; schon arbeitet es mit den Testdaten. Wenn Du fertig getestet hast, löschst Du einfach die „LIBTEST01“ wieder und passt die Bibliotheksliste wieder an.
Sowas ähnliches kennt man unter Windows, wenn man ODBC verwendet. In Delphi oder Visual Basic kann man auch recht einfach in einem Projekt das Ziel einer ODBC-Datenquelle verändern. Aber geht das auch in einem laufenden Programm, oder abhängig vom Benutzer oder der Uhrzeit oder was auch immer? ;-)

Diese Vorgehensweise macht natürlich nur Sinn, wenn in beiden Bibliotheken der Datenaufbau der Dateien gleich ist, aber das ist doch verständlich, oder?

Für die normalen Entwickler und Bastler hier auf meiner Kiste dürfte die Bibliotheksliste aber kein wirkliches Problem darstellen. Jeder Benutzer hat seine eigene Bibliothek als Standardbibliothek eingetragen, und wer sich selbst eine Bibliothek anlegt, muß sich eben selbst darum kümmern. Bitte überlegt, ob es wirklich Sinn macht, mit neuen Bibliotheken zu arbeiten. Nicht immer braucht man wirklich 5 Bibliotheken, von denen 3 leer sind oder nur jeweils ein Objekt enthalten.

Der Pfad zu vielen Daten

Ein komplizierteres Beispiel. Stelle Dir ein großes Programmpaket mit folgenden Bibliotheken vor:

`LIBPGM01`	Programme
`LIBPGM02`	Testprogramme
`LIBDAT01`	Daten (produktiv)
`LIBDAT02`	Daten (zum Testen)
`LIBLNGDEU`	Deutsche Texte
`LIBLNGENG`	Englische Texte
`LIBLNGFRA`	Französische Texte

Nun arbeitet der normale Benutzer mit folgender Bibliotheksliste: `LIBPGM01, DAT01, LIBLNGDEU` (und vielleicht weitere). Durch Austausch der Bibliotheken 01/02 in der Bibliotheksliste kannst Du recht einfach eine Testumgebung bauen. Und wenn Du als Programmierer alle Texte für Bildschirme und Druckausgabe aus speziellen Dateien holst, die in `LIBLNG*` für jede Sprache entsprechend angelegt sind, kannst Du jedem Benutzer durch Hinzufügen der entsprechenden Bibliothek in die Bibliotheksliste eine andere Sprachumgebung setzen. Deiner Phantasie sind keine Grenzen gesetzt! Wie man das im Detail macht, wird zu gegebener Zeit im Beispielbereich gezeigt (bis dahin werden aber noch ein paar Seiten geschrieben und Boote den Rhein entlangfahren, sorry.)

Ist ja alles nur temporär…

Ach ja: Die Bibliotheksliste, die Du eben verändert hast, bleibt nur so lange aktiv, wie Du angemeldet bist. Dies hat den Grund darin, dass unter OS/400 jeder Job seine eigene Bibliotheksliste haben kann. Wenn Du also mit zwei Bildschirmsitzungen am System angemeldet bist, kannst Du auch unterschiedliche Bibliothekslisten haben.

Ähnlich verhält es sich mit der Bibliothek „QTEMP“, deren Namen Du eben im Benutzerteil der Bibliotheksliste bestimmt gesehen hast. Diese Bibliothek ist etwas ganz besonderes. Sie ist erstens nur temporär, und zweitens für jeden Job eigen. Solange Du an einer Bildschirmsitzung angemeldet bist, kannst Du in QTEMP Daten ablegen und ändern. Sobald Du Dich abmeldest, geht der Inhalt verloren. Bist Du gleichzeitig auf einer weiteren Bildschirmsitzung angemeldet und schaust in QTEMP nach, wirst Du kein Objekt aus der Bibliothek QTEMP Deiner ersten Sitzung sehen können! Bedenke aber, dass QTEMP für jeden Job eine eigene Bibliothek darstellt. Und wenn Du einen Job im Batch-Modus (dazu später) laufen lässt und dabei in QTEMP Daten erstellst, sind diese Daten weg, sobald der Job beendet ist.

Aber stelle Dir vor, Du hast eine größere Arbeit zu erledigen, die aus mehreren Programmen besteht. Diese Programme tauschen untereinander Daten über Datendateien aus. Diese kannst Du in QTEMP bearbeiten, wenn Du sie danach nicht mehr benötigst.

SQL mit Struktur

So, jetzt haben wir eine Datendatei mit ein paar Sätzen. Nun wollen wir mal sehen, was in dieser Datei so drin steht. Die AS/400 ist von Hause aus SQL-fähig, eigentlich arbeitet sie intern viel mit SQL. Um aber von aussen oder von der Oberfläche mit SQL zu arbeiten, muß man das SQL-Programm zusätzlich kaufen (weil auch IBM wohl Lizenzgebühren zahlen muß). Dann hat man aber ein nettes Tool zur Verfügung, um SQL-Befehle zu erstellen, auszuführen und zur Weiterverwendung abzuspeichern. Es gibt einmal das interaktive SQL-Tool, auf der anderen Seite kann man SQL-Befehle in eine Datei abspeichern und aufrufen. Weiterhin kann man SQL-Befehle in anderen Programmiersprachen wie RPG und Cobol verwenden.

Springen wir einfach mal rein und starten das interaktive SQL-Tool mit dem Befehl:

```
strsql
```

Es erscheint ein fast leerer Bildschirm, in dem Du SQL-Befehle eintippen kannst. Befehle von Deinen SQL-Sitzungen werden immer gespeichert, so dass Du beim erneuten Starten auf Deine Arbeit zurückgreifen kannst. Auf einen SQL-Kurs verzichte ich hier, dafür gibt es im Internet eine Menge interessante Literatur. Ich erläutere nur einige Besonderheiten auf der AS/400.

Zunächst eine Übersicht über die Begriffe:

AS/400-Begriff	*SQL-Standardbegriff*
Bibliothek	Collection / Database
Physikalische Datendatei (physical file)	Table
Logische Datendatei (logical file)	View / Index
Datensatz (record)	Zeile
Datenfeld	Spalte
Teildatei (Member)	(keine Entsprechung)
Logische Datendatei mit Selektion	(keine Entsprechung, teilweise Index)

Wie Du siehst, kann man mit dem DDS-Interface einiges mehr anstellen als mit den SQL-Befehlen. Natürlich kann man via SQL auch sich Views und Indexe erstellen. Allerdings kann man in einer logischen Datendatei gleichzeitig bestimmte Felder auswählen, bestimmte Sätze auswählen und die Daten sortieren.

Um im interaktiven SQL nun auf eine Tabelle (resp. Datendatei) zugreifen zu können, muss man entweder die benötigte Bibliothek in der Bibliotheksliste stehen haben (siehe oben), oder beim Zugriff die entsprechende Bibliothek (Collection) angeben. Auf einen SQL-Kurs verzichte ich an dieser Stelle und zeige nur, wie man SQL-Befehle eingeben kann.

Probiere einfach mal aus:

```
Select * from adrepf
```

Auf einem neuen Bildschirm „`Daten anzeigen`“ werden die Daten der Datei angezeigt. Du kannst mit BildAuf und BildAb blättern. Die Daten werden zeilenweise angezeigt; da die Felder eines Datensatzes nicht in einer Zeile angezeigt werden können, fehlen die meisten Angaben. Schau mal auf die Funktionstasten, wie man seitwärts die Daten verschieben kann ;-) Die beiden Eingabefelder sind recht selbsterklärend, besonders wenn man die Hilfetaste verwendet (Du merkst, ich nötige Dich zum selbst ausprobieren, einge gute Lernmethode. Kaputt machen kannst Du ausser Deinen eigenen Daten nicht viel.)

Verlasse den Bildschirm, und Du landest wieder im SQL-Eingabebild.

Wie Du bemerkt hast, wurden die Daten aus ADREPF in der gleichen Reihenfolge angezeigt, wie Du sie mit dem DFU eingegeben hast. Grund hierfür ist logischerweise die nicht vorhandene Sortierung in der physikalischen Datei.

Nun weißt Du (in groben Zügen), wie Du mit SQL Daten anzeigen und notfalls auch manipulieren kannst. Bei Gelegenheit werde ich weitere SQL-Befehle erläutern. Wer etwas mehr mit SQL experimentieren will, kann gerne die inzwischen bekannte Taste F4 ausprobieren. Ich lasse Dich nun etwas experimentieren.

Kommen wir nun wieder zurück zu den Dateien.

Ist doch logisch, oder?

Eben im SQL-Fenster hast Du gesehen, dass die Datensätze in der gleichen Reihenfolge angezeigt wurden, wie Du sie eingegeben hast. Dies ist natürlich nicht immer sinnvoll. Zum Einen möchte ein Benutzer auf einem Auswahlbildschirm gerne, dass die Daten in einer bestimmten Reihenfolge (z.B. Nachname) sortiert angezeigt werden, andererseits wird die Suche nach Daten in einem unsortierten Haufen Datensätze sehr uneffektiv.

Du könntest zum Beispiel die Daten mit SQL sortieren, z.B.:

```
Select * from adrepf order by adrename
```

Somit zeigt Dir SQL die Daten sortiert nach Nachname an. Wenn Du unten den Bildschirm beobachtest, nachdem Du den Befehl startest, bemerkst Du vielleicht kurz die Anzeige, dass ein so genannter „`Zugriffspfad`“ aufgebaut wird. Hiermit vermeldet die AS/400, dass ein Benutzer oder ein Programm Daten sortiert haben möchte, für die es kein bereits definierte Sortierung gibt. Das Betriebssystem erstellt nun für das Programm automatisch eine Sortierliste. Dies ist zwar ganz angenehm, benötigt aber zusätzlichen Speicher und Rechenkapazität. Da dieser temporäre Zugriffspfad für jedes Programm, bzw. jeden Benutzer individuell erstellt wird, ist dies eine unnötige Verschwendung von Ressourcen, besonders wenn dies 100 Benutzer gleichzeitig mit großen Dateien machen. So kriegt man auch den dicksten Hobel klein.

Also sollte sich der Programmierer einer Datenbankanwendung (also Du!) darum kümmern, dass erstens die benötigten Zugriffspfade fest definiert werden, und zweitens Programme nicht unnötig mit einer falschen Sortieranfrage auf eine anders sortierte Datei zugreifen.

Fest definierte Zugriffspfade haben mehrere Vorteile: Das Betriebssystem kann sich automatisch darum kümmern, dass der Zugriffspfad ständig aktuell ist. Weiterhin können sich alle Programme und Benutzer im System einen einzigen Zugriffspfad teilen. Das spart Platz und eine Menge Zeit, da bei geschickter Auswahl von Zugriffspfad und entsprechender Programmierung eine Suche innerhalb vieler Gigabytes an Daten in Millisekunden erledigt ist.

Nun habe ich so oft das Wort „Zugriffspfad“ auf einer Seite verwendet, dass Du mich wohl schon erschlagen willst. Sorry, dies ist die Terminologie von IBM. Verwenden wir an dieser Stelle den Begriff „logische Datei“. Dies ist für SQL-Kundige am ehesten mit „Index“ zu übersetzen. Eine logische Datei also ist das Zaubermittel für unsere Sortierprobleme. Sie basiert auf einer physikalischen Datei und enthält nur Daten, wie das System gewünschte Daten in der physikalischen Datei finden kann. Weiterhin kann eine logische Datei die Sicht auf eine physikalische Datei anders darzustellen, wie sie wirklich ist, auch mehrere Dateien verbinden (join) etc…

Fangen wir ganz vorne an.
Wir möchten später ein Programm schreiben, das (unter anderem) es dem Benutzer ermöglichen soll, eine Adresse nach dem Nachnamen zu suchen und anzeigen zu lassen.

Daher benötigen wir zunächst einmal eine logische Datei, die die Daten nach dem Nachnamen sortiert. Bitte stelle sicher, dass Du dich wieder im Bild „`Mit Teildateien arbeiten (mittels PDM)`“ befindest, und die Teildateien für Datenquellen, in unserem Falle QDDSSRC angezeigt bekommst.

Nun erstellen wir uns eine neue Teildatei, die die Bestimmungen für die logische Datei enthält. Aufgrund des ähnlichen Aufbaus und unserer Faulheit bietet es sich an, einfach eine Kopie der Quelle für „ADREPF“ zu erstellen und zu bearbeiten. Bitte trage in das Auswahlfeld vor der Teildatei „ADREPF“ die Zahl 3 ein und drücke DF. Im neuen Bild „Teildateien kopieren“ musst Du nun angeben, wo die Kopie erstellt werden soll. Lasse die Angaben für Zieldatei und Zielbibliothek gleich, und gib in der Bildschirmmitte als neuen Namen den Wert „ADREL01“ ein.

Anhand der Namensgebung erkennt man sofort, um welche Art von Datei es sich handelt, es hat sich eingebürgert, physikalische Dateien auf „PF“ enden zu lassen, logische Dateien auf „Lxx“. Das xx steht für eine Numerierung, da man zu einer physikalischen Datei viele logische Dateien mit unterschiedlichen Sortierungen erstellen kann. Natürlich kannst Du eigene Namen verwenden, wenn Du ein eigenes System für Ordnung hast.

Was fällt noch auf? Die Kopie enthält noch die gleiche Dateiart und den gleichen Text wie das Original. Das ist natürlich nicht erwünscht, eine logische Datei muss die Art „LF“ erhalten. Der Text sollte der Funktion entsprechen. Du kannst in der Liste der Teildateien diese Felder einfach überschreiben. Bitte überschreibe nun für die Teildatei „ADREL01“ die Art mit „LF“, und den Text mit „ADRESSEN – logisch (NAME)“. Nun die Eingaben bestätigen, schon ist der erste Schritt getan.

Jetzt fehlt aber das wichtigste: Der Inhalt der logischen Datei passt natürlich noch nicht unseren Anforderungen. Also mit Auswahl 2 die Datei editieren! Die nötigen Schritte beschreibe ich anhand der Ausgangsdatei, wie oben beschrieben.

Zunächst solltest Du die Zeile 0001 ändern. Das Funktionswort UNIQUE ist bei unserer logischen Datei nicht sinnvoll, da wir als Sortierwert nur den Nachnamen haben werden, und jeder Herr Müller oder jede Frau Mayer dürfte hier Schwierigkeiten bekommen ;-) Weiterhin ist die physikalische Datei bereits als Eindeutig definiert, und mehrere UNIQUE-Regeln, die auf die gleiche physische Datei deuten, sind nicht immer wirklich sinnvoll.

Wichtig ist bei jeder logischen Datei, dass zu jeder Angabe über ein Satzformat angegeben wird, auf welche physikalische Datei sich das Format bezieht (der Name der logischen Datei kann komplett von der physikalischen Datei abweichen, sie kann sogar in einer anderen Bibliothek liegen). Die Funktion hierfür lautet PFILE(…) mit der Angabe der physikalischen Datei. Bitte lösche zunächst Zeile 0001, indem Du ein D über die Folgenummer schreibst und den Befehl bestätigst.

Nun fügst Du der Zeile 0002 am Zeilenende die Angabe

```
PFILE(ADREPF)
```

hinzu. Ich weise noch mal darauf hin, dass diese Funktion natürlich in den Funktionsbereich der Spalte eingetragen wird. Also entweder mit dem Funktionsnamen in der Spalte anfangen, wo darüber UNIQUE stand, oder in der Zeile die Prompt-Taste (wie immer F4) verwenden. Habe ich schon erwähnt, dass man den Prompt-Modus mit F12 beendet und wieder auf die normale Zeilensicht umschaltet?

Den Namen des Satzformates belassen wir auf „ADRFMT“. Der Grund: so kann man einfach in einem Programm die verwendete logische Datei wechseln, ohne das Programm ändern zu müssen (den Zugrifsschlüssel natürlich schon). Genauer siehst Du das nachher im Abschnitt über Details zu den logischen Dateien.

Wir löschen alle vorhandenen Feldangaben. Wenn Du in einer logischen Datei alle Felder der physikalischen Datei übernehmen willst, benötigst Du keine Feldangaben. Aber wenn die logische Datei nur einen Teil der Felder enthalten soll (oder zusätzliche Felder, oder eine Kombination aus vorhandenen Feldern), dann mußt Du alle gewünschten Felder eintragen. Das Schlüsselfeld (letzte Zeile, Namensart „A“) muss geändert werden. Bitte überschreibe dort „ADRNR“ mit dem Namen für das Nachnamefeld „ADRNNAME“.
Die Beschreibung für Deine logische Datei sollte nun so aussehen:

```
Spalten . . . :   1  71          Editieren                    LIBMANUAL/QDDSSRC
 SEU==> ____________________________________________________________   ADREL01
 FMT PF .....A..........A.Name++++++RLängeDDsF......Funktionen++++++++++++++++
0002.00      A          R ADRFMT                    PFILE(ADREPF)
0003.00      A          K ADRNNAME
       ******************Datenende ***************************************


F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Die Beschreibung unserer logischen Datei in DDS

Wenn Du alle Änderungen vorgenommen hast, kannst Du den SEU mit F3 verlassen und die Umwandlung mit der Auswahl 14 vor dem Teildateinamen starten. Wenn Du keine Fehler in der Quelle hast, sollte die logische Datei problemlos erstellt werden. Bei einem Fehler bleibt die Auswahlnummer vor dem Dateinamen stehen. Probiere nun im SQL-Dienstprogramm den Befehl:

```
select * from adrel01
```

Entgegen Deiner ersten Erwartung sind die Daten *nicht* nach dem Nachnamen sortiert. Warum? Der Zugriff per SQL erfolgt bei diesem Befehl ohne Sortierangabe, das Betriebssystem liefert Dir also die Daten in physikalischer Reihenfolge, es gibt sich also stur. Du musst schon genau sagen, dass Du die Daten sortiert haben willst. Nimm also wieder den bekannten Befehl:

```
select * from adrel01 order by adrnname
```

Schon sind die Daten sortiert auf dem Bildschirm. Vorteilhaft ist dies besonders, wenn mehrere Schlüsselfelder zur Suche oder Sortierung verwendet werden. Angenommen, Du hast Deine Adressdatei mit einer logischen Datei mit folgenden Schlüsselfeldern ausgestattet:

```
Name, Vorname, PLZ, Ort
```

Nun wirst Du in Deinen Programmen nicht immer nur nach allen vier Feldern suchen, sondern z.B. nur nach dem Namen. Es reicht also die Angabe dieses einen Feldes, so dass das System Dir die Daten in dieser Reihenfolge liefert. Wir werden später noch ein schönes Kapitel zum Thema suchen und Finden haben, denn was hier unter RPG gilt, gilt für SQL eben so.

Wie Du vielleicht bemerkt hast, hat das System keine Meldung über den Aufbau eines Zugriffspfades ausgegeben, da dieser bereits existiert. Wie angesprochen bringt dies beim gleichzeitigen Zugriff durch mehrere Benutzer einen großen Vorteil.
Fangen wir nun mal ganz vorsichtig mit dem Programmieren in RPG an, und tasten uns weiter…

It's a role playing game

Programmieren – einmal anders

Die Sprache RPG hat ihre Ursprünge bereits in den 60ern, als Methode, um Buchungsvorgänge der damals modernen IBM Buchungsmaschinen zu beschreiben, bzw. um die Bearbeitungsweise der Buchungsmaschine zu steuern. Ausgeschrieben heißt RPG einfach „Report Program Generator“, also Berichts-Generierer. Ursprünglich hat man auch meistens damit einen Bericht, also eine Liste mit Daten erstellt und ausgedruckt. RPG kann heute selbstverständlich viel mehr als nur das. Es ist eine ausgewachsene Programmiersprache der vierten Generation, die im Laufe der Zeit sehr viele Erweiterungen erfahren hat, es gibt RPG, RPG II, RPG III, RPG IV und die Befehle für das ILE-Prozessmodell, das nicht nur die Sprache, sondern das gesamte Systemkonzept berührt. Lass Dich also nicht von einem altbackenen Lehrer beirren, der meint, RPG sei etwas aus „Vorkriegszeiten“. Gerade DB-Anwendungen kann man hiermit eleganter und flotter als mit vielen anderen Sprachen und als auf vielen anderen Systemen erstellen. (Es gibt sogar VisualRPG für den PC)

RPG ist natürlich eine Sprache, die man ausserhalb der AS/400-Welt kaum noch kennt, daher wird RPG auch als „Role Playing Game“, also Abenteuerspiel mit einem Rollencharakter ausgeschrieben ;-) Sie hat eben etwas eigenes.

RPG-Quelldateien werden genauso wie DDS-Dateien mit dem SEU-Programm eingegeben. RPG arbeitet ebenso Spaltenorientiert wie DDS-Dateien. Ausnahme: Seit Betriebssystemversion V5 gibt es eine RPG-Erweiterung namens /FreeFormat. Hier kann man die Befehle in freiem Format eingeben und muss sich nicht mehr um die Orientierung an festen Spalten festhalten. Leider habe ich noch kein V5 auf meinem System, so dass ich diese Erweiterungen vorerst ausklammern muß. Aber die Spaltenorientierung hat auch Vorteile, man kann damit trotzdem (oder deshalb) übersichtliche Programme schreiben. Um Benutzer mit einer älteren AS/400 unter V3R2M0 nicht auszuschliessen, achte ich darauf, dass die Programmbeispiele auch dort laufen. Wo man ein Problem eleganter mit RPG-Befehlen lösen kann, die erst auf RISC-Modellen verfügbar sind, werde ich das entsprechend vermerken.

Zum Thema Programmier-Richtlinien komme ich in einem späteren Abschnitt, wenn die Übungsprogramme länger werden und in einem großen Projekt enden. Eine Grundregel soll aber bereits erwähnt werden: Es gibt nichts schlimmeres, als an einem Programm eines anderen Programmierers zu arbeiten und rätseln zu müssen, was überhaupt passiert. Jeder Programmierer sollte auch in der höchsten Terminnot etwas Zeit haben, sein Programm zu **dokumentieren!**

Nicht jede einzelne Zeile benötigt eine Erklärung, die Funktion eines einzelnen Befehls wird jedem guten Programmierer bald bekannt sein. Aber warum eine Routine gerade *diese* Datei in *dieser* Reihenfolge bearbeitet, und eine Gruppe von *diesen* Feldern *hierhin* verschiebt, dies sollte einen Kommentar wert sein. Auch Schleifen sind ein Thema für sich. Ich habe schon Programme gehabt, in denen ohne Dokumentation zwanzigmal (in Zahlen: 20!) der Befehl `ENDIF` untereinander stand. Es kostete mich Stunden, den Effekt und Sinn der Routinen zu durchblicken.

Echte Puristen von RPG2 werden wahrscheinlich auf mich böse sein, aber ich werde viele Beispiele in RPG4 starten, um den Einstieg von Kennern anderer Sprachen zu vereinfachen. Daher werden einige Prinzipien und Besonderheiten von RPG erst einmal ignoriert.

First Step – Voraussetzungen

Als erstes Programm werden wir ein so genanntes Batch-Programm schreiben. Es gibt zwei grobe Gruppen von Programmen: *Batch* und *Interaktiv*. Ein Batch-Programm wird gestartet, erledigt seine Aufgabe und beendet sich dann. Es bearbeitet Daten, und produziert eventuell Druckausgaben. Es kommuniziert aber nicht mit dem Anwender. Dies ist den interaktiven Programmen vorbehalten. Batch-Programme werden üblicherweise in einem eigenen Speicherbereich ausgeführt, um höchste Leistung zu erreichen. Ein klassisches Beispiel für ein Batch-Programm ist das Erstellen aller Rechnungen der Auslieferungen in einem Handelsbetrieb, die sich am Tag angesammelt haben. Einmal in der Nacht wird das Rechnungsprogramm gestartet, es verbucht alle möglichen Daten und erzeugt die Druckausgabe der Rechnungen.

Interaktive Programme werden für Dateneingaben verwendet, oder um Daten zu selektieren, auszugeben und zu bearbeiten. Du hast in diesem Manual bereits mit dem DFU-Programm Daten in Deine Adressdatei eingegeben. Dies war ein sehr interaktives Programm.

Zur Verwirrung: Man kann natürlich auch ein Batch-Program in der normalen Umgebung für interaktive Programme ablaufen lassen. Allerdings hat dies meist eine stärkere Belastung des Systems zur Folge, und kommunizieren kann der Anwender auch nicht mit diesem Programm. Und ein interaktives Programm kann man auch im Batch-Modus starten, nur wie soll der Anwender mit dem Programm arbeiten?

Aber immer der Reihe nach…

Batch-Programme sind einfacher zu erstellen, da man sich nicht um die Kommunikation mit dem Anwender herumschlagen muss. Ein komplexes interaktives Programm mit vielen Bildschirmen und Ein- und Ausgabemöglichkeiten ist ein nicht ganz einfaches Thema.

Daher ist unsere erste Übung ein Batch-Programm, welches unsere Adressdatei bearbeitet. Aus diesem Grunde solltest Du einige Sätze in Deiner Adressdatei haben, mindestens 10 wären ganz angenehm. Bitte tippe unterschiedliche Daten ein, nicht immer die gleiche (imaginäre?) Adresse.

What to do?

Unser Programm soll später alle Sätze in unserer Adressdatei einlesen. Damit wir ein sinnvolles Beispiel für ein Batch-Programm erhalten, benötigen wir einige Voraussetzungen in unserer Adress-Datei:

a) Zunächst solltest Du mit DFU sicherstellen, dass nicht alle Felder „ADRORT“ in der Datenbank einen Inhalt haben. Starte also bitte DFU und lösche in einigen Datensätzen den Ort. Wie Du mit DFU Datensätze bearbeiten kannst, dürfte nach kurzem Studium der Funktionstasten und Blättertasten herausgefunden sein, ich bin gerade faul und appelliere an Deinen Wissensdurst :)

b) Wir benötigen ein weiteres Feld in der Datei. Fügen wir ein Feld „ADRSTAT“ hinzu, welches ein Kennzeichen enthält, ob die Adresse gültig ist oder nicht. Dieses Kennzeichen wird von unserem Batch-Programm erzeugt.

Schritt a) solltest Du ohne Hilfe hinbekommen. Studiere das DFU-Programm. Solange wir keine interaktiven Programme zur Pflege unserer Daten schreiben können, brauchen wir es ganz dringend ☺

Schritt b) ist nicht wirklich schwer. Starte PDM zur Bearbeitung unser Quellen für Datendateien. Wechsele mit Auswahl `12` in die Quellendatei `QDDSSRC`. Nun bearbeite die Quelle für die physikalische Datei `ADREPF` mit Auswahl `2`. Nach der Zeile mit der Definition für das Feld `ADRMAIL` muss eine weitere Zeile hinzugefügt werden. Setze also den Cursor auf die Zeilennummer `0012.00`, tippe den Buchstaben `I` ein und bestätige mit `DF`. Füge die folgende Zeile hinzu und achte auf die Spalten:

```
FMT PF .....A.........A.Name++++++RLängeDDsF.....Funktionen+++++++++++++++++
       A    A           ADRSTAT        1A        TEXT(´Status´)
```

Selbstverständlich sollte das erste `A` unter den anderen Kennzeichnungen für die Zeilenart stehen, ebenso wie der Name des Felds "`ADRSTAT`" unter die bereits eingegebenen Felder gehört. Die Längen- und Typangabe für „ein Zeichen alpha" hat ebenso ihren festen Platz wie das Funktionswort mit Parameter. Solltest Du Dich in den Spalten vertan haben, wird Dir die Zeile hinterlegt markiert.

Verlasse SEU mit `F3` und bestätige die Angaben.

Nun müssen wir die physikalische Datei entsprechend der Quelldatei ändern. Warum *ändern*? Ganz einfach. Wenn Du nun im PDM vor dem Eintrag der Quelldatei „`ADREPF`" die Auswahl `14` eingibst, will das System eine neue Datei anhand der Beschreibung erstellen. Dies wäre recht fatal, da dann unsere eingegebenen Daten verloren gehen. Also müssen wir dem System mitteilen, dass die Datei nicht *neu* erzeugt, sondern die vorhandene Datei unter Beibehaltung aller Datensätze *geändert* werden soll.

Den Befehl `CRTPF` hast Du bestimmt schon einmal gesehen, spätestens vorhin, als wir unsere Adressdatei angelegt haben. Um nun eine vorhandene physikalische Datei zu ändern, gibt es den Befehl (tusch)

`CHGPF`

Da kommt man fast von selbst drauf… Die Parameter sind sehr ähnlich, allerdings wird die Datei anhand der Definitionen nicht neu erzeugt, sondern eine vorhandene Datei verändert. Gib bitte ein:

`CHGPF FILE(`*`DeineLib`*`/ADREPF) SRCFILE(`*`DeineLib`*`/QDDSSRC) SRCMBR(ADREPF)`

(Beachte den richtigen Bibliotheksnamen).

Nach einigen Sekunden Arbeit meldet das System, dass die Datei geändert wurde. Wenn nicht, kannst Du im Protokollausdruck nachsehen, wo Dein Fehler liegt.

Nun wollen wir sehen, was passiert ist. Starte SQL und schau in Deiner Datei nach:

```
select * from adrepf
```

Blättere mit der Taste F20 bis zum letzten Feld nach rechts. Dort sollte nun das Feld „ADRSTAT“ angezeigt werden, und es ist für jeden Datensatz leer. Wie auch nicht anders zu erwarten war. Wie sieht es nun mit unserer logischen Datei ADREL01 aus? Schau einfach nach. Du wirst feststellen, dass das Feld ADRSTAT hier fehlt. Dies liegt an der Entkoppelung der logischen Datei von der physikalischen Datei. Damit nun die logische Datei wieder zur physikalischen Datei paßt, erstelle sie sicherheitshalber einfach neu. Hier kannst Du die Auswahl 14 ruhig verwenden, da beim Löschen der logischen Datei keine Daten gelöscht werden.

Die logische Datei kannst Du mit Auswahl 14 neu erstellen und somit alle Felder in diese übernehmen. Da eine logische Datei nicht wirklich Daten enthält, kannst Du sie beliebig oft neu erstellen, das System kümmert sich dann wie immer darum, dass der Zugriffspfad stimmt.

Nun haben wir also unsere Grundlage für das Programm, und wollen direkt loslegen. Also Anschnallen, die Finger lockern und weiter geht’s!

Ein Stück Theorie voraus

Es folgt ein kleines Kapitel theoretischer Natur über RPG und dem Programmieren allgemein. Du kannst es überspringen, wenn Du direkt los legen willst, aber für das genaue Verständnis einiger Details ist die Lektüre doch empfohlen.

RPG ist eine alte Sprache, das habe ich schon mal erwähnt. RPG ist mit der Zeit und mit den Anforderungen der Programmierer gewachsen. Früher war RPG gar keine prozedurale Sprache, es gab keine Steuerungsmöglichkeiten über den Ablauf, das Programm fing oben an und endete am Schluss. Inzwischen ist RPG eine vollprozedurale Sprache geworden, mit vielfältigen Steuerungsmöglichkeiten, der Schwerpunkt liegt immer noch auf der Datenverarbeitung. Man kann mit RPG und der Sprachumgebung ILE grosse, Modulare Projekte abwickeln, mit RPG auch CGI-Module für den Webserver schreiben und RPG-Module mit anderen Sprachen wie Cobol, C oder Java verknoten. Wir wollen es aber hier nicht gleich übertreiben sondern konzentrieren uns zunächst auf RPG IV als klassiche Sprache für die Entwicklung Textorientierter AS/400-Anwendungen. Diese sind performanter als jede Java-Entwicklung mit HTML-Oberfläche.

Daher übergehen wir – vielleicht zum Missfallen einiger RPG-Hasen – gleich den RPG-Zyklus aus RPG und RPG II, damit der Lernende nicht zu sehr mit diesen wirklich alten Methoden „versaut“ wird ;-)

Variablen

In der Regel ist ein Programm ohne Variablen sehr langweilig. Erst eine Variable gibt jedem Programm die Möglichkeit, flexibel zu sein. Stell Dir vor, Du schreibst ein Rechnungsprogramm ohne Variablen. Wie willst Du denn das Produkt aus Anzahl und Einzelpreis berechnen, wenn Du nicht für jede Position das Programm ändern und neu umwandeln willst? Eine Variable ist eine Speicherstelle im Computer, deren Inhalt je nach Definition eine bestimmte Form hat und jederzeit geändert werden kann (Ausnahme: Konstanten). Die Variable an sich ist ein sehr umfassender Begriff, und muss nicht unbedingt immer nur in Programmen gültigkeit haben. Daher spricht man auch von Programmvariablen oder Systemvariablen und so weiter.

In RPG wird eine Variable als *Feld* bezeichnet. Da RPG eine Sprache der kaufmännischen Datenverarbeitung ist, aus denen die Daten fast immer aus Datenbanken kommen, ist diese Bezeichnung einleuchtend. In RPG muss ein Feld stets definiert sein mit Herkunft, Typ und Grösse. Die künstlerische Freiheit von Sprachen vie PHP, in der man einem Variablennamen den Wert '5' zuweist und sie somit als Zeichenvariable definiert, und später damit Berechnungen anstellt, gibt es in RPG nicht. Man wird zur übersichtlichen Arbeit gezwungen.

Daten im Allgemeinen

In der kaufmännischen EDV arbeitet man meistens mit Daten auf Magnetplatten oder Bändern. Meist behandelt man hier eine grosse Menge Daten, und die Logiken sind relativ einfach, im Vergleich zur technischen Informatik zum Beispiel. Hier sind die Datenmengen meist geringer, der logische Aufwand zur Bearbeitung aber viel höher.

Innerhalb der Daten unterscheidet man zwischen Daten, die während der Bearbeitung anfallen und nicht langfristig gespeichert werden müssen. Man nennt diese Daten *Transaktionsdaten* oder *Bewegungsdaten*. Das bedeutet nicht, dass eine Transaktionsdatei nach der Bearbeitung nicht mehr benötigt wird, so hat man früher gedacht, als es hauptsächlich Batchverarbeitung gab. Hier ist eher das kurzfristige Rechenfeld für Zwischenwerte gemeint.

Daten, die langfristig vorgehalten werden, aber meist nicht ständig in Bewegung sind oder Veränderungen unterliegen, nennt mann *Stammdaten*. Diese Daten enthalten zum Beispiel Kundeninformationen, Artikelinformationen etc.

All diese Daten werden in einer gewissen Hierarchie sortiert und untergebracht. Abhängig vom System gibt es hier vielleicht leichte Unterschiede in der Benennung, die Hierarchie-Ebenen sind aber gleich. Zusammengehörende Daten, zum Beispiel über alle Kunden, werden in einer *Datei* (oder Tabelle) gespeichert. In dieser Datei sind viele gleichartige Informationen enthalten, die *Sätze* genannt werden. Für jeden Artikel zum Beispiel speichert man einen Satz mit allen relevanten Informationen. Jeder Satz enthält Detailinformationen zu einem einzelnen Artikel, wie Name, Preis etc. Diese einzelnen Elemente an Information nennt man *Feld*. Und so schliesst sich der Kreis, und Du weisst, warum man in RPG eine Variable *Feld* nennt.

In der Regel ist der Aufbau jedes Satzes innerhalb einer Datei gleich; jeder Satz enthält die selben Feldarten, Feldtypen und meist auch in der gleichen Reihenfolge. Es gibt auch Möglichkeiten, unterschiedliche Satzarten in Dateien zu verwenden, aber das lassen wir mal schön bleiben, man kann auch moderner arbeiten.

OS/400 kennt zwar Felder mit variabler Länge (z.B. VARCHAR), meist (in 99% der Fälle) verwendet man aber nur Felder fester Länge. Ist ein Feld `NAME` zum Beispiel mit 30 Stellen definiert, der Inhalt aber nur 17 Stellen lang, werden die restlichen Stellen mit Leerzeichen aufgefüllt. Ebenso werden numerische Felder gespeichert. Ein „normales“ numerisches Feld (Beispiel: 10 Stellen, davon 2 Stellen Nachkomma) wird zehnstellig gespeichert. Enthält das Feld den Wert 1234,56 so wird es als 0000123456 gespeichert. Da Dezimalkomma oder Tausender-Trennzeichen nicht gespeichert werden, obliegt es allein der Definition, dass nach zwischen Stelle 8 und 9 ein Dezimalkomma existiert. Formatierungen auf Bildschirm oder Drucker sind Sache des Programmierers oder des Systems. (Nebenbei gibt es auch ein Zahlenformat, das als „gepackt“ bezeichnet wird. Hier werden die Zahlen hexadezimal gespeichert, so dass immer zwei Ziffern ein Byte belegen. Spart Platz und ist für das System meist sogar noch schneller zu bearbeiten. Das lohnt aber nicht bei einem dreistelligen Zahlenfeld :)

Daten im Besonderen

Neben den Feldern kennt RPG noch andere Variablen. Eine ganz besondere ist hierbei die *Bezugszahl*. Diese ist eine einstellige logische Variable mit dem Inhalt *wahr* oder *falsch*, oder auch *1* oder *0*. Das wirklich besondere an den Bezugszahlen ist der Name: Bezugszahlen haben fast immer numerische Namen von 01 bis 99. Dies rührt aus einer Besonderheit von RPG her, die Programme spaltenorientiert (früher auf Lochkarten) im Editor einzutippen. Bei vielen Befehlen gibt es spezielle zweistellige Spalten, um dort die Nummer einer Bezugszahl einzugeben. Je nach Befehl wird unter besonderen Bedingungen der Inhalt der Bezugszahl auf den entsprechenden Wert gesetzt. Übrigens nennt man Bezugszahlen auch *Anzeiger* oder *Indikator*, um die Begriffsverwirrung zu vervollständigen.
Die Nutzung dieser Bezugszahlen war in RPG, RPG II und RPG III eine Pflicht, um das Programm abhängig von diversen Einflüssen wie Dateioperationen zu steuern. Da einige Programmierer Meister in der Verwendung dieser Bezugszahlen sind, ist das Ergebnis ein sehr unlesbares Programm. Im Extremfall kann man jeden Befehl vom Status von drei (!) Bezugszahlen abhängig machen. Wer solche Programme noch versteht, ist ein wahrer Freak :-)

Zu Zeiten von RPG IV sollte man nach Möglichkeit auf die Bezugszahlen verzichten, was auch zu 99% möglich ist. Gelegentlich kann man sie aber zur einfacheren Programmierung verwenden, so lange man es damit nicht übertreibt.

RPG im Überblick

Ein RPG-Programm besteht aus unterschiedlichen Typen von Zeilen, auch Anweisungen genannt. Jeder Zeilentyp hat eine besondere Bedeutung und Eingabemaske (siehe: Spaltenorientierung). Die Reihenfolge dieser Zeilentypen ist nicht beliebig, sondern festgelegt. Ursache ist auch hier die Lochkarten-Programmierung in den 60ern und 70ern des vorigen Jahrtausends. Da soll einer mal sagen, die IT ist keine langlebige Branche. Aber keine Sorge, es ist einfacher, als es sich anhört. Manchmal nennt man diese Anweisungen auch Spezifikationen.

Beispielsweise gibt es **F**-Anweisungen, die dem Programm mitteilen, mit welchen Dateien (engl. *F*ile) gearbeitet werden. Gemäss dem E-V-A-Prinzip (Eingabe, Verarbeitung, Ausgabe) stehen diese Zeilen am Programmanfang.
Die häufigsten Zeilen in einem Programm sind üblicherweise die **C**-Anweisungen (engl. *C*alculations). Diese enthalten alle Rechenbestimmungen und Verarbeitungsanweisungen. Man könnte sich **C** auch mit *C*ommand merken :-)

Der Typ einer Anweisung (also der Buchstabe **F**, **C**, ...) steht immer in der Spalte 6 einer Programmzeile. Verwirrend ist am Anfang, dass der SEU, den Du startest, die Zeile erst ab dieser Spalte 6 anzeigt (siehe oberste Zeile im SEU). Somit ist diese 6. Spalte hinter der Zeilennummer und einem trennenden Leerzeichen zu finden. Formatierungshilfen in Zeile 3 im SEU helfen Dir aber weiter. Die Spalten 1 bis 5 in einem RPG-Programm werden nicht mehr verwendet, also werden wir damit auch nicht im SEU belastet[I].

Ach ja: Man muss nicht alle existierenden Anweisungsarten in einem Programm verwenden, man nutzt natürlich auch nur die benötigten. Meist sind das mindestens **F** und **C**. Ein Programm ohne *C*-Zeilen ist meist recht langweilig.

Alles in Reih' und Glied!

Jede Zeilenart in RPG IV hat eine bestimmte Formatierung, so dass jedes Element der Zeile an einer bestimmten Spalte stehen muss[II], was am Anfang gewöhnungsbedürftig ist. Der Compiler interpretiert jedes Zeichen abhängig von seiner Position. Dies ist Dir ja von DDS-Quelldateien bekannt, so dass ich jetzt nicht weiter auf diese Besonderheit eingehe, Formatierungshilfen in jedem Beispiel sind ja vorhanden. Besonders am Anfang Deiner Arbeit mit RPG solltest Du Dich nicht von den vielfältigen Möglichkeiten der einzelnen Zeilentypen irritieren lassen. Bei einigen Zeilentypen wird ein Befehl abhängig vom Inhalt auch über mehrere Zeilen eingegeben. Hab keine Angst, die Macht (also dieses Manual) ist mit Dir. Meist verwendet man solche Dinge auch nur bei ganz speziellen Aufgaben.

Weiterhin wirst Du feststellen, dass ein Eintragsfeld (nicht zu verwechseln mit einem Datenfeld) in einer Anweisungszeile nicht komplett ausgefüllt werden muss. Hier gilt dann die Regel, dass Alphanumerische Einträge linksbündig, numerische Einträge rechtsbündig ausgefüllt werden.
Beispiel: In einer C-Zeile kannst Du einen Befehlsnamen angeben. Da diese Namen nicht immer die maximale Länge des entsprechenden Teils der Zeile haben, tippst Du sie linksbündig ein. Gibst Du zum Beispiel bei einer Variablendefinition die gewünschte Länge numerisch ein, so hat dies rechtsbündig zu geschehen.
Jetzt erst mal genug der Theorie.

[I] Wer die Spalten 1 bis 5 trotzdem benutzen will, sollte sich die F-Tasten F19 und F20 im SEU anschauen.
[II] Ausnahme: Ab Betriebssystem V5 gibt es einer Erweiterung namens RPG/free, in dem viele Zeilenarten in einem freien Format eingegeben werden können, wie man es von anderen Programmiersprachen kennt.

Ran ans Programm

Solltest Du noch den Bildschirm „Mit Teildateien arbeiten (mittels PDM)“ vor Dir haben, bitte diesen mit F12 verlassen. Wir müssen nun in unserer Bibliothek eine weitere Datei für Quellentexte erstellen, und zwar für unsere RPG-Programme, da diese Datei vermutlich noch nicht existiert. Wenn doch, überspringe die folgenden Zeilen.

Ein Platz für Programmquellen

Um eine neue Quellendatei zu erstellen, benutzen wir wieder den Befehl

```
CRTSRCPF
```

Diesen Befehl hatten wir bereits für unsere DDS-Datei verwendet, daher schreibe ich nur den Befehl mit allen nötigen Parametern für unsere RPG-Datei:

```
crtsrcpf FILE(DeineLib/QRPGLESRC) RCDLEN(120) TEXT('Meine RPGs')
```

Auffällig ist hier, dass ich als Satzlänge (Parameter RCDLEN) den Wert „120“ angegeben habe, im Gegensatz zu den „112“ für die DDS-Quellen. Dies hat mehr einen kosmetischen Charakter, somit haben wir am Zeilenende mehr Platz für kurze Kommentare.

Nachdem der Befehl ausgeführt wurde, aktualisiere bitte den Bildschirm im PDM einmal mit der Taste F5, nun siehst Du auch den entsprechenden Eintrag. Bitte mit der Auswahl 12 in diese Datei wechseln, der Inhalt ist selbstverständlich leer.

Eine neue Quellendatei

Also müssen wir mit F6 eine neue Teildatei anlegen. Im folgenden Bildschirm müssen wir unsere Eingaben tätigen. Bitte trage als Quellenteildatei ein: „SETADRSTAT“, als Quellenart „RPGLE“ und als Text „Setzen Adress-Status“. Nach dem Bestätigen landest Du im leeren SEU-Editierfenster.

Nun geht es los. Ein RPG-Programm besteht wie eine DDS-Quelle aus vielen spaltenorientierten Zeilen. Allerdings gab es bei einer Dateibeschreibung nur eine Zeilenart, nämlich „A“. Bei einem RPG-Programm gibt es da schon verschiedene Möglichkeiten, und sie müssen in einer bestimmten Reihenfolge eingetragen werden.

Der leere Editorbildschirm sieht wie folgt aus:

```
Spalten . . . :   6  76          Editieren                    DeineLib/QRPGLESRC
 SEU==> _____________________________________________________________  SETADRSTAT
 FMT H  HSchlüsselwörter++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        ****************Datenende*****************************************


 F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
 F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
Teildatei SETADRSTAT zu Datei DeineLib/QRPGLESRC hinzugefügt        +
```

Start einer Editiersitzung für RPG-Programme (ich habe einige Leerzeilen weg gelassen)

Wie Du bemerkst, ist die Format-Zeile (3. Zeile, beginnt mit „FMT H“) derzeit noch sehr einfach. Als erste Zeilenart in einem RPG-Programm kann, *muß* aber nicht, der Typ „H“ erscheinen. H steht für Header-Angaben und definiert Werte, die für das gesamte Programm gelten. Man kann im Datenbereich nur H und darauf folgende Schlüsselwörter (Befehlswörter) eingeben. Zur Veranschaulichung verwenden wir eine H-Spezifikation:

```
    HDEBUG
```

Beachte bitte, dass Du das H unter dem zweiten gleich lautenden Buchstaben in der Formatzeile angibst, also direkt vor der Spalte „Schlüsselwörter“ (hast Du an Deiner Emulation eine Anzeige für aktuelle Zeile/Spalte des Cursors, ist dies Spalte 9).

Direkt hinter dem H folgt das Schlüsselwort „DEBUG“. Dieses sagt dem Compiler später, dass wir einen speziellen Befehl verwenden können, um im Fehlerfalle ein Auszug aller Programmvariablen zu erhalten. Ist zwar jetzt für unsere Übung nicht sehr spannend, aber nun haben wir eine schnuckelige H-Zeile, um deren Inhalt wir uns aber jetzt nicht weiter kümmern, Du kannst die Zeile auch weg lassen.

EVA – immer der Reihe nach

Gemäß der alten EDV-Regel „E – V – A“ (Eingabe, Verarbeitung, Ausgabe) müssen wir nun bestimmen, woher das Programm seine Eingaben erhält. Dafür gibt es die F-Bestimmungen, mit der wir Files / Dateien angeben, mit denen unser Programm arbeitet. Die F-Bestimmungen haben natürlich einen total anderen Aufbau als die H-Bestimmung. Damit wir wissen, wie dieser Aufbau aussieht, sagen wir dem SEU, er soll uns eine Formatzeile vom Typ „F“ anzeigen. Dazu bewege bitte den Cursor links in den leeren Bereich für eine Zeilennummer und Tippe ein „FF“ gefolgt von DF. Nun fügt SEU eine Formatzeile zu unserer Information ein. Du siehst, eine F-Bestimmung ist schon etwas komplizierter. Tippe bitte in die erste Datenspalte ein F und verwende dann den Prompter mit der F4-Taste.

Der Bildschirm ändert sich wie folgt:

```
Spalten . . . :   6  76          Editieren                    DeineLib/QRPGLESRC
 SEU==> ____________________________________________________________ SETADRSTAT
 FMT F  FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0002.00 F
        ****************Datenende*****************************************
Art d. Bed.führung    F         Folgenummer . . . . . 0002.00

           Datei-     Datei-      Datei-      Hinzu.
Dateiname  art     verwendungsart  ende     neuer Sätze  Reihenfolge
__________ _       _           _           _            _
Datei-    Satz-     Verarb.-    Länge des      Art der
format    länge    in Grenzen  Schlüsself.   Satzadressier.
  _      _____        _          _____           _
   Art d.
Dateiorganisation   E/A-Einh     Schlüsselwörter
      _             _______      ____________________________________
Bemerkungen
____________________________

 F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
 F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Bedienerhilfe für eine F-Bestimmung

Wie Du siehst, kann man einer Dateiangabe eine Menge Parameter mit auf den Weg geben. Die meisten davon können wir vorerst ignorieren. Natürlich kannst Du bei Interesse wieder auf jedem Feld die Hilfetaste drücken, um zu erfahren, was dahinter steckt.

In das Feld DATEINAME tragen wir `ADREPF` ein. Wir arbeiten zunächst direkt mit der physikalischen Datei, da wir keine Sortierlogik oder einen Zugriff über Schlüsselwerte benötigen, wenn wir alle Datensätze bearbeiten wollen. Der Dateiname kann maximal zehn Stellen lang sein (Spalten 7 bis 16). Dateinamen auf der AS/400 müssen generell mit einem alphabetischen Zeichen oder dem Sonderzeichen #, $ oder § beginnen. Die restlichen neun Zeichen dürfen alphanumerisch, numerisch oder aus den vier Sonderzeichen #, $, § oder _ bestehen. Du kannst zwar Gross/Kleinschreibung mischen, die AS/400 interessiert sich aber nicht für den Unterschied. Man sollte hier optische Gesichtspunkte verwenden :)

Als DATEIART müssen wir in Spalte 17 den Buchstaben `U` angeben, da wir Datensätze in dieser Datei aktualisieren (=*U*pdate) wollen. Würden wir nur lesen, müsste hier ein `I` (für *I*nput) stehen.

Als nächstes Feld kommt die DATEIVERWENDUNGSART in Spalte 18. Hier wird zwingend bei Eingabedateien angegeben, auf welche Art und Weise Sätze gelesen werden sollen.

Alte RPG-Programme hatten oft eine ähnliche Funktionsweise wie unser erstes Beispielprogramm: Alle Sätze lesen, bearbeiten und eventuell wieder schreiben. Daher gab es in RPG einen speziellen Modus, in dem man die Lese- oder Schreibschleife nicht selbst programmierte, sondern dem System überließ und nur die Verarbeitungsregeln für jeden Satz angab. Diesen sogenannten RPG-Zyklus konnte man nur dafür verwenden, und musste sich nicht um Satzadressierung kümmern. Nachteilig war die Unflexibilität dieser Methode, wenn man gezielt Sätze ansprechen will. Da wir uns selbst um den Zugriff auf jeden einzelnen Satz kümmern wollen, geben wir bei der Verwendungsart ein „F“ ein. Dies bedeutet „Vollprozedural“, also programmgesteuert. Wir müssen also eine Schleife programmieren, die alle Sätze liest.

Das nächste Feld, das einen Wert benötigt, ist das DATEIFORMAT in Stelle 22. Hier wird definiert, ob die Datei intern oder extern beschrieben ist. Früher konnte man das Format der Felder in einer Datei nur im Programm definieren, mit dem Ergebnis, dass bei einer Dateiänderung alle Programme angepasst werden mussten. IBM führte die Möglichkeit ein, Dateien und Programme zu trennen, das Programm bekam bei der Erstellung alle Informationen über das Dateiformat, da es in der Datei gespeichert wurde. Dies nennt man „extern beschrieben“. Da man heutzutage eigentlich nur noch mit externen Dateien arbeitet, gehört hier der Buchstabe `E` hinein.

Letztlich müssen wir bei `E/A-EINH.` (Stellen 36 bis 42) noch den Wert `DISK` eintragen, da unsere Datei natürlich irgendwo auf den Festplatten bzw. im Einstufigen Speicher liegt.

Somit sollte die Dateispezifikation so aussehen:

```
 FMT F  FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter+++++++++++++++++
0002.00 FADREPF    UF   E             DISK
```

Man darf sich hier nicht von den vielen Möglichkeiten verwirren lassen. Mit den modernen ILE-Methoden benötigt es nur weniger Angaben. Dann kommt man schnell zum Ergebnis.

Nachdem wir nun definiert haben, mit welcher Datei wir arbeiten, brauchen wir noch ein paar Befehle, die mit den darin enthaltenen Daten auch etwas (hoffentlich sinnvolles) anstellen.

Dafür braucht es einige Spezifikationen vom Typ „C“ wie Command. Diese Zeilentypen Bestehen im Grunde nur aus 4 Feldern. Tippe einfach in das nächste, freie Zeilennummernfeld den Befehl FC, und Du siehst eine Formatierungshilfe für C-Spezifikationen. Du siehst, dass die vier größten Spalten überschrieben sind mit „FAKTOR1“, „OPCODE“, „FAKTOR2“ und „ERGEBNIS“. Diese vier Befehlsteile bilden fast alles, was ein Command benötigt. Mit ILE braucht man fast immer nur den OpCode und entweder Faktor2 oder Ergebnis. Hintergrund dieser Benennung ist, dass in alten RPG-Varianten jeweils nur kurznamige Felder angegeben werden konnten. Wollte man den Inhalt von „FeldA“ und den Inhalt von „FeldB“ addieren und das Ergebnis in „FeldC“ schreiben, musste man programmieren:

```
      FELDA        ADD    FELDB        FELDC
```

Etwas unübersichtlich, aber man kann damit leben. Mit RPG4 gibt es viele neue OpCodes, und diese können in Faktor2 und Ergebnis einen gesamten Ausdruck enthalten. Dies sieht dann so aus:

```
                   EVAL   FELDC = FELDA+FELDB
```

Das liest sich doch gleich viel einfacher, oder? :-) Trotzdem musst Du die Spaltenorientierung beachten, der OpCode benötigt immer noch seinen festen Platz.

Weiterhin bemerkst Du in der Formatierungszeile ganz am rechten Rand die Felder „HO“, „NI“ und „GL“. (Wenn Du Dich bis jetzt gewundert hast, jedes neue Feld fängt mit einem Großbuchstaben an.)

Hier definiert man sogenannte Bezugszahlen, die bei einigen Operationen wie Dateioperationen gesetzt werden. Was sind um Himmels Willen Bezugszahlen? So nennt man bei IBM Variablen mit einem zweistelligen Namen, die nur den Wert „An“ oder „Aus“ enthalten können, also Boolsche Variablen. Deren Namen kann von „00“ bis „99“ sowie einigen Sonderwerten lauten. Mit diesen Bezugszahlen kann man eine Menge Schweinereien anstellen, zum Teil elegante Routinen, zum anderen auch ein Programm so unleserlich machen, dass es einem anderen Programmierer schwindelig wird. Wichtig sind diese Bezugszahlen für die Programmierung von Bildschirmen und Druckausgaben. Aber auch hier gilt: Eins nach dem Anderen…

Zunächst müssen wir überlegen, was unser Programm tun soll. Wir wollen alle Sätze der Datei lesen, von Anfang bis Ende. Bei jedem Satz wird geprüft, ob das Feld „ADRORT“ einen Wert enthält oder nicht. In letzterem Fall soll das Feld „ADRSTAT“ den Wert „U“ für „ungültig“ erhalten und der Satz in der Datei aktualisiert werden. Fertig ☺

Um mögliche Fehler abzufangen, müssen wir von vorneherein prüfen, ob in der Datei überhaupt Daten stehen. Dies kann man am besten dadurch erledigen, dass man vor Beginn der Schleife liest und prüft, ob ein Fehler aufgetreten ist. Und um einen Satz zu lesen, braucht es einen Befehl. Die ganze Problematik lässt sich recht einfach durch einen Befehl namens „READ“ lösen, der im Fehlerfall eine Bezugszahl auf „an“ setzt.

Tippe nun in die nächste freie Zeile in die erste Spalte ein „C“ und drücke die Prompt-Taste F4. Das Bild sieht ungefähr so aus:

```
Spalten . . . :   6  76          Editieren                    DeineLib/QRPGLESRC
 SEU==> ____________________________________________________________ SETADRSTAT
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis+++++Län++D+HoNiGl
0002.00 FADREPF    UF   E             DISK
0003.00 C
        ****************Datenende***************************************************
Art d. Bed.führung    C         Folgenummer . . . . . 0003.00

GrpStufe N01 Faktor 1       Operation   Faktor 2        Ergebnis
 __      ___ ____________   __________  ______________  ______________
           Dezimal-
Länge      stellen      Ho   Ni   Gl   Bemerkungen
_____        __         __   __   __   ____________________

 F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
 F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Prompten einer C-Bestimmung

Wir können die ersten 3 Felder ignorieren und geben unter „Operation“ ein: „READ“, unter „Faktor2“ dann „ADREPF“. Wenn Du nun auf DF drückst, meckert das System an, dass der Eintrag für die Ergebnisbezugszahl für „GLEICH (GL)“ leer ist.

Beziehungsprobleme

Der Cursor steht an entsprechender Stelle und das System erwartet eine Eingabe. Du kannst mit der Hilfetaste die drei Bezugszahlenfelder „Ho“, „Nı“ und „Gl“ durchforschen. Leider hat sich im Laufe der Entwicklung so einiges getan, so dass IBM diese drei Felder mehrfach belegen musste. Dadurch ist die Verwendung der drei Bezugszahlenfelder etwas undurchsichtig geworden. Im Anhang werde ich eine Liste schreiben, die darstellt, welche der drei möglichen Bezugszahlen bei welcher Situation eingeschaltet wird.

Bei unserem Befehl „READ“ zum Beispiel wird die dritte BZ[1] „Gl“ gesetzt, wenn Du nach dem letzten Satz der Datei lesen wolltest. Da wir dies für unsere Zwecke gebrauchen können, und der Befehl sowieso an dieser Stelle eine Bezugszahl benötigt, geben wir hier eine ein. Nehmen wir die Bezugszahl „80“. Es hat sich eingebürgert, gewisse Bereiche an Bezugszahlen für spezielle Funktionen zu verwenden. Bitte nicht davon total verwirren lassen. „80“ ist eine von mir gewählte Zahl, Du könntest auch „42“ nehmen, wenn dies all Deine Fragen beantwortet…

Drücke nun Datenfreigabe und verlasse das Prompten mit F12. Deine Zeile 3 sollte nun so aussehen:

```
FMT C CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis+++++Län++D+HoNiGl
    C                     READ      ADREPF                                 80
```

Auf Deutsch würde dieser Befehl ungefähr lauten: „Lese einen Satz aus der Datei ADREPF, und wenn das Dateiende erreicht ist, setze Bezugszahl 80 auf ´an´.“ Eigentlich ganz einfach, oder?

Nun haben wir also den ersten Satz gelesen. Sollte kein Fehler (Dateiende=leere Datei) aufgetreten sein, können wir in einer Schleife weiterarbeiten. Die Befehle hierfür sehen so aus:

```
FMT C CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis+++++Län++D+HoNiGl
    C                     DOW       *IN80=*OFF
    C                     IF        ADRORT=*BLANKS
    C                     EVAL      ADRSTAT='U'
    C                     UPDATE    ADRFMT
    C                     ENDIF
    C                     READ      ADREPF                                 80
    C                     ENDDO
```

Schritt für Schritt

Der Reihe nach:

Wir beginnen nun eine Schleife. „DOW“ bedeutet: „Do while“ (tue solange…), die dazugehörige Prüfung lautet „*IN80=*OFF“. Hier wird die Bezugszahl 80 geprüft. Beachte die Schreibweise mit „*IN“ davor, dies ist eine Eigenart von ILE. Man könnte hier jede beliebige Variable oder jedes Feld prüfen, Bezugszahlen werden durch diese Schreibweise adressiert. „*OFF“ bedeutet „aus“. Eine Bezugszahl kann nur „an“ (*ON) oder „aus“ (*OFF) enthalten. Die Schleife läuft also solange, wie die Bezugszahl 80 nicht gesetzt ist, also solange wir nicht das Dateiende erreicht haben.

[1] Aus Tippfaulheit kürzt man „Bezugszahl“ gern mit „BZ“ ab.

Im Falle einer Schleife...

Ach ja: Die Bezugszahl im zweiten READ vor dem ENDDO nicht vergessen, sonst gibt's eine schöne Endlosschleife ☺ In diesem Falle hilft nur eins: das Programm abbrechen.

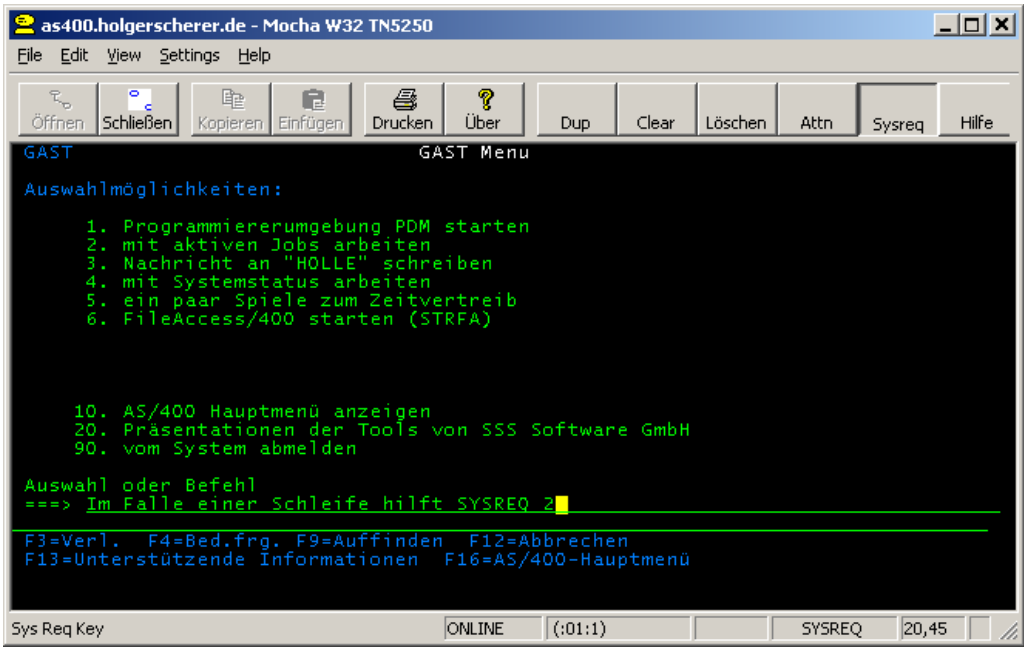
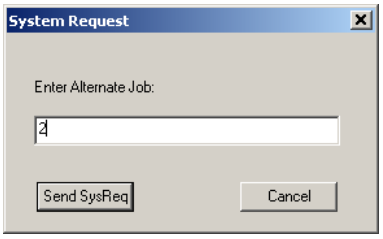


Abbildung 16 : Solltest Du ein Programm haben, das in einer Endlosschleife läuft, musst Du es hart abbrechen

Wie das geht? Im Mocha-Terminal auf den „Sysreq" Knopf klicken und im folgenden Fenster



eine Zahl `2` eingeben. Schon wird das laufende Programm getötet. Wenn Du keine Zahl eingibst, bekommst Du noch ein paar weitere Möglichkeiten. Ob Du herausfindest, was das ist, überlasse ich Deinem Experimentierdrang :-)
Und wenn Du ein anderes TN5250-Programm hast, suche bitte, wie man die Taste „SystemRequest" betätigt. (Evtl. hilft auch Esc oder Strg+Esc.)

Du kannst zwar auch einfach Dein Terminalprogramm schliessen, aber sowas ist dem System gegenüber unhöflich! Nicht, dass etwas schlimmes passieren würde, Du bekommst nur evtl. ein grösseres Job-Protokoll. (siehe: `WRKSPLF`)

Frage und Abfrage

Im nächsten Befehl (`IF`) beginnt eine konditionale Abfrage. Wenn das Feld `ADRORT` nicht belegt ist (`*BLANKS` ist ein Synonym für beliebig viele Leerstellen, was einem leeren Feld recht nahe kommt), soll alles bis zum nächsten `ENDIF` ausgeführt werden.

In dieser Abfrage setzen wir mit dem Befehl `EVAL` das Feld „`ADRSTAT`“ auf den Wert „U“. Damit unsere Änderung auch in der Datei landet, benötigen wir den Befehl `UPDATE`, der den geänderten Satz in die Datei schreibt. Beachte hier, dass als Faktor2 nicht der Dateiname wie bei `READ`, sondern der Name des Satzformates verwendet wird. Dies hat sogar einen gewissen logischen Sinn, da wir mit einem Update-Befehl ja nicht eine Datei an sich aktualisieren, sondern nur den aktuellen Satz, den wir vorher gelesen haben.

Nun lesen wir mit dem bekannten `READ`-Befehl den nächsten Satz und beenden die Schleife. Sollte das Dateiende erreicht werden, wird Bezugszahl 80 auf „an“ gesetzt und die Schleife beendet.

Das war eigentlich schon das gesamte Programm. Nun noch eine Besonderheit von RPG: Aufgrund des vorhin angesprochenen RPG-Zyklus, in dem das durchgängige Lesen der Datei von RPG behandelt wurde, müssen wir auf jeden Fall (!) beim Ende eines Programms dem System auch mitteilen, dass nun Ende ist. Dafür gibt es die besondere Bezugszahl namens „LR“ (last record), die auf „an“ gesetzt werden muß. Diese Bezugszahl sagt dem System, dass das Programm nun endet. Um nun diese Bezugszahl auf „an“ zu setzen, benötigen wir schließlich den Befehl:

```
            EVAL        *INLR=*ON
```

Man beachte die Schreibweise der Bezugszahl

Unser vollständiges Programm sieht nun so aus:

```
Spalten . . . :   6  76          Editieren                    DeineLib/QRPGLESRC
 SEU==> ____________________________________________________________ SETADRSTAT
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis+++++Län++D+HoNiGl
0002.00 FADREPF    UF   E             DISK
0003.00 C                    READ      ADREPF                                 80
0004.00 C                    DOW       *IN80=*OFF
0005.00 C                    IF        ADRORT=*BLANKS
0006.00 C                    EVAL      ADRSTAT='U'
0007.00 C                    UPDATE    ADRFMT
0008.00 C                    ENDIF
0009.00 C                    READ      ADREPF                                 80
0010.00 C                    ENDDO
0011.00 C                    EVAL      *INLR=*ON
        ****************Datenende***************************************

 F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
 F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Unser erstes RPG-Programm!

Dieses Programm ist eigentlich gar nicht so schwer zu verstehen, wenn man die kleinen Besonderheiten der Spaltenorientierung verstanden hat. (Wem die Eingabe schwerfällt, sucht in den Einstellungen seines Terminalprogramms nach einer Einstellung namens “Crosshair Cursor”, mit diesem Fadenkreuz zielt es sich besser).

Mit einem Satz gesagt, macht es nichts anderes, als alle Sätze der Datei ADREPF zu lesen bis zum Dateiende. Wenn in einem Satz das Feld ADRORT leer ist, wird das Feld ADRSTAT in diesem Satz auf „U“ gesetzt und der Satz aktualisiert. Dann geht's zum nächsten Satz. Fertig.

(Jajaja, das könnte man auch mit einem SQL-Befehl erledigen. Aber wir kommen nachher zu Dingen, die man mit SQL nicht mehr so einfach hinbekommt).

Wandeln und testen

Um nun das Programm umzuwandeln, damit es ausgeführt werden kann, beendest Du nun den SEU mit der Taste F3, so dass Du wieder in der Liste der Teildateien landest. Nun schreibst Du vor den Namen Deiner Programmquelle die Auswahl 14 (Umwandeln, siehe auch F23). Nach einer gewissen Arbeitszeit wird Dir das Ergebnis Deiner Mühen gezeigt, entweder eine Meldung über die erfolgreiche Umwandlung, oder ein Fehler. Wie auch beim Erstellen einer Datei kannst Du mit WRKSPLF das Protokoll einsehen.

Prüfe nun vor dem Starten des Programms mit SQL den Inhalt der Datendatei:

```
STRSQL
```

im SQL gibst Du ein:

```
SELECT * FROM ADREPF
```

und schau Dir die Daten genau an.

Nun beendest Du den SQL mit F3 und DF. Dann rufst Du das Programm auf mit:

```
CALL SETADRSTAT
```

Prüfe danach den Inhalt der Datei wieder mit SQL. Änderungen bemerkt? Gut, wir haben unser erstes Programm mit Datenveränderungen zum Laufen gebracht :-)

Nach dem Du die Arbeitsweise des Programms verinnerlicht hast, gehen wir gleich weiter. Ein Computer lebt davon, dass er die Daten nicht nur verarbeiten kann, sondern auch die Möglichkeit bietet, die Daten irgendwie auszugeben. Da man die Informationen nicht immer nur im Büro am Bildschirm benötigt, sondern auch mal unterwegs, muss man diese irgendwie zu Papier bringen.

Ja, man könnte auch mit der Kamera Photos vom Bildschirm machen, aber das ist jetzt keine wirklich sinnvolle Idee :-)

Wir machen Druck!

Schreiben wir also nun ein Prográmmelchen, welches den Inhalt unserer Adressdatei ausdruckt. Auf der AS/400 geschieht das in einer besonderen Weise: Kein Programm kann wirklich „direkt“ drucken. Jedes Programm, welches eine Druckausgabe erzeugt, produziert ein so genanntes *SpoolFile*, zu Deutsch am besten mit „Druckausgabedatei“ zu übersetzen. Das System kümmert sich um all diese Druckausgaben, und der Anwender und der Programmierer können steuern, wie damit umgegangen werden soll. Es kann festgelegt werden, zu welchem Drucker die Ausgabe gesendet werden soll, ob direkt gedruckt werden soll oder die Druckausgabe erst mal im Wartemodus bleibt etc...

Druckausgaben werden in *Ausgabewarteschlangen*, (englisch „OutQueue“) sortiert. Es kann vom Administrator definiert werden, ob jeder Benutzer seine eigene Warteschlange hat, oder pro Gruppe oder gar für alle Benutzer ein und die selbe Ausgabewarteschlange verwendet werden soll etc. Kümmern wir uns jetzt mal nicht so sehr um alle Details, sondern schreiben erst mal ein Programm, das Druckdaten ausgibt.

Ein neues Programm!

Ein ganz einfaches Druckprogramm wäre: Wir drucken alle Sätze aus unserer Datei `KONTOPF`. Nicht wirklich spektakulär, aber lehrreich.

Beginne also im SEU-Bildschirm „`mit Teildateien arbeiten`“ in der Quellendatei `QRPGLESRC` eine neue Teildatei namens `PRTKONTO`. Wie das noch mal geht? Einfach:

Ich gehe davon aus, dass Du Dich noch in diesem SEU-Bild befindest, da Du vorhin schon eine Teildatei mit einem RPG-Programm bearbeitet hast. Nun drückst Du einfach wieder F6, und füllst den Bildschirm „`SEU starten`“ wie folgt aus:

```
                              SEU starten (STRSEU)

Auswahl eingeben und Eingabetaste drücken.

Quellendatei . . . . . . . . . . > QRPGLESRC     Name, *PRV
  Bibliothek . . . . . . . . . . >   LIBMANTEST  Name, *LIBL, *CURLIB, *PRV
Quellenteildatei . . . . . . . .   PRTKONTO      Name, *PRV, *SELECT
Quellenart . . . . . . . . . . .   RPGLE         Name, *SAME, BAS, BASP...
Text 'Beschreibung'  . . . . . .   Drucken Kontenliste
_________

                                                                      Ende
F3=Verlassen   F4=Bedienerf.   F5=Aktualisieren   F12=Abbrechen
F13=Verwendung der Anzeige     F24=Weitere Tasten
```

Neue Teildatei erstellen

Nach Bestätigen mit DF erhälst Du wie gewohnt einen leeren SEU-Bildschirm.

Bevor wir jetzt ein Programm hirnlos abtippen und anschauen, gehe ich bei diesem Beispiel noch einmal entsprechend der Logik Schritt für Schritt vor.

Fangen wir mit dem einfachsten an:

Was wollen wir drucken?

Klar, wir wollen den Inhalt unserer Kontodatei drucken. Also müssen wir diese auch in unserem Programm definieren. Diesmal aber als reine Eingabedatei, nicht – wie im vorigen Beispiel – als Datei zum Aktualisieren.

Fange also mit der folgenden Zeile im SEU an:

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                        PRTKONTO
 FMT FX FDateiname+IPEASF.....L.....A.E/AEinh.Schlüsselwörter++++++++++++++++++
0001.00 FKONTOPF   IF   E             DISK
```

Erster Anfang, unsere Eingabedatei

Wenn Du Dich mit den Spaltenüberschriften der `FMT`-Zeile beschäftigst, oder die Zeile mit F4 promptest, erkennst Du die bekannten Felder für die **F**-Bestimmung wieder:
Als `Dateiname` geben wir bekanntermassen `KONTOPF` an. Die `Dateiart` (unter Spalte `I`) ist jetzt aber diesmal ein `I` für Input (*Eingabedatei*). Unter Spalte `P` (`Datei-Verwendungsart`) schreiben wir ein `F` für *vollprozedural*, somit kümmern wir uns wieder selbst um das Lesen der Sätze.
Es ist manchmal schön, manchmal auch verwirrend dass der SEU in der Formatierungszeile sehr brauchbare Spaltenüberschriften vorgibt. Aber wie soll man auch einen Spaltentitel sinnvoll auf eine Stelle kürzen können? :-)
Schliesslich belegen wir noch die Spalte `Dateiformat` mit einem `E`, um eine extern beschriebene Datei anzugeben, sowie als `E/A-Einheit` wie bekannt `DISK`. Ist doch verständlich. Somit haben wir also eine Datei, aus der wir lesen wollen.

Kommen wir zur nächsten Datei. STOP! Was für eine nächste Datei? „Wir haben doch nur unsere Kontodatei!“ höre ich Dich schon rufen. „Du hast Recht“, werde ich Dir antworten, aber...
Wir wollen unsere Daten ausdrucken. Wie ich schon schrieb, landet jede Druckausgabe in einer Spooldatei. Was sagt uns das? Wir brauchen eine Ausgabedatei, wo hin wir die Daten schreiben, die später auf dem Papier landen sollen.

Wohin drucken wir?

Definieren wir also eine Datei, in die wir schreiben. Nun, ähm, welche Datei? Das Konzept von OS/400 ist hier recht komplex und die Philosophie von der AS/400 trennt Hardware von Software. Wir können also nicht direkt einen Drucker ansprechen. IBM hat dafür die so genannte Druckerdatei geschaffen. Diese Druckerdatei dient als Ziel für die Ausgabe und ist ein normales OS/400-Objekt mit vielen Attributen, wie „welche Druckwarteschlange wird verwendet?“, „wie ist das Papierformat?“, „welche Formatierungsregeln gibt es?“ und so weiter. Wir lernen später, Druckerdateien wie einen Bildschirm zu definieren, um die Formatierung der Druckausgabe extern, also nicht im Programm zu erhalten. Damit kann man dann das Aussehen eines Formulars ändern, ohne im Programm rumfummeln zu müssen.
Zunächst benutzen wir aber erst eine Programm-interne Formatierung der Ausgabe. Da wir dafür eine Druckerdatei benötigen, die keine Formular-Definitionen enthält, nehmen wir eine simple Druckerdatei. IBM stellt uns hierfür eine Datei namens `QPRINT` zur Verfügung. Diese Datei kann man immer nehmen, wenn Du keine grossen Ansprüche hast und Dich im Programm selbst um das Aussehen der Druckausgabe kümmern willst.

Geben wir also als zweite Zeile in unser Programm wie folgt ein:

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                            PRTKONTO
 FMT F  FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0002.00 FQPRINT    O    F   80        PRINTER
```

Unsere zweite Datei

Der `Dateiname` ist `QPRINT`, die `Dateiart` diesmal `O` (für *Output*). Eine `Dateiverwendungsart` müssen wir nicht angeben (siehe Hilfetext für dieses Feld, wir können es leer lassen). Aber als `Dateiformat` müssen wir nun ein `F` eingeben, was „*programmbeschrieben*“ bedeutet (in der Druckerdatei `QPRINT` sind keinerlei Informationen über das Druckformular enthalten, es gibt nur ein Feld, das über die gesamte Zeilenlänge gültig ist).

Da der Compiler also nichts über das Format der Druckausgabe aus `QPRINT` erhalten kann, müssen wir im Feld `Satzlänge` den Wert `80` eintragen (achte auf rechtsbündige Eingabe!). Schliesslich muss als `Einheit` noch `PRINTER` eingetragen werden. Noch Fragen? Ok, keine, dürfte verständlich sein.

Damit wäre schon mal ein grosser Schritt getan. Bevor wir nun die Verarbeitungsbefehle (also die C-Bestimmungen) angeben, beschäftigen wir uns erst mit dem Layout der Druckausgabe. Dieses wird in so genannten O-Bestimmungen in das Programm eingetragen. Diese O-Bestimmungen müssen aber am *Ende* des Programmtexts stehen, achte also darauf, dass Du nachher die C-Zeilen zwischen F- und O-Anweisungen einträgst (mittels des Zeilenbefehls `I` für *Insert*).

Warum ich erst mit den O-Anweisungen anfange? In diesen Anweisungen werden die zur Ausgabe nötigen Definitionen mit Namen eingetragen, auf die wir uns später in den C-Anweisungen beziehen. Um also Verwirrung zu vermeiden, fange ich mit dem Schluß an ;-)

Alle O-Anweisungen (hier steht eben ein O in Stelle 6) beinhalten zwei verschiedene Zeilenarten. Sie fangen zwar mit O an, unterscheiden sich aber im Aufbau. Das mag am Anfang sehr verwirrend erscheinen (ist es ja auch), aber man kann sich arrangieren. Die Zeilenarten sind: Satzbeschreibungen, die sich mit der Ausgabe auf Satzebene befassen, und Feldbeschreibungen, die den Inhalt eines gewünschten Ausgabesatzes darstellen.
Stelle Dir eine Druckerdatei wie eine Datendatei vor. Es gibt aber in einer Druckerdatei immer mehrere Satzformate, wie zum Beispiel für die Seitenüberschrift, den Teil mit den Spaltenüberschriften und die Detailformate, die das Aussehen für eine einzelne Datenzeile beinhalten.
Schauen wir uns dies am gewünschten Ergebnis an. Unser Ausdruck soll später wie folgt aussehen:

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890

  Seite xxx                        Kontenliste                 Datum: tt.mm.jj

 Konto-    Kontenbeschreibung               Anlagedatum
 Nummer

 xxxxxx    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   tt.mm.jj
 xxxxxx    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   tt.mm.jj
 xxxxxx    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   tt.mm.jj
 xxxxxx    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   tt.mm.jj
...
```

Unsere gewünschte Druckausgabe

(Die ersten zwei Zeilen stellen nur als Spaltennummern eine Hilfe für uns dar und sollten natürlich nicht auf dem Papier erscheinen.)

Wir hätten also drei Satzformate für die Druckerdatei. Zum Einen die Seitenüberschrift mit Seitennummer, Titel des Reports und dem Datum, an dem die Druckausgabe erzeugt wurde. Danach kommt eine Leerzeile, dann das Satzformat mit den Überschriftszeilen für die einzelnen Felder, und letztlich die einzelnen Detailzeilen für jedes auszugebende Konto.

Nagut, wir könnte nauch die beiden ersten Satzformate zusammen fassen, da sie eh nur einmal am Seitenanfang auftauchen. Aber ein bisschen Spass muss sein!

Formate und Layout

Schauen wir uns die O-Zeilen für das erste Druck-Satzformat an:

```
FMT O   ODateiname+DF..N01N02N03ExceptnameB++A++Vv+Vn+.....................
0036.00 OQPRINT    E            SEITENKOPF        2  4
0037.00 O                                         7 'Seite'
0038.00 O                       SeiteNr          11
0039.00 O                                        45 'Kontenliste'
0040.00 O                                        68 'Datum:'
0041.00 O                       UDATE         Y  77
```

So drucken wir den Seitentitel (bitte die Zeilennummern ignorieren)

Woah, das sieht ja schon recht herb aus! Gib in die erste Zeile direkt OQPRINT ein (also die Stelle für die Zeilenart und den Namen der Druckerdatei), und drücke dann F4 zum Prompten. Daran erkennt der SEU, welche der beiden Arten von O-Zeilen Du eingeben willst und formatiert entsprechend.
Zunächst haben wir hier das Feld DATEINAME, in das selbstverständlich der Name unserer Druckerdatei geschrieben wird. In das Feld ART (Spaltenüberschrift hier D) müssen wir ein E eintragen. Warum E, wenn im Hilfstext für dieses Feld andere Werte für Kopf- oder Postensätze angeregt werden? Nun, wir haben bei der Festlegung der Eingabedatei (also KONTOPF) festgelegt, dass wir *vollprozedural* arbeiten, also nicht den RPG-Zyklus verwenden wollen. Das bedeutet auch, dass wir die Druckausgabe vollprozedural durchführen müssen und jedes Ausgabeformat selbst „anstossen“ müssen.[I]
Daher definieren wir mit dem E so genannte *EXCPT* (oder EXCEPT)-Sätze. Diese Bezeichnung kommt daher, da der entsprechende Befehl zur Ausgabe eines Drucker-Satzformates so heisst. Wir geben also jedem Satzformat einen Namen, unter dem wir später das Format zur Ausgabe an die Druckdatei senden. Den entsprechenden Namen tragen wir unter EXCEPTNAME in die Stellen 30 bis 39 ein. Noch eins zur Totalverwirrung: Man *muss* keinen Except-Namen angeben, Du könntest auch steuern, welches Format gedruckt werden soll, durch das Setzen entsprechender Bezugszahlen (Du hast bestimmt die entsprechenden Felder beim Prompten gesehen). Aber! Damit kriegst Du garantiert jedes noch so kleine Programm unübersichtlich gestaltet. Also lassen wir es, das Mittelalter ist langsam vorbei.

Dafür kannst Du aber originellerweise mehrere Formate mit dem gleichen Except-Namen definieren, mit dem Ergebnis, dass Du mit einem Except-Befehl mehrere Formate am Stück ausgeben kannst. Praktisch!

Ich gebe hier also dem Format den Namen „Seitenkopf“. Nun kommen noch vier Felder, die es dicke in sich haben. Es handelt sich hier um die Steuerung der Zeilenschaltung und des Vorschubs. Bahnhof?

Ganz langsam. Da man beim Drucken nie so genau weiss, wieviel Zeilen im Detailbereich gedruckt werden, kann man relativ *und* absolut arbeiten. Das bedeutet, Du kannst angeben, wie viele Leerzeilen vor oder nach einem Format gedruckt werden sollen, oder zu welcher Zeilennummer vor oder nach dem Druck eines Formates gesprungen werden soll. Im Prompt siehst Du vier Felder. Der Zeilentransport *vor* und *nach* gibt an, wie viele Leerzeilen *vor* und/oder *nach* der Ausgabe des aktuellen Formats gedruckt werden sollen. Der Vorschub gibt an, zu welcher (absoluten!) Zeilennummer auf dem Papier *vor* und *nach* dem Druck des Formats gesprungen werden soll.

[I]Der besprochene RPG-Zyklus eignet sich – wie Du erkennen magst – hauptsächlich dafür, ohne grossen Aufwand einfache Berichte mit Kopf und Detailebene zu erstellen. Der Programmierer definiert nur eine Eingabedatei und die Ausgabeformate, das Lesen der einzelnen Sätze wird von RPG gesteuert. Ist praktisch, verleitet aber später zu unübersichtlicher Programmierung, weshalb ich diese Methode hier (vorerst) komplett ignoriere und gleich die für viele Programmierer gewohnte selbständige Programmsteuerung anwende.

Das mag anfangs etwas viel für unseren rauchenden Schädel sein, aber ist an sich logisch. Stelle Dir mal so einen einfachen Nadeldrucker vor. Die AS/400 hält Wacht über jede gedruckte Zeile. Wir wollen also einen Ausdruck beginnen. Üblicherweise macht man das auch am Anfang der Seite. Also schreiben wir in das Format für den Seitenkopf rein, dass der Drucker erst zur Zeile Nummer 1 (oder von mir aus 2) der Seite springen soll (was einen Seitenvorschub nötig machen kann). Dann wird die Zeile ausgegeben. Um den Kopf der Seite von den Überschriften zu trennen, sagen wir dem System, dass nach dem Seitenkopf der Drucker zur Zeile Nummer 4 springen soll. Wir machen diese Angaben absolut als Zeilen*vorschub*, da der Seitenkopf immer an der gleichen Stelle oben auf dem Papier erscheinen soll. Alle davon abhängigen Formate wie Überschriften und Detailzeilen werden wir relativ positionieren.

Nachdem wir die erste Zeile eingegeben haben und der Name des Ausgabeformats somit festgelegt ist, kommt in der nächsten Zeile schon der Inhalt des Formats. Wenn Du nach der ersten Zeile DF gedrückt hast, wechselt SEU automatisch die Spaltenformatierung, da sich diese bei der Format*inhalts*-Zeile von der Format*namens*-Zeile unterscheidet. Nicht durcheinander bringen lassen, wenn hier von Zeilen die Rede ist. Ich meine mit Druckzeile eine Zeile auf dem Papier. Mit Formatzeile sind hier die einzelnen Zeilen auf dem Bildschirm gemeint, die ein Druckformat beschreiben.

In dieser Formatzeile geben wir nur eine Konstante aus, nämlich den Text „`Seite`“. Dazu benötigt es neben der Angabe der Konstante noch die Angabe der rechten Spalte des ausgegebenen Objekts. Hm??

Ja! Du gibst bei einem zu druckenden Wert (sei es Konstante oder ein Datenfeld) hier nicht an, in welcher Spalte der Wert *beginnen* soll, sondern wo er *aufhören* soll. Interessant, oder? Grundlegender Gedanke von IBM bei der Festlegung dieser Darstellung ist die Tatsache, dass im Falle von zwei nebeneinander definierten Feldern wohl immer das linke Feld die Beschreibung des rechten Felds ist; wie es in unserem Falle ja auch ist. Links steht der Wert „`Seite`“, rechts der Inhalt vom Feld Feld „`SeiteNr`“. Um bei der Formularplanung auf einem Blatt Papier nicht immer die Länge der einzelnen Felder abzählen zu müssen, gibt man eben das Ende jedes ausgegebenen Feldes an, da fast alle gedruckten Felder auf einem Papier mit fester Breite bestimmt sind.

Fein, das hätten wir auch verstanden (hoffe ich). Dann geben wir in der nächsten Zeile an, wie die Variable SeitenNr ausgegeben wird, und zwar bis Spalte 11. Hups! Die Variable haben wir noch nicht definiert. Macht nichts, kommt nachher.

Wie Du siehst, ist die Angabe des rechten Endes eines Felds einfach. „Seite“ hört in Spalte 7 auf. Addiert man eins für einen Leerschritt und drei für die Länge von „SeitenNr“, kommen wir logischerweise auf 11.

Nun folgt noch die Definition der beiden festen Werte „`Kontenliste`“ und „`Datum:`“

Letztlich in diesem Ausgabeformat drucken wir das aktuelle Datum. Dies erreichen wir, in dem wir als auszugebendes Feld `UDATE` angeben. Dies ist ein reservierter Systemwert, numerisch mit 6 Stellen, der das Datum des aktuellen Jobs enthält. Für unsere ersten Zwecke reicht dies. Wir geben diesen Wert bis Spalte 77 aus. Beim prompten stellst Du fest, dass ich im Feld `Aufb.Schl.` den Wert `Y` eingetragen habe. Dies ist eine Angabe, wie das System das vorher angegebene Feld formatieren soll. Y stellt das Format „xx/xx/xx“ dar, hierbei handelt es sich zwar um das amerikanische Datumsformat, wir werden später aber sehen, wie man Werte noch ganz anders formatieren kann (besonders interessant bei monetären Werten).

Somit hätten wir das erste Druck-Satzformat erstellt. Machen wir uns gleich an das nächste, die Spaltenüberschriften.

Hier verwenden wir die oben angesprochene Methode, mehrere Formate unter dem gleichen Namen zu definieren, so dass sie mit einem Befehl ausgegeben werden können.

Die beiden Formate sind sehr einfach, da sie nur aus Konstanten bestehen.

Schau selbst:

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                        PRTKONTO
 FMT O  ODateiname+DF..N01N02N03ExceptnameB++A++Vv+Vn+.....................
0012.00 OQPRINT    E            SPALTUEBER     1
0013.00 O                                          7 'Konto-'
0014.00 O                                         28 'Kontenbeschreibung'
0014.01 O                                         54 'Anlagedatum'
0014.02 OQPRINT    E            SPALTUEBER     2
0014.03 O                                          7 'Nummer'
```

Die beiden Formate für die Spaltenüberschrift (auch hier die Zeilennummern ignorieren)

Du siehst, zwei Formate mit dem gleichen Namen, und jeweils nur ein paar Konstanten darin. In der Formatnamens-Zeile habe ich einmal als Zeilenvorschub nach der Ausgabe eine und dann zwei Zeilen eingetragen. Somit liegen die beiden Ausgabezeilen direkt untereinander, und nach der zweiten ausgegebenen Druckzeile erhalten wir eine Leerzeile.

Nun kommt es an die Details, die auch nicht wirklich aufwändig sind. Wichtig ist hier nur, dass Du hier als auszugebende Felder diejenigen angibst, die aus der Kontodatei kommen. Schau:

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                        PRTKONTO
 FMT O  ODateiname+DF..N01N02N03ExceptnameB++A++Vv+Vn+.....................
0015.00 OQPRINT    E            DETAILS        1
0016.00 O                       ktonum              7
0017.00 O                       ktotxt             40
0018.00 O                       ktodat             51
        ******************Datenende ***********************************************
```

unsere restlichen Ausgabezeilen

Auch hier gilt: Nach jeder Detailzeile gibt es einen Zeilenvorschum um eine Druckzeile. Du könntest auch zwei angeben, um so zwischen jeder Detailzeile einen Leerraum zu haben, falls es Dir gefällt...

Fein, nun haben wir spezifiziert, wie die Druckausgabe aussehen soll, jetzt kommen wir an die Befehle. Die jetzt folgenden Bestimmungen musst Du zwischen die F-Bestimmungen für die Dateien und die O-Bestimmungen für die Druckoptionen einfügen, sonst meckert der RPG-Compiler Dich wüst an.

Zunächst einmal brauchen wir noch eine Variable „SeiteNr“, dies erledigt uns folgende Zeile:

```
 FMT D  DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0002.01 DSeiteNr          S              3S 0
```

Deklaration einer einfachen Variablen

Damit haben wir eine dreistellige, numerische Variable, die wir mit der aktuellen Seitennummer füllen können. Einfach, übersichtlich, das Prompting und die Feldhilfe erklären Dir alles. Unter Name geben wir den Variablennamen ein. Als Deklarationsart (unter Spalte D) müssen wir linksbündig ein S eintragen, was dem Compiler mitteilt, dass es sich hier um ein eigenständiges Feld handelt[I]. Unter Länge geben wir rechtsbündig den Wert 3 an (siehe auch unsere Ausgabespezifikation). Als interne Datenart wähle ich ein S, was für einen normalen numerischen Wert ohne Besonderheiten steht. Und Dezimalstellen benötigen wir bei einer Seitenzahl nicht wirklich.

Nun kommen die wichtigen Befehle! ;-)

[I]Es gibt auch zusammengesetzte Felder. Diese tolle Angelegenheit erkläre ich nachher.

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                            PRTKONTO
 FMT CX CL0N01Factor1+++++++Opcode&ExtErweiterter-Faktor2+++++++++++++++++++++
0005.00 C                   EVAL      SeiteNr = 1
0006.00 C                   EXCEPT    SEITENKOPF
0007.00 C                   EXCEPT    SPALTUEBER
0008.00
0009.00 C                   READ      KONTOPF                                80
0010.00 C                   DOW       *IN80 = *OFF
0011.00 C                   EXCEPT    DETAILS
0012.00 C                   IF        *INOA = *ON
0013.00 C                   EVAL      SeiteNr = SeiteNr +1
0014.00 C                   EXCEPT    SEITENKOPF
0015.00 C                   EXCEPT    SPALTUEBER
0016.00 C                   ENDIF
0017.00
0018.00 C                   READ      KONTOPF                                80
0019.00 C                   ENDDO
0021.00 C
0022.00 C                   EVAL      *INLR = *ON
```

Alle nötigen Befehle zum Ausdrucken

Zunächst belegen wir die Variable SeiteNr mit dem Anfangswert. Nun wird mit dem EXCEPT-Befehl einmal das Format des Seitenkopfs und einmal die Formate der Spaltenüberschriften ausgegeben. Du siehst, ein EXCEPT reicht für beide definierten Formate „SPALTUEBER“.
Nun beginnen wir mit der Leseschleife. Falls Du Dich fragst, warum ich zwei READ-Befehle habe, hier die einfache Erklärung:

Nehmen wir an, Du programmierst (in pseudocode geschrieben) nur einen READ-Befehl an folgender Stelle:

```
SOLANGE *IN80 = *OFF
     LESE SATZ AUS DATEI
     DRUCKE DETAILDATEN
     ...
ENDE SOLANGE
```

Nehmen wir weiterhin an, die Datei ist leer, oder kann wegen eines sonstigen Fehlers nicht gelesen werden, dann würde Dein Programm aber ohne weitere Prüfungen trotzdem eine Detailzeile ausgaben. Das ist nicht wirklich sauber. Der erste READ-Befehl ausserhalb unserer DOW-Schleife stellt sicher, dass zumindest ein Satz lesbar ist.

Innerhalb der Schleife geben wir also zunächst den aktuell gelesenen Satz als Detaildaten aus. Dann kommt eine IF-Abfrage mit einer Dir nicht bekannten *IN-Bezugszahl. Nach jedem EXCEPT-Befehl wird vom System geprüft, ob das Seitenende der der Druckerdatei erreicht wurde (in QPRINT meist auf 66 Zeilen eingestellt). Wenn dies der Fall ist, sollte der Programmierer sich um den Seitenwechsel kümmern, also einen neuen Seitenkopf etc. ausgeben. Du kannst bei der Definition der Ausgabedatei eine Bezugszahl angeben, die beim Erreichen des Seitenendes (also nach der Ausgabe der letzten druckbaren Zeile) auf *ON gesetzt werden soll. Wenn Du das nicht tust, wählt der Compiler automatisch OA. Also prüfen wir hier diese Bezugszahl. Wenn das Seitenende erreicht wurde, machen wir nichts anderes, als den Zähler der Seitennummer um eins zu erhöhen, sowie einen neuen Seitenkopf und neue Spaltenüberschriften auszugeben. Wenn Du diese IF-Abfrage weglässt, druckt Dein Programm einfach auf Zeile eins in Seite zwei weiter, was dann wegen des fehlenden Seitenkopfs etwas unschön aussieht. Nach der Prüfung auf Seitenwechsel versuchen wir einfach, einen weiteren Satz zu lesen und schliessen die Schleife ab (die dann weiter läuft, wenn *IN80 = aus, also ein weiterer Satz vorhanden ist).

Siehst Du, so einfach geht das. Experimentiere doch mal damit! Weitere Druckereien machen wir später.

Sein oder Design

Eigentlich wollte ich nicht so viel über Datenbankdesign schreiben, aber auf freundliches Bitten einiger Leser werde ich doch einen kleinen Exkurs starten. Wer sich schon länger mit Datenbanken beschäftigt oder das Thema langweilig findet (Obacht! Fallstrick), kann diesen Teil gerne überspringen.

Nein, keine Sorge, es folgt jetzt keine mehrseitige Abhandlung über den Sinn und Unsinn diverser Möglichkeiten, wie man seine Daten irgendwo unterbringt.

Ich will nur ein paar Überlegungen anbringen, wie man beim Programmieren effektiv sein kann, und trotzdem mit Blick in die Zukunft arbeitet. Da gerade dieses Thema sehr vom persönlichen Geschmack abhängt, und fast jeder Programmierer individuelle Vorstellungen hat, rechne ich schon mit Protesten der Leser. Aber ich bin ja für jede Anregung offen, und schon einige sind in dieses Manual eingeflossen.

Grundüberlegungen

Viele Projekte fangen bei einer kleinen, einfachen Idee an, und wachsen schneller, als dem Programmierer lieb ist. Schnell werden Anforderungen gestellt, noch schneller müssen sie umgesetzt werden, und irgendwann verstrickt man sich in einer Stolperfalle, die man sich vielleicht selbst am Anfang gestellt hat.

Dann geht das Chaos los, schnell eine Erweiterung unterzubringen, nebenher die Einschränkungen des anfänglich einfachen Designs loszuwerden und das Ganze noch möglichst fehlerfrei an den Anwender zu bringen.

Auf der anderen Seite kann man ein Projekt mit Blick in die Zukunft beginnen, arbeitet nach einem detaillierten Konzept und versucht, modular und flexibel wie möglich zu bleiben. Dann wird der Aufwand unterschätzt, oder gar am Anfang sind die nötigen Vorarbeiten so hoch, daß das Projekt entweder einschläft oder irgendwann die Notbremse gezogen wird (vielleicht von außen).

Daher möchte ich an dieser Stelle mit Überlegungen anfangen, damit der Leser selbst den Weg für sein Projekt finden kann. Auf keinen Fall sollen diese Seiten als Richtlinie oder „einzig wahrer Weg“ verstanden werden. Denn erstens schreibe ich von meinen Erfahrungen, und zweitens maße ich mir nicht an, fehlerfrei zu arbeiten ;-)

Was machen wir hier eigentlich?

Zunächst sollten wir uns überlegen, was wir machen wollen. Warum wir Daten verarbeiten, wie groß das Projekt werden soll und kann, und somit letztlich, wie das alles über die Bühne gehen müsste. Hierbei fängt man oft nach dem Kohlschen Motto an, „wichtig ist, was hinten bei raus kommt“. Dies ist gewiss kein falscher Ansatz, allerdings konzentiert man sich oft zu schnell und zu sehr nur noch auf das Ergebnis.

Ein Marathonläufer, der nach 42,195km das Ziel überschreiten will, muss aber auch bedenken, wie er die Strecke zurücklegt, an welchen Stellen er Kräfte spart, und zu welchem Zeitpunkt die volle Energie zum Einsatz kommen kann. Läuft er am Anfang zu schnell los, hat er möglicherweise am Ziel keine Puste mehr. Läßt er sich am Anfang zu viel Zeit, kommt er zwar an, aber er wird im Schlussfeld liegen.

Also benötigt er nicht nur eine Strategie, wie er durchs Ziel läuft (Arme hoch reißen, Jubeln, vor Freude tanzen), sondern noch viel mehr, wie er die Strecke durchhält. Ebenso bedarf es einer ungeheuren Vorbereitung und natürlich auch der Kondition. (Ich selbst als relativ unsportlicher Mensch würde mir nie in den Sinn kommen lassen, einfach 42km zu laufen und nachher noch lebend anzukommen, wofür gibt es Autos? :-)

Über diese Strategie sollten wir nun nachdenken. Ich gehe hier nicht von einem konkreten Projekt aus, sondern versuche, diverse Szenarien und Möglichkeiten in das Denken einzubauen. Aber erst mal Schritt für Schritt...

Am Anfang war das Ziel

Natürlich sollten wir uns zunächst überlegen, was das Ziel unseres Projektes ist. Gehen wir einmal von einer einfacheren Angelegenheit aus: Einem Programm zur Erfassung von Buchungen. Fassen wir nicht gleich eine komplette Buchhaltung als Ersatz für SAP FI ins Auge, sondern denken an das kleine Unternehmen, das ein paar Buchungen eingeben will und nachher eine Auswertung darüber erstellt. Wenn wir vorausschauend denken, kann natürlich mehr daraus werden.

Gehen wir also davon aus, dass es ein Konzept für das zu erstellende Programm (oder die Programme) gibt, und dass in Gesprächen mit den späteren Anwendern das Ziel mehr oder weniger grob abgesteckt wurde.

Nun benötigen wir natürlich eine Definition der Datenbank, also des Herststücks der ganzen Angelegenheit. Hier geht es schon los, die ersten Fehler können gemacht werden.

Was wohin, und wie?

Ich möchte erst das Thema der technischen Grundlage ansprechen, bevor wir uns Gedanken darüber machen, wie wir die nötigen Daten unserer Mini-Buchhaltung unterbringen.

Zunächst ist natürlich auch die technische Basis von Bedeutung. Man kann keinen Rennwagen planen, ohne den passenden Motor dazu zu haben.

Da wir auf einer AS/400 arbeiten werden, richten sich unsere Überlegungen also auf deren Möglichkeiten.

Das kleinste Element einer Datenbank ist ein Feld. Ein Feld enthält einen einzelnen Wert, und mehrere Felder fasst man in einem so genannten Satz zusammen. Viele Sätze ergeben eine Datei, und viele Dateien einer Datenbank.

Als Beispiel für ein Feld sei „Konto“ genannt. Ein Konto ist für den Buchhalter ein Sammelplatz für einzelne finanzielle Vorgänge, die zu einem bestimmten Bereich oder einem organisatorischen Element passen, zum Beispiel das Bankkonto des Unternehmens. Dieses Konto hat eine Nummer. Okok, genau gesagt hat es bestimmt mehrere Nummern. Zum einen die Nummer, unter der das Konto bei der Bank geführt wird. In der Finanzbuchhaltung werden die einzelnen Bereiche oder Konten mit eigenen, natürlich eindeutigen Nummern geführt. Diese Nummern werden so organisiert, dass ähnliche Konten ähnliche Nummern haben, zum Beispiel, wenn das Unternehmen mehrere Bankkonten besitzt.

Daraus schließen wir, dass die Nummer, unter der wir (oder der Nutzer des Buchhaltungsprogramms) alle Konten in einem eigenen Satz Nummern führt, und dass diese vom Aufbau her gleich sind. Also können wir alle Konten vom gleichen Typus definieren.

Diese Kontonummer ist Teil einer Buchung. Eine Buchung ist ein Vorgang, der beschreibt, welcher Betrag von welchem Konto auf welches Konto fließt. All diese Daten fasst man zu einem Satz zusammen, genauso wie man in einer Adressverwaltung alle Daten einer Adresse zu einem Adress-Satz zufammenfasst.

Alle Buchungen kann man in einer Datei zusammenfassen, wenn man genügend Daten hat, jede Buchung eindeutig zu unterscheiden und alle Buchungen zu sortieren. Somit vermeidet man unnötige Redundanzen, also mehrfache Speicherung von Daten.

Gehen wir wieder zurück ins Detail. Jedes Feld, welches wir verwenden, muss definiert werden, und diese Definition ist ein sehr wichtiger Teil unseres Datenbankdesigns. Wenn wir hier nicht aufpassen, versauen wir uns das gesamte Projekt.

Jedes Feld einer Datei muss einen eigenen Namen haben, so wie ein Auto nur ein Kfz-Kennzeichen hat (Schweizer und Österreicher: Bitte keinen Widerspruch! :-). Nun kann es allerdings vorkommen, dass wir in mehreren Dateien Informationen über ein Konto ablegen müssen. Wenn wir nun in jeder Datei das Feld für die Kontonummer mit „Konto“ kennzeichnen, werden wir Probleme bekommen, wenn wir in einem Programm mehrere Dateien verwenden, die mit einer Kontonummer arbeiten. Also sollten auch alle Felder für die Kontonummer in allen Dateien eindeutige Namen haben.[1] Daher sollten wir uns genaue Gedanken über den Namen der Datei und den Namen des Feldes machen.

Auf der AS/400 kann unter RPG4 ein Dateiname sowie ein Feldname maximal 10 Zeichen lang sein. Dies mag für viele Leser etwas wenig erscheinen, hat aber seinen Grund in der Spaltenorientierung der Programmierung. Ausserdem verführt die Möglichkeit langer Namen dazu, „geschwätzig“ zu werden. Das kann zu viel Tipparbeit führen, zum Vorteil der Lesbarkeit, zum Nachteil der Programmierung. Aber geben wir uns mit dem zufrieden, was wir haben . Da also ein Feldname auf 10 Stellen begrenzt ist, wir aber dateiübergreifend eindeutige Namen für die Felder erreichen wollen, müssen wir eine Kombination aus Dateiname und Feldname verwenden.

Mein Vorschlag sei hier, dass man den Feldnamen aus einer Kombination von 4 Stellen der Datei und 6 Stellen für die Funktion des Feldes verwendet. Nehmen wir also an, wir haben folgende Dateien:

Dateiname	Funktion des Felds
BUCHUNGEN	Alle Buchungen, die eingegeben werden
KONTEN	Informationen über jedes verwendbare Konto
OPDATEN	Offene Posten zur Verfolgung von Zahlungen

Nehmen wir weiter an, dass wir innerhalb dieser Dateien mit folgenden Feldern arbeiten:

Feldname	Feldfuntkion
KONTONUMMER	Mit welchem Konto wird gearbeitet?
LAUFNUMMER	eine eindeutige, fortlaufende Identifikationsnummer
ANLDATUM	Das Anlagedatum der Information
BEMERKUNG	ein kurzer Text zum Objekt

[1] Es gibt Datenbanksysteme und Programmiersprachen, in denen ein Feld immer über die Kombination aus Dateiname und Feldname angesprochen wird. Dies hat Vorteile bei der Benennung, aber Nachteile bei der Flexibilität. Die AS/400 arbeitet mit der Variante, dass nur über Feldnamen referenziert wird.

Nun könnte man auf Grund dieser Informationen die Felder wie folgt benennen:

Feldname	Funktion
BUCHLAUFNR	Eindeutige Nummer jeder Buchung
BUCHKTONUM	Welches Konto wird bebucht?
BUCHANLDAT	Datum der Buchung
KONTKTONUM	Kontonummer in der Konto-Stammdatei
KONTBEMERK	Bemerkung zum Konto
OPLAUFNR	Eindeutige Nummer jedes Offenen Postens
OPKTONUM	Zu welchem Konto gehört der OP?
OPANLDAT	Wann wurde der OP erstellt?
OPBEMERK	Bemerkung zum OP

Und so fort. Zur Veranschaulichung habe ich den Dateiteil im Feldnamen unterstrichen, denn man muss ja nicht immer alle vier Stellen ausnutzen :-)

Wie Du siehst, habe ich nicht nur bei jedem Feld den Datei-Teil je Datei gleich gesetzt, sondern auch die Feldteile. Sprich: Wenn Du bei dieser Methode in irgendeiner Datei ein Feld xxxxANLDAT siehst, erkennst Du sofort, dass es sich hierbei um das Anlagedatum handelt.

Sicherlich, mit sechs Stellen kann man nicht immer wirklich sprechende Feldnamen erstellen, aber wenn man sich ein gewisses System zurecht legt, kommt man gut zurecht. Man denke nur an die Befehlsnamen der AS/400.

Sollte es zu knapp für Dein Vorhaben sein, kannst Du auch den Teil der Datei auf 3 Stellen verkürzen. Ganz harte arbeiteten früher und auch heute noch mit zwei Stellen für den Dateiteil und 4 Stellen für den Namensteil. Das ist historisch bedingt: In RPG II waren Feldnamen noch auf 6 Stellen begrenzt.

Gut, ich denke, das war einleuchtend. Wir verschwenden die drei oder vier Stellen, damit wir im Programm jederzeit wissen, mit welchem Feld wir arbeiten. Ausserdem kann man somit später einige nette Schweinereien anstellen. ;-) (Besonders, wenn man den Dateinamen-Teil stets gleich lang behält, also zum Beispiel immer die ersten vier Stellen für den Dateinamen-Teil verwendet).

Nun geht es also daran, uns ein kleines Datenbankmodell zusammen zu stricken. Hier müssen für alle Dateien die nötigen Felder definiert werden. Bevor Du in die Finger (oder auf die Tastatur) spuckst und die Definitionen in die DDS-Dateien reinhackst, bremse nochmal. Bei wenigen Dateien und wenigen Feldern ist sowas schnell gemacht, aber man soll ja immer an die Zukunft denken. Angenommen, Du hast innerhalb Deiner Dateien ungefähr 7mal ein Feld KTONUM untergebracht.

Überall hast Du dieses Feld als 6stellig Alpha definiert (warum, sei erst mal egal). Nun stellst Du fest, dass diese sechs Stellen nicht ausreichen, Du benötigst 8 Stellen, besser 10 (Festplattenplatz ist ja inzwischen günstig). Also musst Du nun an diesen 7 Stellen die Definition ändern. Dabei wirst Du mindestens eine Definition vergessen, und im unglücklichsten Fall fällt das noch nicht mal auf. Erst, wenn es zu spät ist und durch die fehlenden Stimmen Informationen verloren gingen. Das Stichwort zur Vermeidung des Übels ist „*Referenzierung*“.

Referenz bezeugen

Dies bedeutet, dass Du für jeden Feldtyp eine Definitionsvorlage erstellst, und in all Deinen Dateidefinitionen darauf zugreifst. In der Praxis ist dies ganz einfach: Du erstellst ein DDS, in dem alle Felder definiert werden, die in all Deinen Dateien vorkommen. Und in den Beschreibungen greifst Du auf die Vorlage zurück.

Wie geht das nun? Ganz einfach. Du erstellst ein DDS für eine physikalische Datei namens #REFERENZ (oder wie Du sie auch immer nennen magst). In diese Datei trägst Du für jedes Feld, das Du in Deinem Projekt verwendest, ein sogenanntes Referenzfeld ein. An Stelle der ersten Buchstaben für den Dateiteil kannst Du REF oder etwas ähnlich einleuchtendes Verwenden. Manche nehmen XX oder ## etc. Ausserdem solltest Du wirklich jedes einzelne Feld dort verewigen. Wenn Du 4 verschiedene Datumsfelder in Deinem Projekt hast (Anlagedatum, Geburtsdatum, Rechnungsdatum etc.) solltest Du hierfür jeweils einen eigenen Eintrag machen. Erstens, um schon in der Referenzdatei zu sehen, welche Felder es insgesamt gibt, zweitens, um mögliche Formatierungen vorzusehen, drittens, um jedem Feld einen Text mit der Feldverwendung zu übergeben. Diese Informationen werden beim Referenzieren natürlich vererbt. Und viertens kann man sich später (wenn man in größere Projekte eintaucht), sich damit zum Beispiel die Hilfe vereinfachen. So, genug der Gründe, Du wirst sie später genau erkennen.

Eine Referenzdatei könnte vielleicht so aussehen:

```
 Spalten . . . :     1  71         Editieren                   MEINPROJEKT/QDDSSRC
 SEU==> ____________________________________________________________  #REFERENZ
 FMT A* .....A...........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
0001.00      A* Meine Referenzdatei
0002.00      A* -------------------
0003.00      A            R ##REFREC
0004.00      A              ##KTONUM       10A         TEXT('Kontonummer')
0005.00      A                                         COLHDG('KtoNum')
0006.00      A              ##KTOTXT       30A         TEXT('Kontobezeichnung')
0007.00      A                                         COLHDG('Kontobezeichnung')
0008.00      A              ##ANLDAT        8S 0       TEXT('Anlage-Datum')
0009.00      A                                         COLHDG('AnlDat')
0010.00      A
```

Beispiel für eine Referenzdatei

Und so weiter... Ein paar Hinweise: Zu jedem Referenzfeld gebe ich hier zwei Funktionen an. Erstens `TEXT()`, und zweitens `COLHDG()`. Beide Funktionen fügen dem Feld eine Beschreibung hinzu. Der `TEXT` ist die Beschreibung, die im System bei Auflistungen von Feldern verwendet wird (z.B. wenn Du im SQL mal die Taste F4 verwendest). `COLHDG` steht für „column heading“, also Spaltenüberschrift. Dies ist der Text, der z.B. in der Ausgabe von SQL über jeder Spalte steht. `TEXT` kann natürlich länger sein, da Feldauflistungen immer zeilenweise erscheinen. Die Spaltenüberschrift sollte logischerweise nicht breiter als der Feldinhalt sein, sonst verschenkt man viel Platz. Plakativ wirst Du das nachher sehen, wenn wir mit SQL arbeiten.

Du kannst aber auch die Spaltenüberschrift mehrzeilig gestalten, dies würde im DDS so aussehen:

```
0008.00       A            ##ANLDAT       8S 0       TEXT('Anlage-Datum')
0009.00       A                                      COLHDG('Anlage-' 'Datum')
```

Eine mehrzeilige Spaltenüberschrift mit COLHDG()

Also einfach in der Funktion zwei Texte jeweils in Hochkomma und durch ein Leerzeichen getrennt angeben. Allerdings ist somit im SQL die Feldüberschrift zweizeilig (klar doch, oder?).
Beachte: Wenn Du ein Feld, das nur 4 Stellen breit ist, mit einer Spaltenüberschrift von 15 Zeichen Breite versiehst, wird in einer SQL-Ausgabe das Feld auch 15stellig angezeigt, Du verschwendest Platz auf dem Bildschirm.

Nächste Frage: Warum definiere ich das Anlage-Datum als 8stellige Dezimalzahl? Nun, es hat etwas mit Faulheit zu tun. Wenn man das Datum so verwendet, kann man recht einfach damit arbeiten und Rechnen. Angenommen, das Feld BUCHDAT ist gleich 20020520 (20. Mai 2002). Somit kann man schnell einen Monat draufrechnen, in dem man sagt `EVAL BUCHDAT=BUCHDAT+100.` Oder das Datum um ein Jahr verringern, in dem man den Wert 10000 abzieht... Aber keine Sorge, die Ausgabe auf dem Bildschirm kann man auch schön formatieren. Natürlich kann man so zwei Daten schnell vergleichen, oder eine Liste danach sortieren. Nun haben wir also ein paar Referenzfelder, wie verwenden wir sie nun? Ausserdem ist die Handhabung von Datumsfeldern etwas aufwändiger, daher kommt dies später im Text.

Wo ist mein Referent?

Erzeugen wir nun also eine Datenbankdatei, deren Definition sich auf unsere Referenzdatei bezieht. Dies sieht dann so aus:

```
 Spalten . . . :     1  71         Editieren                  MEINPROJEKT/QDDSSRC
 SEU==> ____________________________________________________________  BUCHPF
 FMT A* .....A............A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
0001.00      A* Beispiel für Buchungsdatei
0002.00      A* -------------------------
0002.01      A                                    REF(#REFERENZ)
0003.00      A           R BUCHREC
0004.00      A             BUCHKTONUMR            REFFLD(##KTONUM)
0006.00      A             BUCHKTOTXTR            REFFLD(##KTOTXT)
0008.00      A             BUCHANLDATR            REFFLD(##ANLDAT)
0010.00      A
```

Wir beziehen uns auf unsere Referenzdatei

Man beachte: Zuerst wird vor der Definition des Records (Kennzeichen `R`) eine Zeile mit der Funktion `REF()` eingebaut. Diese Funktion sagt, aus welcher Datei die Referenzinformationen stammen. Diese sollte idealerweise in einer Bibliothek in der Bibliotheksliste existieren. Nun folgen die Felddefinitionen. Hierbei wird direkt nach dem Feldnamen in die R (Referenz)-Spalte ein `R` eingetragen. Auf dem Bildschirm ist kein Leerzeichen zwischen Feldname und dem R, dafür haben wir ja die Spaltenorientierung. Und im Funktionsbereich sagen wir, von welchem Feld (in unserer REF-Datei) die Datendefinition stammt. Da sich OS/400 die Informationen aus der Referenzdatei holt, benötigen wir natürlich keine weitere Angaben über das Feld. Ist doch ganz einfach, oder?

Vorsicht! Bevor Du nun im Eifer des Gefechts Deine Datenbankdatei mit Auswahl 14 (oder CRTPF) erstellst, muss zunächst die Referenzdatei erstellt werden. Sie wird zwar nie Daten enthalten, aber muss als physikalische Datei existieren, nicht nur als Definition im DDS. Ansonsten wirft Dir das System ein paar harrsche Fehler um die Ohren.

Wenn Du nun – aus welchem Grund auch immer – das Feld KTOTXT ändern musst (vielleicht, weil 30 Stellen irgendwann zu wenig sind), ist es nur nötig, die Definition der Referenzdatei zu editieren und die Referenzdatei mit `CHGPF` zu ändern. Danach änderst Du mit `CHGPF` alle Datenbankdateien und erstellst (man weiß ja nie) alle logischen Dateien neu. Idealerweise hat man für solche Fälle ein CL-Programm (wieder etwas, wozu wir später noch kommen werden :) Besonders Sinn macht dies natürlich, wenn Du ein Feld in verschiedenen Dateien verwendest. Beispiel: Du verwaltest in vielen Dateien ein Feld über die Belegnummer, die Du mal 6stellig definiert hast. Je nach Anwendung hast Du das Problem für Auftragsnummern, Rechnungsnummern etc. Nun reichen die sechs Ziffern nicht mehr, da der Anwender immer mehr Aufträge im Jahr produziert. Nun willst Du das Feld auf 8 Stellen erweitern. Das würde bei ca. 100 Dateien schon in Arbeit ausarten. Einfacher ist es doch wirklich, ein Feld für die generelle Definition einer Belegnummer zu haben, und nur diese zu ändern. Dann (am besten mit einem Skript) alle Dateien anpassen und Ruhe ist :-)

Interaktive Programme Teil 1

Verlassen wir mal wieder etwas die Theorie und kommen wir zu einem interessanteren Thema: *Interaktive Programme*.
Vorhin haben wir ein Batchprogramm geschrieben. Dies nennt man *Batch*, weil man es startet und es nicht mit dem Benutzer interagiert, das heißt, es werden keine Ausgaben gemacht und der Nutzer kann keine Eingaben tätigen. Das Programm läuft (sofern es fehlerfrei ist), einfach durch. Es erledigt seine Arbeit und wird beendet. Weiterhin kann man ein Batchprogramm „im Hintergrund“ in einem eigenen Systembereich (auch Subsystem genannt) laufen lassen, während man sich anderen Themen zuwendet. Somit kann das Batchprogramm laufen und die Anwender in Ruhe arbeiten. Das Batchprogramm kann man bei Bedarf anhalten und später weiter laufen lassen. Deutsch nennt man dies auch Stapelprogramme.

Ein interaktives Programm ist meist schon aufwändiger. Man benutzt es zum Pflegen von Daten, zur Eingabe oder zur Abfrage von Daten, die später einem Batchprogramm sagen, was es zu tun hat. Für ein interaktives Programm muss man schon etwas mehr tun als nur ein paar Zeilen in RPG zu schreiben.

Durch die konsequente Trennung von Programm, Daten und Hardware auf der AS/400 besteht ein interaktives Programm immer aus mehreren Teilen:

- das Programm selbst,
- einer Definition über den Aufbau und das Verhalten des Bildschirms und der Druckausgaben sowie
- den Dateien, mit denen das Programm arbeiten soll.

Der erste Punkt ist klar, der dritte auch. Sprechen wir kurz den zweiten Punkt an:

Bildschirmdefinitionen

Oder auch: Was wird wo, wie, wann und womit aus- und eingegeben. Wer die Programmierung unter DOS oder Unix kennt, wird erst mal verwundert sein. Wo ist das Problem? Ich programmiere (in BASIC) einfach:

```
PRINT 'WIE HEISST DU? :'
INPUT NAME$
```

Und schon habe ich ein interaktives Programm (in anderen Programmiersprachen analog).

Klar, das sieht einfach aus, und ist auch einfach. Aber schon wer unter Microsoft Windows™ in einem Fenster etwas ausgeben will, muss mehr tun.

Auf einer AS/400 ist dies (natürlich) wieder ganz anders.

OS/400 verfolgt ein etwas anderes Konzept. Obiges BASIC-Beispiel ist Zeilen-orientiert. Du gibst eine Zeile aus, und liest in der nächsten Zeile etwas ein. OS/400 arbeitet generell Bildschirm-orientiert, das bedeutet, Du musst immer einen gesamten Bildschirm (meist 80x24 Zeichen) definieren, mit Ausgabefeldern, Eingabefeldern, Datenfeldern, Positionen und Eingabeoptionen. Dann sagt Dein Programm dem System, es soll den Bildschirm ausgeben, und die Eingaben des Benutzers abholen. Wer ganz früher unter DOS mit Pascal eine größere Eingabemaske programmiert hat, weiß, wie schnell man graue Haare dabei bekommen kann. Bei der AS/400 kümmert sich das Terminal um die vom Programm übergebene Bildschirmdefinition und sendet danach nur noch die Benutzereingaben zurück. Hört sich sehr kompliziert und merkwürdig an, aber habe ich nicht gewarnt, dass man so einiges vergessen sollte, was man vom PC gewohnt ist? ;-)

Wie sieht's aus?

Wie macht man das nun mit einer Bildschirmdefinition? Nun, eigentlich banal... Ein Bildschirm wird als Datei definiert, ähnlich wie eine Datendatei. Dein Programm arbeitet fast genauso mit dieser Bildschirmdatei wie mit einer Datenbank. Man schreibt Sätze (mit denen das Aussehen des Bildschirms beschrieben wird) in diese Bildschirmdatei rein, und liest die Benutzereingaben aus einem Satz (genauer Satzformat) aus. Schon fertig. Okok, das mit dem „banal“ war ein Spaß. Gehen wir mal ins Detail.

Erstellen wir uns eine ganz einfache Datendatei und ein winziges Pflegeprogramm dafür.

Zunächst erstellen wir uns eine Datendatei mit Informationen über die Konten, mit denen unsere Mini-Buchhaltung arbeiten wird. Starte nun den PDM, gehe in Deine Bibliothek und dort in die Datei QDDSSRC, in der Du die Definitionen für Dateien abgelegt hast. Dort erstellst Du mit der Taste F6 eine neue Teildatei (Name: KONTOPF, Typ PF). Der Aufbau dieser Datei soll wie folgt aussehen:

```
 Spalten . . . :   1  71          Editieren                    LIBMANUAL/QDDSSRC
 SEU==> _____________________________________________________________   KONTOPF
 FMT PF .....A..........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
0001.00      A                                     UNIQUE
0002.00      A           R FMTKTO
0003.00      A             KTONUM          6P 0       TEXT('Kontonummer')
0004.00      A             KTOTXT         30A         TEXT('Kontobeschreibung')
0005.00      A             KTODAT          8S 0       TEXT('Anlagedatum')
0006.00      A           K KTONUM
       ******************Datenende ***************************************


F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Unsere kleine Kontodatei

Bei der Definition von KONTOPF habe ich mal auf Referenzierung verzichtet, es ist nur ein erstes Beispiel. Natürlich kannst Du auch auf Deine Referenzdatei zugreifen, beachte aber bitte die Längen der Felder.

Erstelle die Datei bitte mit CRTPF oder der Auswahl 14. Die nötigen Parameter solltest Du inzwischen angeben können.

Wenn die Datei erfolgreich erstellt ist, machen wir uns gleich an die Bildschirmdatei. Eine Bildschirmdatei erstellt man wie eine Datendatei.

Erstelle also nun die nächste Teildatei, diesmal mit dem Namen KONTOBS und Typ DSPF). DSPF steht hierbei für „Display File“, also Bildschirmdatei, logisch.

Hier siehst Du die Zeilen für eine ganz einfache (!) Bildschirmdatei.

```
 Spalten . . . :   1  71          Editieren                    LIBMANUAL/QDDSSRC
 SEU==> ___________________________________________________________    KONTOBS
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0001.00      A                                     REF(KONTOPF)
0002.00      A          R DSPKTO1
0003.00      A                                   1 35'Verwalten Kontodaten'
0004.00      A                                   5  5'Kontonummer:'
0005.00      A            B1KTONUM  R        I  5 20REFFLD(KTONUM)
0006.00      A                                   7  5'Kontotext  :'
0007.00      A            B1KTOTXT  R        I  7 20REFFLD(KTOTXT)
0008.00      A                                   9  5'Anlagedatum:'
0009.00      A            B1KTODAT  R        I  9 20REFFLD(KTODAT)
       ******************Datenende ***************************************


F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Unsere erste Bildschirmdatei

Wie Du vielleicht erst auf den zweiten Blick feststellst, stehen in der FMT-Zeile mehr Spalten als bei einer einfachen Datendatei zur Verfügung. Diese werden wir nun vorstellen:

Die erste Spalte A dient wieder der Definition des Zeilentyps, dieser ist wie bei Datendateien immer A, also ganz einfach. Die folgenden Felder A, N01, N02 und N03 ignorieren wir zunächst. Sie dienen der Konditionierung, also der Bestimmung unter besonderen Voraussetzungen. Das kann ein hochkompliziertes Thema werden, also wenden wir uns dem später zu.

Die nächste Spalte A dient wieder der eventuellen Angabe eines speziellen Definitions-Typs. Gehen wir mal die Zeilen durch.

In Zeile 0001.00 definiere ich - wie bei einer Datendatei - eine Quelle für eine Referenz. Hier hole ich mir Felddefinitionen aus der zugehörigen physischen Datei. Natürlich solltest Du später bei „sauberer“ Programmierung dafür eine Referenzdatei verwenden! Hier mache ich das nur aus Faulheit und der Anschaulichkeit halber.

In der Zeile 0002.00 wird ein Satzformat DSPKTO1 erstellt, daher auch das R in der Namensart-Spalte. Unter diesem Namen werden alle hier angegebenen Felder verwaltet, das bedeutet, man spricht die Bildschirmdatei über diesen Satzformat-Namen an.

In Zeile 0003.00 kommen gleich drei neue Spalten zum Einsatz. Neben der Angabe eines Feldnamens und dem R für die Referenzierung steht in Spalte F (Feldart) ein I. Dieses I gibt an, daß es sich um ein Eingabefeld (Input) handelt. Weitere Feldarten folgen später. Die Spalte ZEI dient der Angabe der Zeile, in der das hier definierte Feld stehen soll, POS gibt entsprechend die Position oder Spalte in dieser Zeile an. Wir definieren hier aber kein Feld im eigentlichen Sinne, sondern die Ausgabe des Texts „Verwalten Kontodaten“ in Zeile 1, Spalte 35. Dies ist unsere Überschrift für den Bildschirm. Da wir im Funktionsbereich einen festen Text in einfachen Hochkommata angeben, benötigen wir keinen Feldnamen in der Spalte NAME.

Das gleiche machen wir in Zeile 0004.00, dort definieren wir einen festen Text „Kontonummer:“, der vor dem entsprechenden Eingabefeld stehen soll.

Zeile 0005.00 beschreibt ein Eingabefeld. In der Spalte Name verwende ich einen eigenen Namen B1KTONUM für das Feld, nicht den Namen aus der physischen Datei, somit erstelle ich eine Kopie des Originalfelds.[I] Aus der physischen Datei hole ich mir mit dem R in der Spalte Ref die Definition, im Funktionsfeld steht die Angabe für das Originalfeld. Die Angabe von Zeile und Spalte für die Position dürfte klar sein, rechts von unserem festen Text „Kontonummer:“. Einleuchtend, oder?

Alle weiteren Zeilen sind somit wohl selbst erklärend, es ist wirklich nicht schwer, einen einfachen Bildschirm zu erstellen.

Nun brauchen wir zunächst einmal ein ganz einfaches Programm, das uns den definierten Bildschirm anzeigt. Dieses werden wir dann Schritt für Schritt um die benötigten Funktionen erweitern.

Ein Bild – ein Programm

Beende also bitte die Eingabe, speichere die Definition ab und erstelle die Bildschirmdatei mit Auswahl 14. Hierbei sollten keine Fehler auftreten. Prüfe dies anhand der Meldung des Compilers nach dem Umwandeln und der Begutachtung des Spoolfiles.[II]

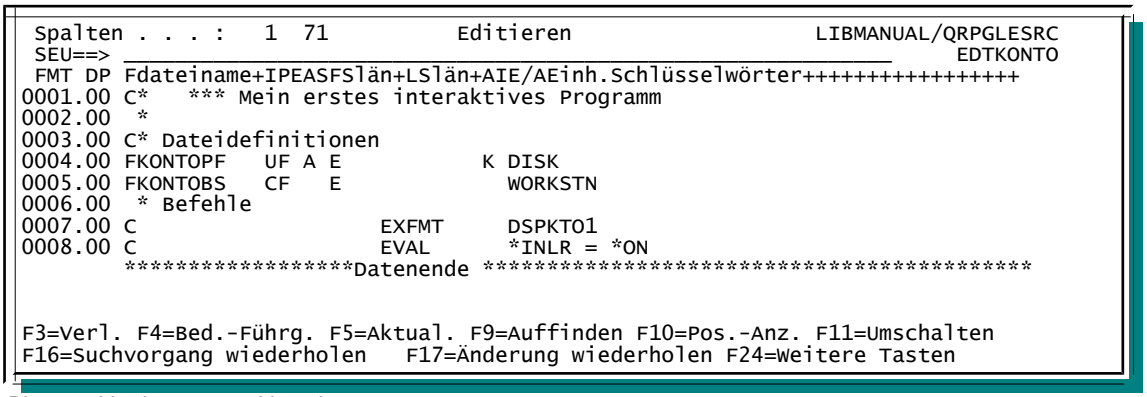
Nun wechsle bitte in die Datei QRPGLESRC für unsere RPG-Quellen. (erst F12, dann Auswahl 12 vor dem Dateinamen).

Hier erstellen wir eine neue Teildatei (Taste F6, wie bekannt) mit dem Namen EDTKONTO und dem Typ RPGLE. Als Text kannst Du „Mein erstes interaktives Programm“ eingeben.

Wir erstellen uns nun den Grundstock für unser Verwaltungsprogramm. Dieses soll es uns später ermöglichen, die zur Verfügung stehenden Konten inklusive aller Felder zu verwalten, und natürlich die Eingaben zu prüfen. Der Benutzer sollte durch die bereits angelegten Kontendaten blättern sowie vorhandene Datensätze löschen können.

Bitte tippe das Programm wie folgt ab (nutze die Promptingfunktion, nachdem Du in der ersten Spalte die Zeilenart angegeben hast). Wenn Du bereits fingerfertig genug bist, kannst Du natürlich auch direkt eintippen, aber trotz des zuschaltbaren Fadenkreuzes (Crosshair cursor) ist es manchmal etwas mühsam, die richtige Spalte zu finden.

[I] Es ist immer sinnvoll, in einem Bildschirm mit Kopien der verwendeten Felder zu arbeiten. Man muss zwar vor dem Anlegen eines Satzes die Feldinhalte in die Felder der Datendatei kopieren, aber es ist flexibler und sicherer, als direkt die Datenfelder zu nutzen.
[II] Ein kleiner Absatz über das Lesen eines Spoolfiles mit Fehlermeldungen findest Du im Anhang. Aber das erwähnte ich bereits.



Die erste Version unseres Verwaltungsprogramms

So, bevor wir das Programm umwandeln und starten, ein paar Hinweise zu den Befehlen. Da Du vorher schon ein RPG-Programm eingegeben und gestartet hast, werde ich nicht mehr alles ganz detailliert beschreiben.

Die ersten drei Zeilen stellen nur einen Kommentar für uns dar, dies wird durch einen Stern `*` in der zweiten Spalte (direkt rechts von der Spalte mit der Spezifikationsangabe) deutlich gemacht. Bei einer Kommentarzeile muss man den Spezifikationstyp nicht angeben, siehe Zeile `0003.00`.

In Zeile `0004.00` definieren wir unsere Datendatei (die natürlich schon existieren sollte) namens `KONTOPF`. Diese Datei ist für Lesen und Schreiben (engl. Update) durch das `U` gekennzeichnet. Das `F` bedeutet wieder „full processing“, also wahlfreier Zugriff. Mit dem `E` bestimmen wir wieder eine extern (also nicht im Programm) definierte Datei. Das `K` bedeutet „keyed access“, also Zugriff über einen Schlüssel. Und `DISK` gibt wie bekannt an, wo die Datei sich befindet.

(Wer immer noch nicht die `F4`-Taste lieben gelernt hat, sollte diese als Eingabehilfe verwenden. Nirgends kann man so gemütlich die einzelnen Felder auswählen und in jedem Feld mit `F1` etwas Lesematerial ansehen :-)

So, das ist ja eigentlich für Dich nichts neues, eine solche Dateibestimmung hatten wir schon in unserem ersten Batch-Programm. Die nächste Zeile 0005.00 ist da schon interessanter.

Die Angabe des Dateinamens neben der `F`-Bestimmung ist verständlich, diese Bildschirmdatei haben wir ja eben erstellt. Dann folgt als Dateiart ein `C`. Dies bedeutet „kombinierte Ein-/Ausgabedatei“ (combined). Dies ist ein minimaler Unterschied zu einer U-(Update)-Datei, da man in eine Bildschirmdatei nicht so schreibt wie in eine Datendatei, z.B. wird üblicherweise nicht direkt ein Datensatz adressiert, da auf dem Bildschirm nur ein „Datensatz“ sichtbar ist (Ausnahmen bestätigen die Regel und werden später besprochen). Die Dateiverwendungsart ist wieder `F`, da wir selbst bestimmen, wann was mit der Datei angestellt wird. Ein `E` für „extern beschrieben“ ist ebenso selbstverständlich. Und als Neuerung wird als E/A-Einheit nun `WORKSTN` angegeben, als Kürzel für „Workstation“, also Bildschirm. Ist doch gar nicht soo schwer, oder

Nach einem weiteren Kommentar kommt ein neuer Befehl.

Das Kürzel EXFMT steht für „execute format“. Danach folgt der Name des Satzformates, den wir in der Bildschirmdatei definiert haben. „Execute format“ bedeutet „Format ausführen“. Damit ist gemeint, daß man das Bildschirmformat auf den Bildschirm bringt und dem System sagt, es soll warten, bis der Benutzer eine Funktionstaste oder Datenfreigabe drückt. Danach liefert das Betriebssystem die Eingaben des Benutzers in die im Format vorhandenen Felder zurück an unser Programm.

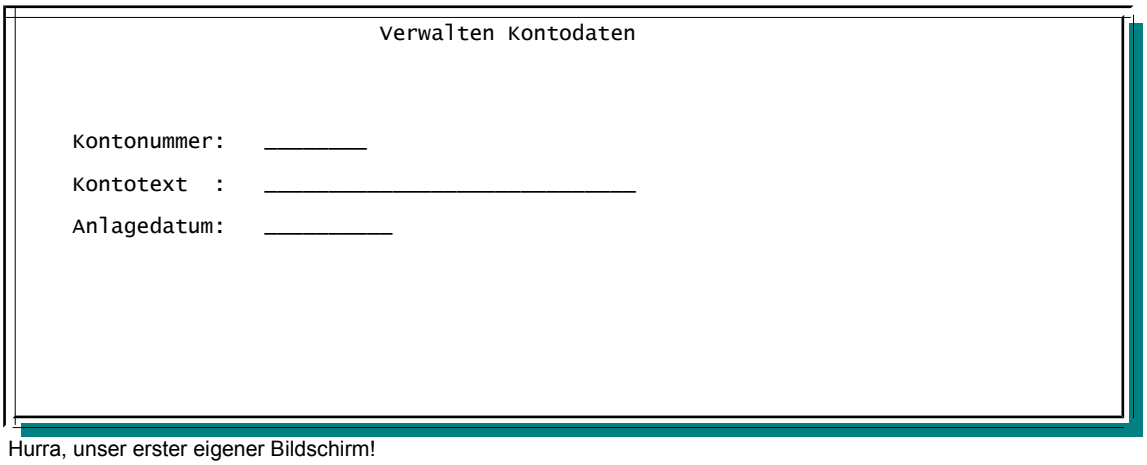
Der letzte Befehl sagt dem Programm wieder, dass es sich freundlich beenden soll.

Nun, was macht unser Programm? Ganz einfach, nicht philosophieren, sondern probieren! Also mit Auswahl 14 umwandeln und in der Befehlszeile mit

```
call edtkonto
```

starten und sehen, was passiert.

Wenn Du alles richtig eingetippt hast, sollte nach erfolgreicher Umwandlung und Programmstart Dein Bildschirm so aussehen:



Hurra, unser erster eigener Bildschirm!

So, das sieht doch schon mal gar nicht so schlecht aus. Wir haben einen eigenen Bildschirm definiert und auf dem Bildschirm ausgegeben. Zwar sieht das ganze noch etwas blass, grün und leer aus, aber das kann man ändern. Zunächst wollen wir uns mal ansehen, was der Bildschirm so alles kann. Der Cursor blinkt im Feld rechts vom Text „KONTONUMMER:“. Weil es das erste Eingabefeld auf dem Bildschirm ist, war das Betriebssystem so freundlich und hat das für uns gemacht. Du kannst mit dem Cursor auf dem Bildschirm herumfahren, mit Tab die Felder wechseln und natürlich Daten eingeben. Tippe bitte mal in das Feld „KONTONUMMER“ den Wert „TEST“ ein...

Geht nicht? Dein Terminalprogramm zeigt unten ein X und akzeptiert keine Tasten mehr? Unten links steht vielleicht eine vierstellige Nummer? Komisch... obwohl, eigentlich ist es das nicht.

Eingabeprüfung – Was geht?

Schau Dir bitte noch mal die Definition Deiner Bildschirmdatei an. Das Eingabefeld `B1KTONUM` ist wie definiert? Achja, es ist eine Kopie von `KTONUM` in der Datei `KONTOPF`. Und wie ist *dieses* Feld definiert? Wir erinnern uns: Dort haben wir `6P 0` eingegeben. Das bedeutet `6` Stellen, gepackt dezimal mit `0` Nachkommastellen. Also ein spezielles numerisches Format. Wie speziell, lassen wir mal gerade beiseite, wichtig ist, dass es *numerisch* definiert ist. Daraus schließt das System beim Darstellen des Bildschirms automatisch, dass Buchstaben in diesem Feld nicht wirklich Sinn machen und verweigert die Akzeptanz unserer Eingabe. Gar nicht so doof, was? Das System sperrt die Tastatur für weitere Eingaben, damit man nicht beim wilden Blindtippen mehrere Felder mit falschem Inhalt belegt.

Damit Du weiter tippen kannst, musst Du eine etwas merkwürdig genannte Taste namens „Grundstellung“ drücken. Das ist nichts schweinisches, sondern bedeutet nur ein Rücksetzen des Terminals. Englisch wird diese Tastemit „Reset“ - meiner Meinung nach - auch nicht viel besser beschriftet. Auf jeden Fall ist damit in fast jedem 5250-Terminal und am PC die linke `STRG`-Taste gemeint. Probiere es aus.

Nun kannst Du weiter tippen. Schreibe also nun in das Feld „Kontonummer“ den Inhalt `123` rein und drücke die Eingabetaste. Nein, nicht Datenfreigabe, sondern Deine Returntaste. Wie Du bemerkst, wird die eingegebene Zahl rechtsbündig im Feld ausgerichtet. Somit wird gleich deutlich, dass das Feld wirklich numerisch ist. Dieses Verhalten kann man natürlich anpassen, wenn man mag.

Der Cursor steht nun im Feld „Kontotext“, hier tippen wir einmal „`Ich bin ein kleines Konto`“ ein. Hoppla... alles Großbuchstaben. Das war nun aber nicht so gewollt. Keine Sorge. Von Hause aus werden Textfelder auf dem Bildschirm so behandelt, dass man nur Großbuchstaben schreibt, egal ob man die Shift-Taste verwendet oder nicht. Liegt wohl daran, dass viel früher Kleinbuchstaben an einem Terminal Luxus waren, damals, als es noch keine PCs gab :-) Natürlich kann man auch dies ändern.

In das letzte Feld gehört unser Anlagedatum. Da dieses Feld in der physikalischen Datei als `8S 0` definiert ist (also ein numerisches Feld mit 8 Stellen), können wir hier auch nur Ziffern eingeben. Damit man das Datum mit Punkten eingeben kann, wird hier auch später ein Zusatz zu diesem Feld definiert. (Und um die Profis z u beruhigen: Später definieren wir natürlich ein richtiges Datumsfeld).

Nun haben wir etwas in unseren Bildschirm eingetippt. Wie kommen wir aber wieder raus? Unten stehen keine F-Tasten (wir haben auch nirgendwo angegeben, dass dort etwas stehen soll), wenn wir `F3` drücken, wird uns dies angemeckert. Von Hause aus kann man einen Bildschirm nur mit `DF` verlassen, also mit der Option, dass sich das System um die eingegebenen Daten kümmern soll.

Dies reicht uns zunächst, wir wollen den Bildschirm und das Programm mit etwas mehr Leben füllen, ist ja sonst zu langweilig hier :)

Ein bunter Hund

Zunächst wollen wir etwas Farbe ins Spiel bringen, alles nur Grün ist ja echt nicht sehr spannend. Wechsele also bitte wieder in die Quelle für die Bildschirmdatei. Füge nun unter der Zeile `0003.00` („Verwalten Kontodaten“) eine neue Zeile ein. Vergiss nicht das `A` in Spalte 6 unter den anderen, und gehe dann nach rechts in die Funktionsspalte. Unter das erste Hochkomma schreibst Du nun `COLOR(WHT)` und verlässt die Datei wieder. Da Du nur in der Funktionsspalte etwas angegeben hast, erkennt der Compiler automatisch, dass diese Funktion zu der darüber liegenden Definition gehört. Je nach Funktion und Definition kannst Du mehrere Funktionen (jeweils in einer eigenen Zeile) angeben.

Die Funktion `COLOR` sagt dem System, dass das aktuelle Feld (in unserem Fall die Zeile mit der Bildschirmüberschrift) nicht in Grün, sondern einer anderen Farbe dargestellt werden soll. `WHT` bedeutet hier „weiß“. Folgende Farben kannst Du auswählen:

`WHT = weiß; GRN = grün; YLW = gelb; RED = rot; TRQ = Türkis; BLU = blau; PNK = lila`

Nun definiere auf die gleiche Weise bitte alle Feldbeschriftungen (die Definitionen ab Spalte 5) in Farbe Blau, in dem Du `COLOR(BLU)` in einer Zeile darunter schreibst. Dies wären 5, 7 und 9. Hier kannst Du gleich die Kopierfunktion von SEU nutzen. Füge also unter `0005.00` die Zeile

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0005.01      A                                      COLOR(BLU)
```

(Wie bekannt: Bitte selbst auf die Spalten achten!)

hinzu. Nun gehst Du an den Anfang dieser Zeile, schreibst über die Zeilennummer ein `C`, sowie über die Zeilennummer 0007.00 ein `A`. Nach `DF` wird die Zeile erwartungsgemäß kopiert. Dann kopierst Du diese Zeile noch hinter die Beschreibung des dritten Eingabefeldes. Ach ja: Je nach System- und Compilerversion verlangt SEU, dass die Funktionsnamen und Parameter in Großbuchstaben geschrieben werden.

Die Eingabefelder lassen wir grün, schreiben also keine weitere Funktion dazu. Nun den SEU wieder mit `F3` verlassen und die Bildschirmdatei mit Auswahl `14` neu erstellen. Vermutlich fragt Dich das System, ob Du das bestehende Objekt löschen willst. Gehen wir davon aus und geben `J` ein.

Solltest Du diese Abfrage nicht haben wollen, wartest Du bitte, bis die Datei erstellt wurde. Drücke nun die Taste `F18`. Es erscheint das Bild „`Standardwerte ändern`“. Hier kannst Du diverse Grundeinstellungen für den SEU anpassen. Uns interessiert hier die zweite Zeile „`Objekt ersetzen`“. Wenn hier ein `J` steht, wirst Du nicht gefragt, ob beim Umwandeln ein vorhandenes Objekt ersetzt werden soll.

Wenn nun die Bildschirmdatei erfolgreich erstellt wurde, starte bitte Dein Programm wieder mit

```
CALL EDTKONTO
```

Das Ergebnis sieht doch schon etwas augenfreundlicher aus, oder?

Bildschirmfunktionen für Faule

Spielen wir noch etwas mit den Funktionen der Bildschirmdatei herum, um einen Eindruck zu bekommen, was man so anstellen kann, ohne programmieren zu müssen.

Bitte das Programm mit DF verlassen und mit dem SEU die Quelle für KONTOBS bearbeiten.

Füge nun unter die Zeile mit dem Feld B1KTOTXT folgende Zeile ein:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0010.01      A                                        CHECK(LC)
```

(Dass die Funktion genau unter die Angabe des Referenzfeldes gehört, sollte klar sein. Jede Funktion gehört hier hin). Die Funktion CHECK() wird für diverse Prüfungen verwendet, man kann dem System sagen, welche Angaben für dieses Feld gültig sind, bzw. welche Buchstaben die Gültigkeitsprüfung erfüllen. LC bedeutet einfach nur, dass Kleinbuchstaben (**l**ower **c**ase) erlaubt sind.

Unter die Zeile für das Feld B1KTONUM schreiben wir die Funktion CHECK(RZ). Diese Prüffunktion sagt dem System, es soll nach Drücken der Returntaste den Feldinhalt rechtsbündig anordnen und mit Nullen auffüllen (**r**ight **z**ero). Macht natürlich nur bei numerischen Eingabefeldern Sinn, damit kann man aber optisch unterschiedliche numerische Werte differenzieren. Zum Beispiel kann man eine Kundennummer rechtsbündig mit Nullen links davor gefüllt anzeigen, im Gegensatz zu einem Geldbetrag, der rechtsbündig ohne vorangestellte Nullen, aber mit einem Dezimalpunkt angezeigt wird und sich so viel besser fühlt.

Gut, belassen wir es nun mal mit den Feinheiten des Bildschirms und kümmern uns um das Programm und seinen Zweck. Wir können nun also drei Eingabefelder bestücken, und wenn wir Datenfreigabe drücken, wird das Programm beendet. Schon mal nicht schlecht, aber das kann ja nicht alles sein. Wir wollen das Programm nun Schritt für Schritt erweitern.

Zunächst einmal sollten wir nach dem der Benutzer Daten eingegeben hat diese prüfen. Es macht ja auch keinen Sinn, ein Konto ohne Bezeichnung anzulegen, oder ohne Kontonummer. Allerdings sollten wir dem Benutzer auch mitteilen, dass etwas nicht passt. Beginnen wir einfach mal in ganz kleinen Schritten. Das Programm wird jetzt immer wieder mal um einige Zeilen erweitert. Danach solltest Du es jeweils umwandeln, starten und ausprobieren. So lernt es sich am einfachsten.

Zunächst einmal erweitern wir die Bildschirmdatei (bitte mit dem SEU entsprechend die Quelldatei wählen). Gehe also in den Source für KONTOBS. Dort fügen wir ganz unten die folgende Zeile ein:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0017.00      A            B1MELDUNG  00060A O  21  5COLOR(RED)
```

Eine neue Zeile in knalligem Rot.

Wir erstellen also ein neues Feld namens „B1MELDUNG“ mit der Länge 60, dem Datentyp A für Alpha. Das O besagt, dass es ein reines Ausgabefeld (**O**utput) ist. Es wird in Zeile 21 ab Spalte 5 angezeigt. Da das Feld nicht referenziert ist (Schande über mich, Du kannst es gerne auch in unsere Referenzdatei eintragen), habe ich im Funktionsbereich Platz und kann dort die Funktion für eine Ausgabe in Farbe „rot“ angeben.

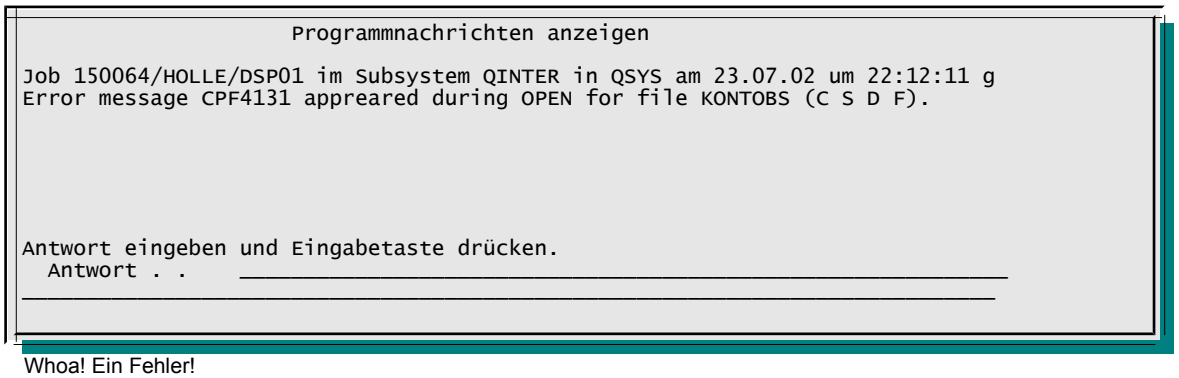
Übrigens: Man muss die Felder nicht unbedingt in der Reihenfolge in der Quelldatei stehen haben, wie sie am Bildschirm angezeigt werden. Allerdings erleichtert es die Behandlung des Quellcodes ungemein, wenn man sich beim Eintippen schon mal das Ergebnis vorstellen will, oder für eine Änderung ein Feld sucht ;-)

Nun bitte den Quellcode verlassen (und natürlich speichern) sowie mit Auswahl 14 umwandeln. Wollen wir mal sehen, was die neue Zeile so bewirkt.

Rufe nun unser Programm auf mit:

```
CALL EDTKONTO DF
```

Hoppla, was ist passiert?Aktuell ist nicht aktuell genug!



Whoa! Ein Fehler!

Autsch, irgendwas mag das System nicht so ganz. Vielleicht ist Deine Meldung auf Deutsch, das hängt von der Systeminstallation der AS/400 ab. Auf jeden Fall haben wir hier eine Fehlermeldung, die AS/400 will etwas von uns. Wer mit dieser Meldung überhaupt nichts anfangen kann, möge den Cursor bitte auf die Zeile mit der Fehlermeldung setzen (die mit `Error message` anfängt). Wir haben hier nämlich zwei Meldungen.

Die erste Zeile gibt nur an, wann der Job gestartet wurde. Die zweite Zeile enthält die letzte Nachricht und ist für uns interessant. Nun drückst Du sanft auf die Taste F1.

Du erhälst einen Bildschirm mit weiteren Nachrichteninformationen. Vielleicht hilft Dir die (evtl. englische) Beschreibung nicht viel weiter, daher gehen wir mal systematisch vor.

Was will die Kiste von uns? Die AS/400 hat einen Fehler festgestellt, und teilt dies dem Benutzer durch eine Nachricht mit. Diese Benachrichtigung funktioniert ebenso wie das Senden einer Nachricht mit SNDMSG. Genau diesen Befehl nutzt auch der Compiler, um auf sich aufmerksam zu machen. Da es mehrere Möglichkeiten gibt, in diesem Falle fortzufahren, wartet das System nun auf eine Antwort vom Operator oder dem Programmbenutzer, damit es weiß, was nun passieren soll. Diese Benachrichtigungen kann man auch automatisiert abfangen und auswerten.

Hinter der Fehlermeldung stehen in Klammern die möglichen Antworten an das System, im erweiterten Hilfebild steht auch ausgeschrieben, was die Buchstaben bedeuten. Im Falle von Fehler CPF4131 (das ist die Fehlernummer, sollte man sich merken) gibt es die Möglichkeiten C, S, D und F. Die erste Auswahl bedeutet „Programm abbrechen“ (**C**ancel), die anderen drei Auswahlen bieten unterschiedliche Dump-Ausgaben, die dem erfahrenen Programmierer zeigen, was schief gelaufen ist. Wir begnügen uns mit der Auswahl *C* (bitte in die Antwortzeile eingeben und DF drücken). Das Programm wird abgebrochen, Du landest wieder im SEU-Bildschirm.

Aber was ist nun eigentlich genau der Fehler CPF4131? Mit diesem Fehler wirst Du oft konfrontiert, da wir eine Eigenart (oder Schutzfunktion) der AS/400 übersehen haben.

Fehlernummer identifizieren

Wenn Du über eine CPFxxxx-Meldung mehr erfahren willst, tippe mal den Befehl:

DSPMSGD *CPF4131* (oder die entsprechende Nummer) ein.

Dies bedeutet „display message description“ und zeigt Dir eine Menge Details. Es folgt ein Menü, in dem Du bitte den Punkt 1 auswählst. Ich überlasse Dir die Forschung in der Nachrichtenbeschreibung. Kurz gesagt bedeutet CPF4131 lapidar „Aktualitätsprüfung“, folgender Absatz ist für uns sehr interessant:

```
                    Formatierten Nachrichtentext anzeigen
                                                          System:   S4404756
Nachrichten-ID . . . . . :   CPF4131
Nachrichtendatei . . . . . :   QCPFMSG
  Bibliothek . . . . . :     QSYS

Nachricht . . . :   Aktualitätsprüfung für Datei, &2 Bibliothek &3, Teildatei
  &4.
Ursache . . . . . : Die Eröffnung der Datei &1 wurde angefordert. Tatsächlich
  eröffnet wurde Datei &2 in Bibliothek &3. Bei der tatsächlich eröffneten
  Datei stimmt die Satzformat-Aktualitäts-ID des Programms nicht mit der
  tatsächlich eröffneten Datei überein.
```

Der genaue Inhalt der Nachricht CPF4131

Da in unserem Programm nur extern beschriebene Dateien (KONTOPF und KONTOBS) verwendet werden, prüft das System bei jedem Programmstart für zu öffnende Datei, ob die Aktualität also die Version von Programm und Datei zusammenpasst. Wenn das Erstellungs- oder Änderungsdatum der *Datei* neuer als das Datum des *Programmes* ist, geht das System (meist zu Recht) davon aus, dass in der Datei Änderungen vorgenommen werden, die den reibungslosen Ablauf des Programms stören können.

Zum Beispiel könntest Du ein Feld gelöscht haben, und das Programm benötigt dieses. Wir haben zwar in unserem Fall ein Feld hinzugefügt, aber trotzdem ist das System hier penibel. In der Beschreibung zum Nachrichtentext schlägt das System verschiedene Lösungen für das Problem vor:

```
Fehlerbeseitigung: Einen der folgenden Schritte durchführen und die
  Anforderung wiederholen:
    - Das Programm erneut umwandeln.
    - Den Parameter LVLCHK mit Hilfe des geeigneten Befehls OVRDBF (Mit
  Datenbankdatei überschreiben), OVRDSPF (Mit Bildschirmdatei überschreiben),
  OVRICFF (Mit ICF-Datei überschreiben) oder OVRPRTF (Mit Druckerdatei
  überschreiben) mit dem Wert *NO überschreiben.
```

Das System hat Ideen parat.

Die einfachste Lösung ist auf jeden Fall der erste Vorschlag: Das Programm, das die Datei benutzt, neu umzuwandeln. Bitte beachte, dass bei der Aktualitätsprüfung das Objekt vom Typ „Datei“ geprüft wird, nicht die Quelldatei. Du kannst also gefahrlos eine Quelldatei für einen Bildschirm ändern, wie Du magst. Erst wenn Du die Bildschirmdatei mit `CRTDSPF` neu erstellst, wird das benutzende Programm sich aufregen.

Das Gleiche gilt natürlich für alle Typen von Dateien.

Diese Regel sollte man sich immer im Hinterkopf behalten: Wenn Du an Dateibeschreibungen etwas änderst, wandele die zugehörigen Programme um. Dann wirst Du mit `CPF4131` nicht oft belastet. Natürlich kann dies bei vielen Programmen und Dateien einiges an Arbeit bedeuten, aber wir werden Mittel kennen lernen, die Umwandlungsarbeit zu automatisieren.

Bitte wandele nun das Programm `EDTKONTO` noch einmal um. Es sollte hier kein Fehler auftreten. Nun rufe das Programm wieder auf, und...

Hm... Du siehst keine Veränderung. Sowas aber auch, da machen wir uns so viel Mühe, und nichts passiert. Nun... überleg mal... was haben wir gemacht? Wir haben ein neues Feld hinzugefügt, das am unteren Rand des Bildschirms angezeigt werden soll. Aber, was haben wir *nicht* gemacht? Richtig, dem Feld einen Inhalt gegeben. Das Feld wird zwar angezeigt, aber da es leer ist, sehen wir nichts davon. Klingt logisch, und ist auch so :-)

Also sollten wir vor der Bildschirmausgabe (`EXFMT`) die Variable `B1MELDUNG` mit einem Inhalt belegen. Füge also nach der Variablendefinition und vor dem `EXFMT` folgende Zeile in den Source für `EDTKONTO` ein:

```
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0010.00 C                   EVAL      B1MELDUNG='Hallo, bitte tipp was ein!'
```

Nur die zweite eintippen, die erste Zeile ist zur Orientierung

Eine Bemerkung am Rande: Bitte die Zeilennummern, die ich hier im Manual angebe, nicht immer als absolut festgelegt betrachten. Du solltest selbst prüfen, an welche Stelle ein Befehl gehört. Ebenso kann ich die Spaltenformatierung nicht immer hundertprozentig hier im Manual darstellen; beim Erstellen des PDF geht auch schon mal was schief, trotz StarOffice. Wenn Du Dir nicht sicher bist, welches Wort in welche Spalte gehört, gehe wie folgt vor:

Füge mit `I` im Zeilennummernbereich eine Leerzeile ein, belege die erste Spalte im Datenbereich mit dem Kennzeichen für das Zeilenformat und prompte mit `F4`. Noch eine nette Funktion vom SEU ist diese: Üblicherweise siehst Du nur am Bildschirmkopf eine Formatierungszeile (das ist die mit `FMT` beginnende Zeile), die Dir bei den Spalten hilft. Wenn Du aber auf einem Bildschirm mehrere Zeilen verschiedenen Typs hast, wird oft nicht die benötigte Formatierungszeile angegeben. Du kannst Dir aber jederzeit eine weitere holen. Tippe zum Beispiel in die Zeilennummer einer `C`-Zeile den Zeilenbefehl `FC` und drücke `DF`. SEU fügt Dir eine Formatierungszeile ein. Der Befehl lautet `F`, der Parameter ist der gewünschte Zeilentyp.

Weiterhin gilt, dass die Spaltenorientierung etwas „flexibel“ ist. Befehle aus RPG3 wie z.B. `MOVEL`, `Z-ADD` etc. haben eine andere Spaltenformatierung wie der RPG4-Befehl `EVAL`. Probiere es aus, in dem Du im Prompting in das Feld „Operation“ eingibst: `EVAL` und dann die `DF` drückst. SEU wird ein fehlendes Feld mokieren. Beende das Prompting mit `F12` und beobachte die Formatierungszeile am Bildschirmkopf. Ungewohnt, aber man lernt, damit zu leben, und oftmals auch, diese Eigenarten zu nutzen.

Wenn ich Dich nun auffordere, das Programm erneut zu testen, sollte klar sein, dass Du dazu den SEU mit `F3` beenden musst, den Source hierbei natürlich sichern solltest. Nun mit Auswahl `14` umwandeln und mit `CALL` starten. Das ist alles Routine :-)

Dein Bildschirm „`Verwalten Kontodaten`“ sollte nun einen Fusstext in roter Schrift haben. Fein!

Du hast also gesehen, wie man eine einfache Statuszeile verwendet. Löschen wir nun wieder die Vorbelegung für das Feld `B1MELDUNG` und wenden uns der Programmlogik zu.

Bisher können wir etwas eingeben, aber das Programm interessiert sich nicht dafür. Ausserdem sollten wir ein paar Funktionstasten einbauen, damit man zum Beispiel mit `F12` die Funktion abbrechen kann. Aber wieder: Schritt für Schritt.

Funktional – phänomenal

Wir ändern das Programm so ab, dass Ausgabe und Auswertung des Bildschirms in einer Schleife laufen. Allerdings sollte jede Schleife irgendwann beendet werden, sonst ärgert sich der Benutzer, dass er das Programm nicht sauber beenden kann. Ich habe schon oft solche Programme gesehen, die wegen eines Logikfehlers in einer Endlosschleife landen und man nicht mehr mit den angegebenen Funktionstasten sauber rauskommt. Unbedarfte Benutzer schalten in einem solchen Fall oft einfach den Bildschirm aus oder beenden das Terminalprogramm. Nicht wirklich die richtige Lösung, und die AS/400 beschwert sich mit einem mehr oder weniger ausführlichen Fehlerprotokoll. In großen Programmen habe ich nach einem solchen Abbruch schon Fehlerprotokolle mit über 30.000 Seiten gehabt. Die helfen zwar niemandem, belegen aber Speicher.

Unser Programm soll auf jeden Fall auf die Taste `F12` reagieren, mit der ein Benutzer sagt: „*Raus, ich mag nicht mehr, ignoriere meine Eingaben!*“. Die Behandlung der Funktionstasten wird im Bildschirmformat definiert, das Programm fragt nur den Status ab. Wenn Du derzeit im Programm versuchst, das Programm mit `F3` oder `F12` abzubrechen, wird Dir mitgeteilt, dass diese Funktionstaste nicht erlaubt ist. Wir haben ja auch nirgendwo definiert, dass der Benutzer diese Taste benutzen darf.

Hierfür gibt es eine Funktion für DDS-Dateien, die da heisst `CFXX`. Das `XX` steht hierbei für die Nummer der Funktionstaste, die erlaubt ist. Bitte starte SEU für die Quelldatei von `KONTOBS`. Füge nach der Definition des Satzformates `DSPKTO1` diese Zeile ein:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0002.01      A                                      CF12(12 'Abbruch')
```

Wir erlauben eine Funktionstaste!

Die Funktion CF12 gehört natürlich wieder in den Funktionsbereich, orientiere Dich einfach an der Angabe für die Referenzdatei zwei Zeilen höher. Beachte, dass bei Bildschirmdateien die Zeilenart zwar A lautet, die Formatierungszeile aber DP als Zeilenart angibt. Ist halt so.

Da CF12 nicht unterhalb eines Feldes steht, sondern über allen Feldangaben im Bereich des Formats DSPKTO1, gilt diese Funktion für das gesamte Format, dies nennt man eine formatbezogene Definition. Du könntest die Funktion CF12 auch direkt unterhalb der Angabe der Referenzdatei, aber oberhalb aller Formatdefinitionen angeben, dann ist F12 für alle Formate in dieser Datei erlaubt, dies nennt man eine dateibezogene Definition. Allerdings ist es nicht immer sinnvoll, für alle Formate einer Bildschirmdatei eine Taste generell zu erlauben, da Du in einer Bildschirmdatei viele unterschiedliche Bildschirme definieren kannst. Die Standardtasten F1, F3 und F12 sollten aber in einem guten Bildschirm immer vorhanden sein.

In der Funktion verwenden wir noch zwei Parameter. Der erste Parameter 12 bedeutet, dass die Anzeigevariable 12 eingeschaltet wird, wenn F12 gedrückt wird. Der Text in Hochkommata kann für Hilfsreferenzen verwendet werden.

Genug geredet. Bitte erstelle die Bildschirmdatei mit Auswahl 14 neu; danach wenden wir uns der Quelle für unser Programm zu.

Wir brauchen jetzt eine Schleife, in der immer wieder das Bildschirmformat angezeigt wird, bis der Benutzer die Taste F12 drückt. Erst dann sollte die Schleife beendet werden. Später fügen wir weitere Funktionstasten hinzu, die eine entsprechende Funktion auslösen. Erst bei F12 beenden wir das Programm, bei anderen Funktionstasten läuft es eventuell weiter.

Schleifen bis zum Ende

Bitte wechsle in den Programmquelltext und ändere den Hauptteil wie folgt ab:

```
 Spalten . . . :   1  71          Editieren                    LIBMANUAL/QRPGLESRC
 SEU==> ____________________________________________________________     EDTKONTO
 FMT DP Fdateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter+++++++++++++++++
0001.00  *   *** Mein erstes interaktives Programm
0002.00  *
0003.00  * Dateidefinitionen
0004.00 FKONTOPF   UF A E           K DISK
0005.00 FKONTOBS   CF   E             WORKSTN
0006.00  * Initialisierung
0007.00 C                   EVAL      *IN12 = *OFF
0008.00  * Hauptschleife
0009.00 C                   DOW       *IN12 = *OFF
0010.00 C                   EXFMT     DSPKT01
0011.00 C                   ENDDO
0012.00
0013.00  * ENDE!
0014.00 C                   EVAL      *INLR = *ON
        ******************Datenende ***********************************************

F3=Verl. F4=Bed.-Führg. F5=Aktual. F9=Auffinden F10=Pos.-Anz. F11=Umschalten
F16=Suchvorgang wiederholen   F17=Änderung wiederholen F24=Weitere Tasten
```

Unser Verwaltungsprogramm mit Schleifchen :-)

In Zeile 7 siehst Du den `EVAL`-Befehl. Dieser dient der Zuweisung eines Wertes zu einer Variablen. Du bemerkst den Variablennamen `*IN12` und erinnerst Dich an die letzte Programmzeile, die ähnlich aussieht. Die Variablen mit Namen `*INxx` sind besondere Variablen, die man zur Programmsteuerung verwenden kann. Sie sind definiert als einstellige Alphavariablen, aber sie können nur die Werte '0' (Null, nicht) oder '1' (Eins, ja) beinhalten. Alternativ kannst Du für '0' auch den Sonderwert `*OFF`, und für '1' den Sonderwert `*ON` verwenden. Diese beiden Sonderwerte sind gleichwertig zu '0' und '1' und können im Programm auch gemischt verwendet werden. Der Programmierer kann sie nach Geschmack verwenden, abhängig davon, was für ihn lesbarer erscheint. Ich persönlich bevorzuge `*OFF / *ON`. Man nennt diese einstelligen Sondervariablen auch „Anzeiger“ oder „Bezugszahl“, englisch „Indicator“, da sie einen binären Zustand anzeigen. Mit ihnen kann man eine Menge netter Schweinereien anstellen, elegante Routinen mit wenigen Zeilen erstellen, aber durch übermäßigen Gebrauch ein Programm fast unleserlich gestalten :-)

Du siehst, dass ich im Block „Initialisierung“ erst mal die Bezugszahl 12 auf AUS setze. Man sollte nicht nur wegen der Übersicht die verwendeten Variablen vor Gebrauch initialisieren.

Danach beginnt mit dem Befehl `DOW` eine Schleife. Ausgesprochen heißt der Befehl „do while“, also tue solange, und als Bedingung steht dahinter „Bezugszahl 12 ausgeschaltet ist“.

In der Schleife wird mit EXFMT der Bildschirm aufgerufen, danach mit ENDDO die Schleife beendet.

Ist doch ganz einfach, oder? Nun wird im Programm der Bildschirm so lange ausgegeben, bis der Benutzer mit der Taste F12 demonstriert, dass er genug hat :-)

Probier's aus!

Gut, wie wäre es denn mit einer Zeile, die dem Benutzer mitteilt, dass er F12 drücken darf? Das macht das System bei normalen Bildschirmdateien leider nicht automatisch, auf der anderen Seite kann man das mit etwas Aufwand selbst recht flexibel erledigen. Also rein in den Quelltext und über der Zeile mit der Fehlermeldung eine Zeile mit der Angabe der F-Taste erstellen. Mein Vorschlag wäre:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0019.00      A                                      22  2'F12=Abbruch'
0020.00      A                                          COLOR(BLU)
```

Ein simpler Text zeigt eine F-Taste an

Na, wie sieht das aus? (Für die Kenner der Materie: Ja, es geht noch ganz anders... nicht so ungeduldig!) Fein, oder? Nun habe ich Dir einen kleinen Schönheitsfehler eingebaut. Wenn Du in die Felder etwas eingetippt hast und mit DF bestätigt hast, wirst Du bestimmt bemerkt haben, dass beim erneuten Ausgeben die Felder wieder leer sind. Woran liegt das? Schau bitte noch einmal genau in den Quelltext für die Bildschirmdatei.

Betrachte die Definition für die ersten drei Felder. Links von der Angabe von Zeile und Spalte haben wir in der Spalte FLDART ein I eingetragen. Das bedeutet „nur Eingabe“ (**I**nput). Du kannst zwar dort etwas eintippen, aber mit der Ausgabe später hapert es etwas. Wenn Du unser Meldungsfeld betrachtest, dort haben wir das Feld als O („nur Ausgabe“, **O**utput) definiert. Nun wollen wir aber beides haben.

Ganz einfach: Definiere die Eingabefelder als B („both“, **b**eides). Schon bleibt der Feldinhalt nach Datenfreigabe erhalten. (Ein Leser hatte das Phänomen, dass sein Kontoprogramm nach dem Lesen eines Satzes zum Bearbeiten die vorhandenen Daten nicht gezeigt hat. Das war genau der Fehler.)

Gut, machen wir in großen Schritten weiter. Wir wollen nun einiges an Funktionalitäten einbauen. Zunächst soll nach Eingabe durch den Nutzer geprüft werden, ob seine Eingaben auch einen Sinn machen bzw. auch vollständig sind.

Dazu gehört nach dem EXFMT für die Bildschirm eine Routine, die prüft, ob die Eingaben korrekt sind. Ausserdem sollten wir in einer Variable speichern, ob die Eingaben überarbeitet werden sollten. Hintergrund ist, dass entschieden werden muss, ob die Schleife beendet werden soll (F12), oder die Eingaben nicht ok sind (bitte nochmal eingeben), oder alles ok ist und mit den Eingaben etwas angestellt werden soll (speichern vielleicht?).

Alles eine Definitionsfrage

Wir benötigen also eine Routine mit diversen Prüfungen, sowie eine Variable, die vermerkt, ob die Eingaben brauchbar sind.

Gehen wir also in Quelltext unseres Programms und fügen einige Zeilen hinzu.

Zunächst benötigen wir eine Variable. Da wir dort eigentlich nur zwei Zustände benötigen (Eingabe ok / nicht ok), reicht eine logische Variable, wie sie die Bezugszahlen darstellen. Wir verwenden aber keine Bezugszahl *INxx, sondern einen anderen Namen. Hintergrund ist, dass man die Bezugszahlen am besten für Steuerungen im Bildschirm und bei der Druckausgabe verwenden kann, und somit sparsam damit umgehen sollte, wir haben ja nur 99 davon :-)

Definieren wir uns also eine Variable namens #EINOK, dies sieht wie folgt aus: (bitte diese Zeilen direkt hinter den Dateispezifikationen, Zeilenart F eingeben):

```
FMT D   DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0005.01 * === Variablen
0005.02 D#EINOK           S               1A   INZ(*OFF)
```

Einfache Variablendefinition

Zunächst eine Kommentarzeile mit beliebigem Inhalt (nur der * ist wichtig), danach die Definitionszeile (daher auch der Zeilentyp D). Der Name „#EINOK“ ist frei gewählt, ich verwende gerne ein # als Namensbeginn, um eine einstellige Variable als Anzeiger oder als Arbeitsfeld innerhalb eines Programms zu kennzeichnen.

Prompte bitte die D-Zeile, um die einzelnen Felder zu sehen. (Wie bekannt, erst das D in die Typ-Spalte nach Zeilennummer und Leerzeichen eingeben und direkt F4 drücken. Ignorieren wir die Felder „E“ und „S/U“. In „Deklarationsart“ bitte ein S eintragen. Dies bedeutet, dass Du ein einzelnes Feld definieren willst. Hier kannst Du andere Angaben machen, siehe F1 (was die einzelnen Möglichkeiten bedeuten, werden wir noch kennen lernen). Danach folgen zwei Felder „Von“ und „Bis / Länge“, die angeben, wie groß die Variable definiert werden soll. Da wir nur eine einzelne Variable benötigen, können wir uns „Von“ sparen und geben nur die Länge an. Später, wenn wir Datenstrukturen definieren, wird es hier schon interessanter. Die interne Datenart lautet A für alpha. Dezimalstellen hat unser Feld ganz bestimmt nicht. Das Schlüsselwort INZ mit Parameter gibt an, welchen Wert die Variable beim Programmstart haben soll. So können wir uns bei einfacheren Programmen die Programmierung eines Initialisierungsblocks sparen.

Moment! Warum initialisieren wir dann später die Bezugszahl 12? Klar! Diese wird in der Bildschirmdatei definiert, dort aber nicht initialisiert. Logisch.

Fein, wir haben unsere erste selbst gebastelte Variable, nun müssen wir sie auch verwenden.

Also folgt nun eine Routine zur Prüfung der Felder. Wenn alles ok ist, wird #EINOK auf AN gesetzt, ansonsten wird sie auf AUS gesetzt. Die Routine wird aber nicht ins Hauptprogramm geschrieben, sondern wir machen eine Unterroutine, damit alles übersichtlich bleibt.

Mit Routine an die Arbeit!

Unterroutinen wird wohl jeder kennen, der schon einmal programmiert hat. Sie bringen Übersichtlichkeit in den Programmtext, ausserdem kann man so mehrfach verwendete Befehle sparsam, also nur einmal im Programm unterbringen. In RPG werden Unterroutinen üblicherweise am Schluß eines Programms definiert. In unserem Fall ist dies also hinter der letzten Zeile, in der der Anzeiger `*INLR` für die Anzeige des Programmendes. Und das ist auch in den anderen Fällen so.

Füge also dahinter bitte erst mal ein oder zwei Leerzeilen und eine Kommentarzeile mit einigen Sternchen ein, damit wir die Unterroutine auch optisch vom Programm getrennt haben.

Danach folgt die Definition der Unterroutine, die immer aus zwei Zeilen besteht, dem Anfang und dem Ende (Alpha und Omega gibt es fast überall ;-)

Dies sieht so aus:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0019.00 C     CHECK01        BEGSR
0020.00 C                    ENDSR
```

Eine sehr leere Unterroutine

Hiermit hätten wir unsere Unterroutine namens `CHECK01` definiert. In der ersten Zeile muss im Feld „`Faktor1`“ der Name der Routine, sowie als Operation „`BEGSR`“ angegeben werden. Beendet wird die Routine mit der Operation „`ENDSR`“. Sieht sehr übersichtlich aus, nun müssen wir dazwischen noch etwas Leben einbauen.

In unserem Beispielprogramm sollen also zumindest das Feld `B1KTONUM` sowie `B1KTOTXT` gefüllt sein. Fehlt eine Angabe, soll die Eingabe als ungültig erklärt werden, Variable `#EINOK` auf AUS gesetzt werden.

Dazu benötigen wir ein paar Abfragen.

Was ist wenn?

Abfragen in Programmen wird auch jeder Programmierer schon mal erstellt haben, stellen sie doch das Herz jeder Logik dar. Nehmen wir folgende Variante: Zunächst glauben wir an das Gute im Anwender und setzen die Variable `#EINOK` auf „AN“. Dann prüfen wir, ob unsere Anforderungen erfüllt werden (beide Felder belegt). Wenn nicht, wird `#EINOK` auf „AUS“ gesetzt. Das sieht so aus:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0018.01 C                   EVAL       #EINOK = *ON
```

Prüfen wir nun in unserer Routine, ob `B1KTONUM` leer ist, bzw. Null ist (es handelt sich ja um ein numerisches Feld).

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0019.01 C                   IF         B1KTONUM = 0
0020.00 C                   EVAL       #EINOK = *OFF
0021.00 C                   ENDIF
```

Wenn, dann, jawoll!

Liest sich einfach: Wenn `B1KTONUM` = Null ist, dann setze Variable `#EINOK` auf AUS. Jeden `IF`-Block muss man mit `ENDIF` abschließen.

Nun müssen wir noch das zweite Feld `B1KTOTXT` prüfen. Dies könnten wir in einer zweiten `IF`-Abfrage erledigen, also:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0022.00 C                   IF         B1KTOTXT = *BLANKS
0023.00 C                   EVAL       #EINOK = *OFF
0024.00 C                   ENDIF
```

Hier wird nicht auf Null geprüft, da `B1KTOTXT` alphanumerisch ist. Wir prüfen, ob das Feld leer ist, in RPG bedeutet dies, mit Leerzeichen gefüllt. Der Sonderwert `*BLANKS` ist gültig für die Aussage „leeres Feld, kein Inhalt, nur Leerzeichen“.

Man sollte alphanumerische Felder immer mit *BLANKS auf „leer“ prüfen. Würdest Du schreiben:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0022.00 C                   IF        B1KTOTXT = ''
```

dann beschwert sich die AS/400 über ein ungültiges Token. Damit ist die leere Zeichenkette hinter dem Gleichheitszeichen gemeint. Sowas wie ein einzelnes Leerzeichen als String / Zeichenkette gibt es hier nicht. Also bleiben wir bei der Variante mit `*BLANKS`.

Sei elegant!

Nun ist die zweifache `IF`-Abfrage nicht ganz elegant, da wir das auch in einer Prüfung erledigen können. Da es reicht, wenn eines der beiden Felder nicht belegt ist, könnten wir auch schreiben:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0019.01 C                   IF        B1KTONUM = 0
0019.02 C                             OR
0019.03 C                             B1KTOTXT = *BLANKS
0020.00 C                   EVAL      #EINOK = *OFF
0021.00 C                   ENDIF
```

Sein oder nicht sein, das ist hier die Frage!

Das Beispiel dürfte verständlich sein. Die Aufteilung der Abfrage auf drei Zeilen 0019.01 bis 0019.03 ist rein optischer Natur. Gerade kompliziertere Abfragen sind auf mehrere Zeilen aufgeteilt erheblich lesbarer. Aber genauso gut können wir unsere drei Zeilen so zusammenfassen:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0019.01 C                   IF        B1KTONUM = 0 OR B1KTOTXT = *BLANKS
```

Die Abfrage in komprimierter Form

Sieht doch auch nicht schlecht aus. Wie Du es machst, ist egal, der Effekt ist der selbe.

Nun haben wir also eine Prüfroutine, die nach unseren Wünschen eine Variable belegt. Was nun? Zunächst müssen wir nach der Ausgabe und Eingabe des Bildschirmformats unsere Prüfroutine aufrufen. Also zwischen `EXFMT` und `ENDDO` folgende Zeile einfügen:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0013.01 C                   EXSR      CHECK01
```

Hallo Routine, ich rufe Dich!

Der Befehl EXSR steht für „**ex**ecute **S**ub**r**outine“, also Unterroutine ausführen. Nun wird also geprüft, ob die Eingaben, die der Nutzer macht, auch einen Sinn ergeben.

Wie teilen wir nun dem Anwender das Ergebnis der Prüfung mit? Logisch, wir haben doch ein Meldungsfeld definiert, also lass' es uns benutzen! Geben wir nun aus, dass entweder beide Felder eingegeben werden müssen, oder dass alles ok ist. Dies kann man so erledigen (bitte direkt nach dem Aufruf der Unterroutine eingeben):

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0014.01 C                   IF        #EINOK = *OFF
0014.02 C                   EVAL      B1MELDUNG = 'Bitte Kontonummer und +
0014.03 C                             Kontobezeichnung eingeben!'
0014.04 C                   ELSE
0014.05 C                   EVAL      B1MELDUNG = 'Alles ok!'
0014.06 C                   ENDIF
```

Hier haben wir zwei Zweige in der IF-Abfrage, der erste, wenn die Prüfung erfolgreich ist (also #EINOK ausgeschaltet ist), und nach der Operation „ELSE“, wenn die Abfrage nicht zutrifft. Beachte bitte die Zeilen 0014.02 und 0014.03, hier erstreckt sich aus Platzgründen die Zuweisung des Textes zur Variablen B1MELDUNG über zwei Zeilen. Erreicht wird dies, in dem in der ersten Zeile hinter dem Wort „und“ die zuzuweisende Zeichenkette nicht mit einem einfachen Hochkomma abgeschlossen wird. Statt dessen wird am Zeilenende ein Pluszeichen angegeben, und die Zeichenkette im erweiterten Faktor 2 fortgesetzt und erst hier mit dem einfachen Hochkomma abgeschlossen. Beachte bitte, dass Du in der ersten Zeile nach dem „und“ ein Leerzeichen belässt, ansonsten werden nachher im Ergebnis die Worte „und“ sowie „Kontobezeichnung“ ohne Leerzeichen zusammengefügt!

Probiere nun unser Programm aus.

Auch das funktioniert. Da Du nun weisst, wie man IF-Abfragen mit ELSE programmiert, könnten wir unsere Prüfroutine eleganter gestalten:

```
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0019.01 C                   IF        B1KTONUM > 0 AND B1KTOTXT <> *BLANKS
0019.02 C                   EVAL      #EINOK = *ON
0019.03 C                   ELSE
0020.00 C                   EVAL      #EINOK = *OFF
0021.00 C                   ENDIF
```

Das sieht doch schon recht gut aus, oder?

Nun fehlt dem Programm aber noch etwas der Pepp. Schließlich müssen wir die geprüften Eingaben auch abspeichern, sonst nützt das schönste Pflegeprogramm nichts. Erweitern wir also wieder etwas und lernen bei der Gelegenheit gleich noch ein paar Dateibefehle kennen.

Machen wir uns es etwas einfach. Der Benutzer gibt ein paar Daten ein. Dann prüfen wir, ob die Daten logisch korrekt sind. Wenn ja, prüfen wir, ob in der Datei schon ein Satz mit der angegebenen Kontonummer existiert. Sollte dies nicht der Fall sein, schreiben wir einen neuen Satz in die Datei, ansonsten geben wir eine Meldung aus, daß dieser Satz bereits existiert.

Bisher haben wir bei korrekten Eingaben nur die Variable B1MELDUNG zur Ausgabe des Texts „Alles OK!“ gesetzt. Diese Zeile ersetzen wir nun durch den weiteren Ablauf. Dafür gibt es eine neue Unterroutine.

Mach eine kurze Verschnaufpause, dann lies weiter.

Alles Routine – nächster Teil

Fügen wir also unserem Programm eine neue Unterroutine hinzu, die das Schreiben des neuen Satzes erledigt. Schau Dir mal folgende Zeilen an:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
 FMT CX CL0N01Factor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0036.00 C     WRITE01        BEGSR
0037.00 C                    EVAL       KTONUM = B1KTONUM
0038.00 C                    EVAL       KTOTXT = B1KTOTXT
0039.00 C                    EVAL       KTODAT = B1KTODAT
0040.00
0041.00 C                    WRITE      FMTKTO
0042.00
0043.00 C                    EVAL       B1KTONUM=0
0044.00 C                    EVAL       B1KTOTXT=*BLANKS
0045.00 C                    EVAL       B1KTODAT=0
0046.00 C                    EVAL       B1MELDUNG = 'Daten wurden geschrieben.'
0047.00 C                    ENDSR
```

Unsere Routine zum Schreiben

Hier müssen wir zuerst das wichtigste erledigen. Der Inhalt unserer drei Bildschirmvariablen muss den Feldern in der Datendatei zugewiesen werden. Sonst würden wir einen sehr leeren Satz schreiben, und das ist ja nicht in unserem Sinne.

Ach ja: Wer sich wundert, in den obigen Bildschirm habe ich mal beide Formatzeilen für C-Bestimmungen kopiert, einmal `FMC C` und `FMT CX`. Erster ist für RPG-Befehle mit fest fixierten Operanden, letzterer ist für RPG4-Befehle mit längeren Parametern, wie beispielsweise `EVAL`. Du merkst den Unterschied?

Dann kommt ein neuer Befehl: `WRITE`, der genau dies tut. Er schreibt unseren neuen Datensatz in die Datei. Danach räumen wir etwas auf, leeren unsere Eingabefelder am Bildschirm und teilen dem Benutzer mit, dass der Satz erfolgreich hinzugefügt wurde. Diese Routine muss nur noch aufgerufen werden.
Tue dies in unserer Hauptroutine, wo wir den Inhalt unserer Anzeigevariablen `#EINOK` prüfen.

```
0015.00 C                   IF        #EINOK = *OFF
0016.00 C                   EVAL      B1MELDUNG = 'Bitte Kontonummer UND +
0017.00 C                             Kontobezeichnung eingeben!'
0018.00 C                   ELSE
0019.00 C                   EXSR      WRITE01
0020.00 C                   ENDIF
```

Es wird ernst!

So sollte die Abfrage nun aussehen. Wandle das Programm um und starte es.
(Habe ich schon erwähnt, dass Du nicht immer `call <programm>` eingeben musst, um ein Programm zu starten? Nein? Ok, es geht einfacher, in dem Du *nach* dem Umwandeln vor dem Eintrag in der SEU-Liste statt Auswahl `14` die Auswahl `C` (für „call“) eingibst. Man ist ja faul...)

Fülle nun die Eingabefelder in unserem Programm mit den folgenden Daten:

```
                                Verwalten Kontodaten

    Kontonummer:   000001_

    Kontotext  :   Mein_erstes_Konto_____________

    Anlagedatum:   20020901__


 F12=Abbruch
```

Wir legen einen Satz an

Dann drücke `DF`. Unser Programm sollte uns nun darüber informieren, dass der Satz geschrieben wurde. Probiere es mit SQL aus. Beende zunächst das Programm mit `F12`.

Zur Erinnerung:
`strsql` eingeben. Dort folgenden Befehl ausführen:

```
select * from kontopf
```

Fein, gelle?

Gib noch einen Datensatz für das Konto 2 ein.

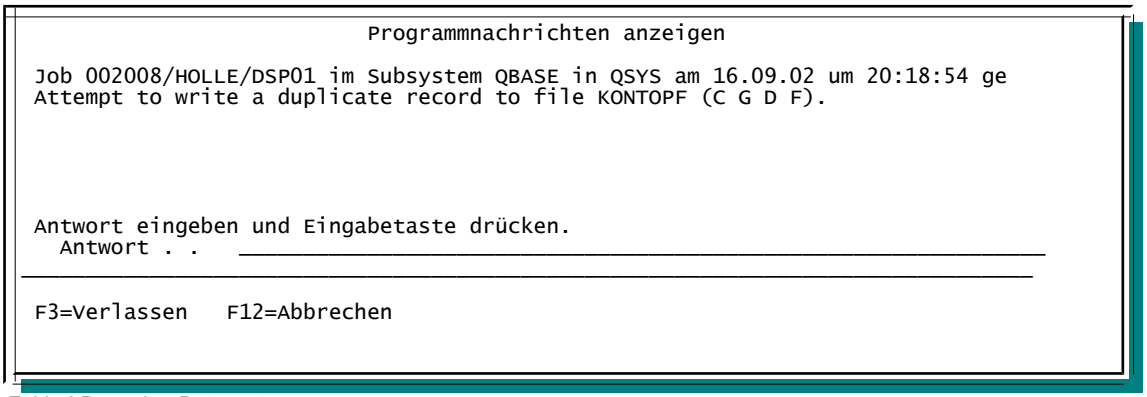
Alles doppelt oder was?

Soderle, nun haben wir schon mal ein Programm, das Datensätze schreibt. Wie wir Datensätze der Reihe nach lesen können, wissen wir auch. Das ist doch schon mal was. Nun kommen wir langsam aber sicher an das Eingemachte, dann folgt ein kurzer Überblick, und es kann richtig los gehen.

Was ich nun ansprechen möchte, ist der erste Schritt zur richtigen Fehlerbehandlung. „Fehler? Ich mache keine Fehler!“ glaubt jeder und liegt damit falsch. Jeder macht Fehler, und oftmals sind es die banalsten Fehler überhaupt.

Was kann denn nun bei unserem tollen Programm noch schiefgehen? Eine ganze Menge! Zum Beispiel könnte der Anwender eine Kontonummer eingeben, die bereits existiert. Probiere aus, was passiert.

Folgende Meldung kommt, wenn ich Daten für ein Konto Nummer 1 eingebe, und ein entsprechender Datensatz schon existiert:



Fehler! Doppelter Datensatz

Unmissverständlich macht uns das System darauf aufmerksam, dass versucht wurde, einen doppelten Datensatz zu schreiben. Warum doppelt? Schau bitte noch mal in die Definition unserer Datendatei `KONTOPF`. Diese haben wir als eindeutig (`UNIQUE`) definiert. Als eindeutiger Schlüssel ist mit Namensart „`K`“ (Key) nur das Feld `KTONUM` belegt. Also darf in dieser Datei jeder numerische Inhalt in diesem Feld nur einmal vorkommen. Also beschwert sich das System zu Recht.

Aber kann uns die AS/400 nicht höflicher darauf hinweisen? So eine Nachricht sieht ja ziemlich erschreckend und unhöflich aus. Nun, diese Nachrichten werden nur gesendet, wenn der Programmierer zu faul war, sich selbst um die Fehlerbehandlung zu kümmern. Es zeugt von keinem guten Stil, wenn ein Anwender so eine Nachricht sieht :-)

Brich bitte die Nachricht mit `C` ab. Andere Antworten können zu Debugzwecken verwendet werden, das machen wir nachher einmal, da das Lesen eines Dump etwas kompliziert ist.

Fehler abfangen oder Fehler vermeiden?

Nein, das ist keine Fangfrage, sondern eine Frage der Arbeitsweise. Wir haben zwei Alternativen, diesem doppelten Datensatzfehler zu begegnen. Wir können im Falle des Fehlers eine entsprechende Verarbeitung starten, oder wir können imr Voraus die Möglichkeit vermeiden, dass so ein Fehler passiert.

Fangen mir mit der ersten Variante an:

Der `WRITE`-Befehl in unserem Beispiel ist noch nicht vollständig. Man kann zusätzlich den Status der Operation abfragen. Hier kommen (um kompatibel für unsere V3R2-Freunde zu bleiben), die geliebten Bezugszahlen ins Spiel. Editiere bitte den Quellcode für **EDTKONTO** und springe in die Zeile mit dem `WRITE`-Befehl.

Prompte nun den Befehl mit F4 und betrachte die Felder. In der unteren Prompt-Zeile findest Du drei Felder namens **Ho**, **Ni** und **Gl**. Das hat nichts mit Bienenprodukten zu tun, sondern ist die Abkürzung für „Hoch“, „Nicht erfolgreich“ und „gleich“. Diese drei Felder können bei jedem Befehl angegeben werden, machen aber nicht überall Sinn. Sie sind hauptsächlich für Vergleichs- und Dateioperationen gedacht.

In diese drei Felder kannst Du jeweils zweistellig den Namen bzw. die Nummer einer Bezugszahl eintragen. Du erinnerst Dich an die Bezugszahlen, die logische Variablen mit den möglichen Werten „An“ oder „Aus“? Diese sind hierfür gedacht. IBM mußte wohl etwas tricksen, um mögliche Kombinationen damals auf Lochkarten unter zu kriegen, daher die drei etwas komisch benannten Felder. Am Anfang ist es recht schwer, sich zu merken, bei welchem Befehl welche Bezugszahl in welchem Fall gesetzt wird, aber man findet sich zurecht. Und falls man an wichtiger Stelle eine Bz vergisst, wird SEU Dich schon darauf hinweisen.

Beim `WRITE`-Befehl wird die Bezugszahl „Ni“ (ich kürze ab hier mit Bz ab.) für „nicht erfolgreich“ gesetzt, wenn beim Schreiben etwas schief gelaufen ist. Damit kann recht gut festgestellt werden, dass der Satz schon vorhanden ist, andere Fehler beim Schreiben in Dateien dürften wirklich nicht auftreten. Trage bitte in die Bz. „ni“ die Zahl 70 ein. Du solltest Dir angewöhnen, erstens die verwendeten Bezugszahlen im Programm zu dokumentieren, zweitens ein gewisses System einfallen lassen. Da man die Bezugszahl nach der Schreiboperation nicht wirklich weiterhin benötigt, kann man diese mehrfach verwenden. Nutzen wir also die 70 immer, um nach einer Schreiboperation den Status zu erfragen. (Benutzer von Systemen ab V4 brauchen hier garkeine Bezugszahlen.)

Dein Befehl sieht nun so aus:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0041.00 C                   WRITE     FMTKTO                                 70
```

Wir schreiben mit Statusabfrage

Nun kommt vom System keine Meldung mehr, wenn ein doppelter Satz auftritt. In dem Fall wird die Bz.70 auf „An“ gesetzt.

Was mach ich nun?

Gute Frage. Auf jeden Fall sollte man den Anwender darauf hinweisen, dass der Satz nicht geschrieben werden konnte, soviel Höflichkeit muss sein. Wir fragen also nach dem `WRITE`-Befehl unsere Bezugszahl ab und handeln entsprechend in einer `IF`-Abfrage. Bitte ändere die Routine wie folgt ab:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0036.00 C     WRITE01       BEGSR
0037.00 C                  EVAL       KTONUM = B1KTONUM
0038.00 C                  EVAL       KTOTXT = B1KTOTXT
0039.00 C                  EVAL       KTODAT = B1KTODAT
0041.00 C                  WRITE      FMTKTO                                 70
0042.00 C                  IF         *IN70 = *OFF
0042.01  * Alles ok!
0043.00 C                  EVAL       B1KTONUM=0
0044.00 C                  EVAL       B1KTOTXT=*BLANKS
0045.00 C                  EVAL       B1KTODAT=0
0046.00 C                  EVAL       B1MELDUNG = 'Daten wurden geschrieben.'
0047.00 C                  ELSE
0047.01  * Das war nix!
0048.00 C                  EVAL       B1MELDUNG = 'FEHLER: Satz nicht +
0049.00 C                             geschrieben!'
0050.00 C                  ENDIF
0051.00 C                  ENDSR
```

Unsere Routine mit Fehlerabfrage

So, das sieht schon recht gut aus. Probiere das Programm nun und versuche, einen Satz mit doppelter Kontonummer zu schreiben. Unser Programm weist uns freundlich darauf hin und lässt die Eingabefelder so, wie wir sie belegt haben. Somit kann der Anwender die Kontonummer ändern oder generell über seine Eingabe nachdenken.

Alles aktuell?

Jetzt kennen wir schon einige wichtige Befehle. Machen wir gleich weiter mit unserem Programm. Bisher können wir nur neue Sätze eingeben. Nun wollen wir dem Benutzer auch die Möglichkeit geben, vorhandene Sätze zu bearbeiten und wieder zu aktualisieren. Dafür sollten wir unser Programm etwas umbauen, bevor es zu unübersichtlich wird. Unsere Logik wird folgende sein: (Vorsicht, diese Logik muss nicht unbedingt der Weisheit letzter Schluss sein, sie dient nur zur Übung).

Neue Modi

Mit einer Funktionstaste (hier: F6) soll der Benutzer zwischen „Aktualisierungsmodus“ und „Einfügemodus“ wechseln können. Im „Aktualisierungsmodus„ wird geprüft, ob eine neue Kontonummer eingegeben wurde, wenn ja, diese gesucht und im Erfolgsfall geladen und zum Ändern bereitgestellt. Wurde die Kontonummer nicht geändert, gehen wir davon aus, dass der Nutzer den Satz nun aktualisieren möchte. Im „Einfügemodus“ arbeitet das Programm wie bisher.

Hierfür sind einige Änderungen nötig:

Wechsele bitte zunächst in den Quellcode für den Bildschirm KONTOBS. Zunächst wollen wir unseren Bildschirm etwas aufpeppen. Wir fügen der Vollständigkeit halber die F3-Taste hinzu, weiterhin wollen wir dem Benutzer anzeigen, in welchem Modus er sich befindet. Schließlich braucht es noch die Taste F6, um zwischen Einfügen und Ändern zu wechseln.

Die Änderungen am Anfang sind rot markiert, diese bitte hinzufügen:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0003.00      A                                      CF03(03 'Ende')
0004.00      A                                      CF06(06 'Modus')
0005.00      A                                      CF12(12 'Abbruch')
0006.00      A                                  1 35'Verwalten Kontodaten'
0007.00      A                                      COLOR(WHT)
0008.00      A N50                              2 37' (Eingabemodus) '
0009.00      A  50                              2 37'(Änderungsmodus)'
0010.00      A                                  5  5'Kontonummer:'
0011.00      A                                      COLOR(BLU)
0012.00      A            B1KTONUM   R        B  5 20REFFLD(KTONUM)
```

Änderungen in unserer Bildschirmdatei, Teil 1

Die Zeilen 3 und 4 dürfen klar sein, zwei neue Funktionstasten werden als „erlaubt“ definiert. Gleichzeitig werden die passenden Bezugszahlen definiert.
In Zeile 8 kommt etwas ganz neues ins Spiel. Bitte diese Zeile prompten. Hier habe ich nur einen neuen Text definiert, aber vorne in der Zeile in das Feld N01N02N03 etwas eingetragen. Wofür ist dieses Feld gut? Damit kann man Spezifikationen konditionieren, auf deutsch, von Bezugszahlen abhängig machen. In unserer Zeile 8 bedeutet dies, dass diese Zeile nur gilt, wenn Bz.50 **nicht** eingeschaltet ist. Das N steht für „nicht“, die 50 für die Bezugszahl. Insgesamt kann man eine Zeile von bis zu drei Bezugszahlen abhängig machen, das kann aber schon sehr unübersichtlich werden. Also nur vorsichtig Gebrauch davon machen! Man kann diese Vorgehensweise auch anders lösen. Zum Beispiel, in dem man eine Variable ausgibt, die entsprechend vorher im Programm belegt wird.

Zeile 9 ist gegenteilig konditioniert, sie gilt, wenn Bz.50 **ein**geschaltet ist. Somit schliessen sich diese beiden Zeilen gegenseitig aus, es kann nur die eine oder andere Zeile angezeigt werden. Daher ist es auch möglich, dass beide Texte an der gleichen Position in Zeile 2 ab Spalte 37 auf dem Bildschirm definiert sind, sie können sich ja nie überschreiben. Alles klar? :-)

Somit ist auch die Bz.50 definiert und kann später im Programm verwendet werden. Auch diese Bezugszahl sollte tunlichst dokumentiert werden, es gibt Programmierer, die besonders in Bildschirm- und Druckerdateien Bezugszahlen in rauhen Mengen verwenden und nachher nicht mehr wissen, wann was überhaupt angezeigt oder gedruckt wird. Oder: Warum das Programm überhaupt noch funktioniert ;-)

Und hier die Änderungen im Fusszeilenbereich:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0022.00      A                                      22  2'F3=Ende'
0023.00      A                                          COLOR(BLU)
0024.00      A N50                                  22 11'F6=Ändern'
0025.00      A  50                                  22 11'F6=Eingabe'
0026.00      A                                          COLOR(BLU)
0027.00      A                                      22 23'F12=Abbruch'
0028.00      A                                          COLOR(BLU)
```

Änderungen in unserer Bildschirmdatei, Teil 2

Neben der Ausgabe des Textes für F4 habe ich wie oben die Beschreibung für F6 konditioniert.

Bitte wandele die Bildschirmdatei um, es dürfen selbstverständlich keine Fehler auftreten :)

Nun kommen einige Änderungen in unserem Programm, da sich die Logik natürlich erweitert hat, und wir das Ganze gleich in einer menschenwürdigen, äh, menschenlesbaren Form unterbringen wollen.

Volles Programm

Ich drucke hier nun das gesamte Programm inklusive aller Änderungen ab. Wenn Du auf meiner Maschine arbeitest, musst Du es nicht abtippen, sondern kannst es aus der Beispielbibliothek[I] kopieren.

```
 FMT H  HSchlüsselwörter++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ***************** Datenanfang ***********************************************
0001.00 H*   *** Mein erstes interaktives Programm ***
0002.00 H*-------------------------------------------------------------
0003.00 H* Bezugszahlen: (Nr - *OFF / *ON)
0004.00 H* 12 - Funktionstaste 12
0005.00 H* 50 - Updatemodus aus / an
0006.00 H*       gleichzeitig Steuerung in Bildschirmdatei KONTOBS
0007.00 H* 70 - Schreiboperation erfolgreich / fehlerhaft
0008.00 H* 71 - Leseoperation erfolgreich / fehlerhaft
```

Die H-Spezifikationen

Hier die H-Spezifikationen (Header). Schmerzlos, besteht nur aus Kommentaren. Die Bezugszahlen, die man verwendet, sollten schön brav dokumentiert werden :-)

[I] Siehe im Anhang auf Seite 219

Nun die F-Spezifikationen (Files). Hier die bereits bekannten Definitionen der Dateien

```
 FMT F  FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter+++++++++++++++++++
0010.00  *
0011.00  *-----------------------------------------------------------------
0012.00  * Dateidefinitionen
0013.00 FKONTOPF   UF A E           K DISK
0014.00 FKONTOBS   CF   E             WORKSTN
```

Die F-Spezifikationen

Nun folgen die D-Spezifikationen (Definitions), in der die Variablen definiert werden. Hier definiere ich zwei Arbeitsfelder. Deren Namen beginne ich aus Gewohnheit immer mit # so dass ich gleich im Quellcode erkenne, dass die Variable nicht zu einer Datei gehört, sondern nur im Programm verwendet wird. Die zweite Variable #NUMTXT hat den Zweck der Typumwandlung. Die Kontonummer ist numerisch, später geben wir die Meldung aus, dass das Konto xy geladen wurde. Hier wird die Kontonummer erst in Text umgewandelt (in diese Variable), und dann ausgegeben.

```
 FMT D  DName++++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter+++++++++++++++++++
0016.00  *-----------------------------------------------------------------
0017.00  * Variablen
0018.00 D#EINOK           S              1A   INZ(*OFF)
0019.00 D#NUMTXT          S              6A   INZ(*BLANKS)
```

Die D-Spezifikationen

Nun geht es mit den C-Spezifikationen (Calculations oder Commands) weiter:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0021.00  *-----------------------------------------------------------------
0022.00  * geklonte Variable
0023.00 C     *LIKE         DEFINE    B1KTONUM      #KTONUM
0024.00  *-----------------------------------------------------------------
0025.00  * Jetzt gehts los!
0026.00  *-----------------------------------------------------------------
0027.00
0028.00  * Initialisierung
0029.00 C                   EVAL      *IN03=*OFF
0030.00 C                   EVAL      *IN06=*OFF
0031.00 C                   EVAL      *IN12=*OFF
0032.00 C                   EVAL      *IN50=*OFF
0033.00 C                   EVAL      #KTONUM=0
```

Die C-Spezifikationen am Anfang

Zuerst klone ich mir eine Variable. Mit dem `*LIKE DEFINE` – Befehl erstelle ich eine neue Variable (die im Ergebnis angegeben ist). Die neue Variable istgenauso definiert wird wie die Variable im Faktor2. Hier nutze ich die Variable #KTONUM für Zwischenberechnungen.

Danach werden die verwendeten Bezugszahlen und Hilfsvariablen geleert, sofern es nicht schon in den D-Spezifikationen geschehen ist (siehe `INZ()`-Parameter).

Zur Hauptroutine:

```
 FMT F  FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0035.00  * Hauptroutine
0036.00 C                   DOW       *IN12=*OFF
0037.00 C                             AND *IN03=*OFF
0038.00 C                   EXFMT     DSPKTO1
0039.00
0040.00  * Bei F6 zwischen Ändern/Einfügen wechseln
0041.00 C                   IF        *IN06=*ON
0042.00 C                   EVAL      *IN50=NOT(*IN50)
0043.00 C                   ENDIF
```

Beginn der Hauptroutine

Hier der Anfang der Hauptroutine. Diese läuft wie bekannt in einer DO-Schleife. Nach der Aus/Eingabe vom Bildschirmformat DSPKTO1 prüfen wir zunächst, ob F6 gedrückt wurde. In diesem Falle wechseln wir zwischen Eingabe- und Änderungsmodus. Dafür negieren wir einfach den Wert von *IN50. Nicht an ist ja aus, und nicht aus ist an :-) Somit kann man Bezugszahlen einfach und elegant nutzen, siehe unsere Bildschirmdefinition.

Nun der zweite Teil der Hauptroutine:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0045.00  * Änderungsmodus? entsprechend prüfen.
0046.00 C                   IF        *IN03=*OFF
0047.00 C                             AND *IN06=*OFF
0048.00 C                             AND *IN12=*OFF
0049.00 C                             AND *IN50=*ON
0050.00 C                   EXSR      CHKNUPD
0051.00 C                   ENDIF
0052.00
0053.00  * Eingabemodus? Entsprechend prüfen.
0054.00 C                   IF        *IN03=*OFF
0055.00 C                             AND *IN06=*OFF
0056.00 C                             AND *IN12=*OFF
0057.00 C                             AND *IN50=*OFF
0058.00 C                   EXSR      CHKNWRT
0059.00 C                   ENDIF
0060.00 C                   ENDDO
0061.00
0062.00  * Anzeiger "letzter Satz" ansetzen und raus aus dem Programm!
0063.00 C                   EVAL      *INLR=*ON
```

Der Rest des Hauptteils

Nun wird für Eingabe- und Änderungsmodus geprüft. Es darf jeweils keine F-Taste gedrückt sein. Abhängig von *IN50 wird eine Unterroutine aufgerufen. Diese werden gleich beschrieben.

Unterhalb der Routine?

Hier also die einzelnen Unterroutinen. Diese dürfen natürlich in beliebiger Reihenfolge im Quellcode stehen, aber müssen nach der Hauptroutine kommen. Ausserhalb von BEGSR und ENDSR dürfen nach der ersten Routine keine Befehle vorkommen.

Die Unterroutinen:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0066.00  *---------------------------------------------------------------------
0067.00  *! Prüft im Eingabemodus ob Daten vollständig sind
0068.00  *---------------------------------------------------------------------
0069.00 C     CHECK01       BEGSR
0070.00 C                   IF        B1KTONUM > 0 AND B1KTOTXT <> *BLANKS
0071.00 C                   EVAL      #EINOK = *ON
0072.00 C                   ELSE
0073.00 C                   EVAL      #EINOK = *OFF
0074.00 C                   ENDIF
0075.00 C                   ENDSR
```

Die Routine „CHECK01“

Bereits bekannt und auch nicht sehr schwierig, oder?

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0077.00  *---------------------------------------------------------------------
0078.00  *! Prüfungen im Updatemodus
0079.00  *---------------------------------------------------------------------
0080.00 C     CHKNUPD       BEGSR
0085.00 C                   IF        B1KTONUM <> #KTONUM
0086.00 C                   EXSR      GETREC
0087.00 C                   ELSE
0088.00 C                   EXSR      UPDREC
0089.00 C                   ENDIF
0090.00 C                   ENDSR
```

Die Routine CHKNUPD

So, auch hier kann man nicht viel falsch machen. Wenn das Eingabefeld B1KTONUM geändert wurde (dafür wird später der Inhalt in das Zwischenfeld #KTONUM kopiert), dann gehen wir davon aus, dass der Anwender einen anderen Datensatz laden will. Ansonsten gehen wir davon aus, dass er den geladenen Satz wieder wegschreiben will.

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0182.00 C     KEYLISTEN     BEGSR
0183.00 C     *LIKE         DEFINE    KTONUM         kKTONUM
0184.00 C     kyKONTOPF     KLIST
0185.00 C                   KFLD                     kKTONUM
0186.00 C                   ENDSR
```

Die Keyliste(n)-Definition

Schnell ein Einschub. Hier die Routine KEYLISTEN. Die Befehle könnten auch am Anfang der C-Spezifikationen stehen, aber aus optischen Gründen schiebe ich die gerne in eine eigene Routine. Diese muss nicht aufgerufen werden, der Compiler erkennt die Befehle und definiert die Keylisten.

Zunächst klone ich mir noch das Feld KTONUM nach kKTONUM. Dann definiere ich die Schlüsselliste kyKONTOPF, und darin das Schlüsselfeld kKTONUM.

Wenn man mit einem Schlüssel auf eine Datei zugreifen will (und erst dadurch wird eine Datenbank so richtig flott), muss man definieren, welche Werte man sucht. Die SQL-Profis kennen die Abfragen mit `SELECT`. In RPG kann man mit SQL oder mit den RPG-Eigenen Befehlen arbeiten. Intern verwendet die AS/400 für beide Varianten fast die gleichen Routinen, gelegentlich ist SQL flexibler, andererseits sind die RPG-Befehle etwas optimierter, wenn es um den direkten Zugriff auf einen einzelnen Satz geht. Für viele ist die Wahl zwischen SQL-Zugriff oder RPG-Zugriff auch eine Glaubensfrage :)

Wir müssen also in RPG zuerst den Schlüssel definieren, mit dem wir auf die Datei zugreifen wollen. Die Felder im Schlüssel sollten mit dem Schlüssel, der in der Datei definiert ist, übereinstimmen. Ansonsten baut sich das Betriebssytem im Notfall selbst einen Index, was natürlich Zeit raubend ist. Da unsere Kontodatei nur ein Schlüsselfeld hat, ist unser Schlüssel natürlich sehr kurz :-)

Warum ich den ersten Buchstaben des Schlüsselnamens klein schreibe? Reine Optik :-)

So, auf zur nächsten Routine: (bitte zur nächsten Seite blättern)

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0092.00  *-----------------------------------------------------------------
0093.00  *! Liest angegebenen Satz zum Aktualisieren
0094.00  *-----------------------------------------------------------------
0095.00 C     GETREC        BEGSR
0096.00 C                   EVAL      kKTONUM = B1KTONUM
0097.00 C     kyKONTOPF     CHAIN     FMTKTO                             71
0098.00 C                   IF        *IN71 = *OFF
0099.00 C                   EVAL      B1KTONUM = KTONUM
0100.00 C                   EVAL      B1KTOTXT = KTOTXT
0101.00 C                   EVAL      B1KTODAT = KTODAT
0102.00  * numerisches Feld in Text einfügen.

0102.01  * entweder Zeilen 103-106 (Systeme bis Release V4R1M0):
0103.00 C                   MOVE      B1KTONUM      #NUMTXT
0104.00 C                   EVAL      B1MELDUNG = 'Satz '+
0105.00 C                             #NUMTXT +
0106.00 C                             ' wurde erfolgreich geladen.'
0107.00  * - - - - - - - - - - -
0108.00  * Alternative ab V4R2:
0109.00  *                  EVAL      B1MELDUNG = 'Satz '+
0110.00  *                            %CHAR(B1KTONUM) +
0111.00  *                            ' wurde erfolgreich geladen.'
0112.00  * - - - - - - - - - - -
0113.00 C                   ELSE
0114.00  * Den Satz gibt es nicht, entsprechend vermelden!
0115.00 C                   EVAL      B1KTOTXT = *BLANKS
0116.00 C                   EVAL      B1KTODAT = 0
0117.00 C                   EVAL      B1MELDUNG = 'FEHLER: Satz wurde +
0118.00 C                             nicht gefunden!'
0119.00 C                   ENDIF
0120.00 C                   EVAL      #KTONUM = B1KTONUM
0121.00 C                   ENDSR
```

Die Routine GETREC

Hier ist schon etwas mehr los in der Routine. Wir wollen den Satz einlesen, den der Nutzer in `B1KTONUM` bestimmt hat. Hier kommen nun die oben vorgestellten Schlüssellisten zum Zuge. Ich schiebe die gesuchte Kontonummer in unser Zwischenfeld `kKTONUM`, welches ja Bestandteil unseres Schlüssels ist. Ich könnte auch in der Schlüsseldefinition direkt das Feld `B1KTONUM` anstelle `kKTONUM` verwenden, allerdings habe ich dann keine Kontrolle mehr über den Inhalt.

Bitte beachte die Zeilen `102.01` bis `112.00`: Bei Betriebssystemen bis V4R1 war die Funktion `%CHAR()` im RPG nicht eingebaut, Du musst also die Zeilen `103.00` bis `106.00` verwenden (inklusive Definition von `#NUMTXT` am Programmanfang). Ansonsten kannst Du die Zeilen `107.00` bis `112.00` verwenden.

Gut, nach dem wir den Schlüssel, mit dem wir nach Daten suchen wollen, belegt haben, schicken wir die AS/400 auf die Suche. Der Befehl CHAIN sucht in einer angegebenen Datei oder dem angegebenen Format nach dem in Faktor1 definierten Schlüssel und liest im Erfolgsfall den ersten gefundenen Satz ein. Da unsere Datei UNIQUE, also eindeutig definiert ist, haben wir auf jeden Fall den gefundenen Datensatz. Kann CHAIN keinen dem Schlüssel entsprechenen Satz finden, wird die Bezugszahl in „Ho“, hier also *IN71, auf *ON gesetzt. „Ho“ch bedeutet hier „nur höhere Sätze in der Sortierung gefunden“. Also auf jeden Fall nicht das, was wir haben wollten. Wenn *IN71 also nach dem CHAIN auf *OFF steht, können wir davon ausgehen, dass wir den vom Nutzer gesuchten Satz gefunden haben, und schieben die Datenfelder in unsere Bildschirmfelder. Ansonsten können wir davon ausgehen, dass der Datensatz nicht existiert und behandeln die Felder entsprechend.
Diese Suche mit CHAIN und einem (richtig) definierten Schlüssel ist sehr effektiv und sehr schnell. Später werden wir lernen, wie man verschiedene Schlüssel auf eine Datei definiert und so auch innerhalb hunderter Millionen Datensätze in kurzer Zeit (also ein paar Millisekunden) den gesuchten Datensatz findet. Jede Datenbank steht und fällt mit den Schlüsseln und dem richtigen Zugriff. Somit sind auf größeren Hobeln Datenselektionen mit Bearbeitungsgeschwindigkeiten von über einer Millionen Sätzen pro Sekunde ein Kinderspiel. Das ist natürlich nicht nur auf der AS/400 so.

Bewegung ins Spiel!

In dieser Routine findest Du noch einen komisch aussehenden Befehl namens MOVE. Dieser ist nicht so ganz ohne. Wir wollen ja im Erfolgsfall dem Nutzer mitteilen, dass wir den Satz geladen haben. Hierzu muss der numerische Wert der Kontonummer zusammen mit Zeichenwerten ausgegeben werden. Also muss der numerische Wert in einen Zeichenwert umgewandelt werden. Auf älteren Systemen mit älteren Systemversionen geht das nicht über eine Funktion, die man von anderen Sprachen her kennt. Siehe das Beispiel im Kommentar, ab V4R2 gibt es die Funktion %CHAR, die das viel anschaulicher erledigt. Auf den älteren Systemen nutze ich MOVE. Wenn Du das Beispiel auf meinem System nachvollziehen willst, kannst Du die Variante mit %CHAR nehmen.

MOVE macht das, was der Name sagt. Es schiebt den Wert von einer Variablen in eine andere. Das klingt trivial, damit kann man sich aber so manches graue Haar einfangen. Einen Exkurs zu MOVE bringe ich später. Hier schiebe ich einfach einen numerischen in einen Zeichenwert. Da beide gleich lang definiert sind (schau nach), ist es kein Problem.

Mehr muss man zu dieser Routine jetzt wohl erst einmal nicht sagen, oder? Dann gleich weiter zur nächsten Routine:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0123.00  *---------------------------------------------------------------
0124.00  *! Aktualisiert den vorher eingegebenen Satz
0125.00  *---------------------------------------------------------------
0126.00 C     UPDREC        BEGSR
0127.00 C                   IF        B1KTOTXT <> *BLANKS
0128.00 C                             AND B1KTODAT <> 0
0129.00  * Arbeitsfelder in Datensatz schieben
0130.00 C                   EVAL      kKTONUM = B1KTONUM
0131.00 C                   EVAL      KTOTXT = B1KTOTXT
0132.00 C                   EVAL      KTONUM = B1KTONUM
0133.00 C                   EVAL      KTODAT = B1KTODAT
0134.00  * Aktualisieren
0135.00 C                   UPDATE    FMTKTO
0136.00  * Felder initialisieren
0137.00 C                   EVAL      B1MELDUNG = 'Satz wurde aktualisiert.'
0138.00 C                   EVAL      B1KTONUM = 0
0139.00 C                   EVAL      B1KTOTXT = *BLANKS
0140.00 C                   EVAL      B1KTODAT = *ZEROS
0141.00 C                   EVAL      #KTONUM = 0
0142.00 C                   ELSE
0143.00  * Sonst: Wenn nicht vollständig
0144.00 C                   EVAL      B1MELDUNG = 'FEHLER: Bitte alle +
0145.00 C                             Felder ausfüllen!'
0146.00 C                   ENDIF
0147.00 C                   ENDSR
```

Die Routine UPDREC

Wie Du Dich bestimmt erinnerst, wird diese Routine aufgerufen, wenn der Anwender Daten auf dem Bildschirm hat, Datenfreigabe drückt, aber die Kontonummer nicht verändert hat. Somit gehen wir davon aus, dass er alte Daten aktualisieren will. Dies erledigt die Routine `UPDREC`.
Zunächst wird noch einmal geprüft, ob die Daten korrekt sind (also alle nötigen Felder gefüllt). Wenn ja, werden die Bildschirmfelder in die Datensatzfelder geschoben und mit `UPDATE` der Satz weggeschrieben. Danach wird das Meldungsfeld belegt und alle anderen Felder (auch das Zwischenfeld für das Erkennen der Kontonummeränderung) geleert. Sollten die Daten nicht vollständig sein, gibt es eine entsprechende Meldung.

So, viel gibt es hier auch nicht zu sagen. Einfach erst mal diese Routine anschauen und verinnerlichen. Zu den interessanten Schweinereien kommen wir gleich.

Hier nun die Schreibroutine:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0149.00  *------------------------------------------------------------------
0150.00  *! schreibt im Einfügemodus den neuen Satz
0151.00  *------------------------------------------------------------------
0152.00 C     WRITE01       BEGSR
0153.00  * Erst mal die Arbeitsfelder den Datensatzfeldern zuweisen
0154.00 C                    EVAL      KTONUM = B1KTONUM
0155.00 C                    EVAL      KTOTXT = B1KTOTXT
0156.00 C                    EVAL      KTODAT = B1KTODAT
0157.00  * Satz schreiben. Im Erfolgsfall (*IN70=*OFF), Eingabefelder
0158.00  * löschen und Meldung geben. Ansonst Felder belassen und
0159.00  * Fehler ausgeben
0160.00 C                    WRITE     FMTKTO                                70
0161.00 C                    IF        *IN70 = *OFF
0162.00 C                    EVAL      B1KTONUM=0
0163.00 C                    EVAL      B1KTOTXT=*BLANKS
0164.00 C                    EVAL      B1KTODAT=0
0165.00 C                    EVAL      B1MELDUNG = 'Daten wurden geschrieben.'
0166.00 C                    ELSE
0167.00 C                    EVAL      B1MELDUNG = 'FEHLER: Satz nicht +
0168.00 C                              geschrieben!'
0169.00 C                    ENDIF
0170.00 C                    ENDSR
```

Die Routine WRITE

Auch hier ist eigentlich für Dich inzwischen alles alter Käse. Die Routine wird aufgerufen, wenn ein neuer Satz geschrieben wird. Zunächst einmal werden wieder alle Bildschirmfelder in die Datensatzfelder geschoben (das kommt häufiger im Programm vor, da könnte man auch eine eigene Routine für erstellen). Danach gibt's ein schmerzloses `WRITE`. Die verwendete Bezugszahl `*IN70` wird auf `*ON` gesetzt, wenn hierbei etwas schiefgegangen ist (zum Beispiel, weil ein Satz mit dieser Nummer schon existiert). Abhängig von diesem Ergebnis gibt's entsprechende Meldungen, bei Erfolg werden die Eingabefelder geleert.
Das dürfte einleuchtend sein.

Hier noch der Vollständigkeit halber die Routine, die das Ergebnis von `#EINOK` prüft:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0172.00  *------------------------------------------------------------------
0173.00  * Im Eingabemodus: Fehlermeldung ausgeben oder Schreiben
0174.00  *------------------------------------------------------------------
0175.00 C     CHKNWRT       BEGSR
0176.00 C                    EXSR      CHECK01
0177.00 C                    IF        #EINOK = *OFF
0178.00 C                    EVAL      B1MELDUNG = 'Bitte Kontonummer UND +
0179.00 C                              Kontobezeichnung eingeben!'
0180.00 C                    ELSE
0181.00 C                    EXSR      WRITE01
0182.00 C                    ENDIF
0183.00 C                    ENDSR
```

Die Routine CHKNWRT

Keine Beschwerden! Das hätte man auch etwas anders strukturieren können. Aber erstens sind wir ja noch am Anfang, und zweitens kannst Du das auch ruhig selbst machen :-)

So, last but not least kommt noch das wichtigste, unsere Schlüsseldefinitionen. Die setze ich aus Gewohnheit immer in eine Routine KEYLISTEN an das Ende der C-Bestimmungen. Diese Routine musst Du nicht aufrufen, der Compiler erkennt die darin enthaltenen Befehle von selbst und bearbeitet sie.

```
FMT C   CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0184.00  *---------------------------------------------------------------
0185.00  * Hier definieren wir unsere Keylisten (oder Schlüssellisten)
0186.00  * Diese Routine wird nie aufgerufen, sie dient nur dazu,
0187.00  * die Definitionen ordenlich in eine Ecke zu packen, sie können
0188.00  * überall im Quellcode im C-Bereich stehen.
0189.00  *---------------------------------------------------------------
0190.00 C     KEYLISTEN     BEGSR
0191.00 C     *LIKE         DEFINE    KTONUM        kKTONUM
0192.00 C     kyKONTOPF     KLIST
0193.00 C                   KFLD                    kKTONUM
0194.00 C                   ENDSR
        ******************Datenende ********************************************
```

Unsere Keylisten

Soderle, wir haben es geschafft! Ein eigenes kleines Programm zum Anlegen und Ändern von Datensätzen. Aber... hm, da war doch noch was. Jaaah, wie stehts mit Löschen? Kein Problem, der Befehl heisst DELETE, also mach mal. ;-)

Löschen!

Ok, war ein Spaß. Löschen ist aber gar nicht so schwer, wenn man genau weiss, was man löschen will. Nehmen wir an, wir wollen dem Benutzer erlauben, den aktuellen Satz, den er im Änderungsmodus auf dem Bildschirm geladen hat, mit der Taste F11 zu löschen. (Hat er keinen Satz geladen, können wir den aktuellen Satz auch nicht löschen, klaro?)

Dazu müssen wir zunächst in unserer Bildschirmdatei dem Benutzer erlauben, diese Funktionstaste überhaupt zu verwenden. Dies geht am Einfachsten, wenn Du folgende Zeile in Deine Bildschirmdatei einträgst (natürlich an passender Stelle!):

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0005.01      A                                      CF11(1 'Löschen')
```

Zusätzlich die Taste F11 erlauben

Nun springe bitte an das Ende der Quelle für die Bildschirmdatei. Bei den F-Tasten werden wir nun ein paar optische Änderungen mit einem Trick vornehmen. Die Taste F11 soll nur im Bildschirmfuss dokumentiert sein, wenn sie auch verwendet werden kann, also im Änderungsmodus. Dazu blenden wir den Text für die F11-Taste nur ein, wenn F6 als „Eingabe“ bezeichnet wird (wir uns also im Änderungsmodus befinden, siehe Programmlogik). Da die F-Tasten aber in numerischer Reihenfolge angezeigt werden sollen, muss dafür der Text für die Taste F12 verschoben werden. Nehmen wir hier eine für unsere Zwecke in diesem Beispielprogramm brauchbare Logik: Da die Anzeige des F6-Textes von der Bezugszahl 50 abhängt, können wir damit auch die Anzeige vom F11-Text und die Position vom F12-Text mit dieser Bezugszahl steuern. Siehe die nötigen Änderungen auf der nächsten Seite.

```
 Spalten . . . :    1  71          Editieren                LIBMANTEST/QDDSSRC
 SEU==>                                                          KONTOBS
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0022.00      A                                    22  2'F3=Ende'
0023.00      A                                        COLOR(BLU)
0024.00      A N50                                22 11'F6=Ändern'
0024.01      A                                        COLOR(BLU)
0025.00      A  50                                22 11'F6=Eingabe'
0026.00      A                                        COLOR(BLU)
0026.01      A  50                                22 23'F11=Löschen'
0026.02      A                                        COLOR(BLU)
0027.00      A N50                                22 23'F12=Abbruch'
0027.01      A                                        COLOR(BLU)
0027.02      A  50                                22 36'F12=Abbruch'
0028.00      A                                        COLOR(BLU)
0029.00      A            B1MELDUNG     60A  O 24  5COLOR(RED)
     ******************Datenende ***************************************************
```

So steuern wir den Bildschirm elegant.

Wenn also *IN50 auf *ON steht, wir also im Änderungsmodus sind, wird:
1. der Text für F6=Eingabe angezeigt.
2. daneben der Text für F11=Löschen angezeigt.
3. Der Text für F12=Abbruch erst ab Spalte 36 angezeigt.

Ansonsten, wenn *IN50 auf *OFF steht (Eingabemodus) passiert dies:
1. der Text für F6=Ändern wird angezeigt.
2. Der Text für F12=Abbruch wird ab Spalte 23 angezeigt.
3. Es wird kein Text für F11=Löschen angezeigt.

Der Text für F3=Ende wird immer angezeigt.

Soweit verständlich? Ok, ich denke, doch. Bitte also diese Änderungen in die Quelle KONTOBS eintragen und die Bildschirmdatei mit Auswahl 14 neu erstellen.

Nun geht's an die Änderungen im RPG-Programm. Die Taste F11 sollte vor allen anderen F-Tasten abgefragt werden. Hintergrund: In unserer jetzigen Logik müssen wir aufpassen, dass wir uns mit den Funktionstasten nicht zu sehr verzetteln, später werden wir das übersichtlicher gestalten. Ausserdem ist die gedrückte Taste F11 als Fehler zu behandeln, wenn wir uns im Eingabemodus befinden; somit sollte im Eingabemodus keine weitere Prüfung oder Verarbeitung statt finden.
Suche Dir also im Quelltext die Stelle, nach der F6 geprüft wurde; Du kannst die folgende Routine auch davor einbauen.

Prüfen wir nun also auf die F11 und handeln enstprechend:

```
 Spalten . . . :    6  76          Editieren                LIBMANTEST/QRPGLESRC
 SEU==>                                                          EDTKONTO
 FMT CX CL0N01Factor1+++++++Opcode&ExtErweiterter-Faktor2++++++++++++++++++++++
0044.01  * Taste F11 gedrückt? Nur erlaubt im Änderungsmodus, dann
0044.02  * den Satz löschen. Im Eingabemodus eine Meldung bringen.
0044.03 C                   IF        *IN11=*ON
0044.04 C                   IF        *IN50=*OFF
0044.05 C                   EVAL      B1MELDUNG = 'Taste F11 ist im +
0044.06 C                             Eingabemodus nicht erlaubt!'
0044.07 C                   ELSE
```

Beginn der F11-Prüfung

So, zunächst klammern wir alle folgenden Befehle ein, dass sie nur bearbeitet werden, wenn F11 gedrückt wurde.

Danach gibt es zwei Möglichkeiten: *IN50 ist *OFF oder *ON. Im ersten Fall läuft das Programm im Eingabemodus, also ist die Verwendung von F11 nicht erlaubt. Wenn dem so ist, geben wir eine Meldung in unserer Meldungszeile aus und fertig.

Wenn wir im Änderungsmodus sind, müssen wir auf eine weitere Stolperfalle aufpassen: Es könnte ja sein, dass der Benutzer im Änderungsmodus eine Kontonummer eingegeben hat, die es gar nicht gibt. Somit wurde kein Satz geladen und wir können nicht einfach den aktuellen Satz löschen. Also brauchts noch eine IF-Abfrage[1]. Testen wir am besten auf den Kontotext. Dieser muss ja belegt sein, wenn ein Satz geladen wurde, da er ja auch belegt sein musste, wenn ein Satz geschrieben wurde. (Okok, ist vielleicht etwas blauäugig, in unserem Testprogramm geht's, und später solltest Du halt auch ein Auge darauf haben, ob Dateioperationen Erfolg hatten :-)

Nach dem ELSE-Befehl geht es wie folgt weiter:

```
 Spalten . . . :    6  76          Editieren               LIBMANTEST/QRPGLESRC
 SEU==>                                                            EDTKONTO
 FMT CX CL0N01Faktor1+++++++Opcode&ExtErweiterter-Faktor2+++++++++++++++++++++
0044.08 C                   IF        B1KTOTXT <> *BLANKS
0044.09 C                   DELETE    FMTKTO
0044.10 C                   EVAL      B1MELDUNG = 'Satz wurde gelöscht!'
0044.11 C                   EVAL      *IN50=*OFF
0044.12 C                   EVAL      B1KTONUM = *ZEROS
0044.13 C                   EVAL      B1KTOTXT = *BLANKS
0044.14 C                   EVAL      B1KTODAT = *ZEROS
0044.15 C                   ELSE
0044.16 C                   EVAL      B1MELDUNG = 'Du hast keinen Satz +
0044.17 C                             geladen, den ich löschen könnte!'
0044.18 C                   ENDIF
0044.19 C                   ENDIF
0044.20 C                   ENDIF
0044.21
```

Der Rest unserer Routine

Die erste IF-Abfrage nach dem ELSE-Zweig (von *IN50) prüft, ob wir auch wirklich einen Satz auf dem Bildschirm haben. Wenn ja, wird

- der aktuelle Satz mit DELETE gelöscht,
- eine Meldung ausgegeben,
- *IN50 auf *OFF gesetzt und somit das Programm in den Eingabemodus gesetzt sowie
- die Eingabefelder geleert.

Ansonsten gibt es nur einen Hinweis, dass wir gar keinen Satz löschen können.

Die drei folgenden ENDIF zeigen, dass man die Anzahl der verschachtelten IF-Abfragen im Auge behalten sollte. Besonders wenn ein Programme lange gepflegt wird und immer neue Abfragen dazu kommen, wird es hier sehr schnell unübersichtlich. Besser wäre es, die gesamte Routine für F11 in eine Unterroutine zu setzen.

Gut, nun hast Du ein komplettes Pflegeprogramm, das man als Grundgerüst für spätere Aktivitäten nutzen kann.

Die Quellen für dieses Beispiel findest Du in der Bibliothek LIBMANTEST in der Quellendatei QEDTKTO!

[1]Du merkst, man kann schnell eine Menge verschachtelter IF's haben, sie Ende dieser Routine. In einem größeren Projekt sollte man das ganze etwas strukturierter und mit Routinen machen, sonst hat man nur noch ENDIF Zeilen untereinander, und keiner blickt mehr durch.

Feinheiten interaktiver Programme

Auf besondere Nachfrage kommen jetzt noch ein paar Beispiele für interaktive Programme, insbesondere Details zur Programmierung und Attribute der vielfältigen Möglichkeiten, die Du hast.

Mehrere Formate

Oft kommt es vor, dass Du einen Bildschirm darstellen willst, der aus mehreren Teilen besteht. Viele Programmierer erstellen einen Bildschirm aus:

- Überschrift (Name des Programms, Titel, aktuelle Zeit etc.)
- Hauptteil (mit Details, Datensätze etc)
- Fussteil (Funktionstasten, Nachrichten etc)

Um dies einfach zu bewerkstelligen, kannst Du etwa so vorgehen:
Definiere in Deiner Bildschirmdatei *mehrere* Satzformate:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen++++++++++++++++++
        ***************** Datenanfang ****************************************
0001.00       A           R FMTKOPF
0001.01       A             KOPFUNAM      10A  O  1  2TEXT('Benutzername')
0001.02       A                                       COLOR(BLU)
0002.00       A             KOPFJNAM      10A  O  2  2TEXT('Bildschirmname')
0003.00       A                                       COLOR(BLU)
0004.00       A                                    1 73DATE
0005.00       A                                       COLOR(BLU)
0006.00       A                                       EDTCDE(Y)
0007.00       A                                    2 73TIME
0008.00       A                                       COLOR(BLU)
0009.00       A             KOPFTITL      55A  O  1 13TEXT('Überschrift')
0010.00       A                                       COLOR(WHT)
0011.00
```

Das Format für den Bildschirmkopf (INTERAKT01)

Hier definieren wir uns das Format `FMTKOPF`, das als Bildschirmüberschrift verwendet wird. Zunächst kommen in Zeile 1 und 2 links der Benutzer- und Bildschirmname (selbstverständlich musst Du die hier als Beispiel angeführten Variablennamen in Deinem Programm belegen, bevor Du das Format ausgibst). Dann folgt in Zeile 1 ab Spalte 73 das aktuelle Systemdatum durch das Schlüsselwort `DATE` (Benutze die Hilfefunktion um mehr über dieses Schlüsselwort herauszufinden). Die Angabe `EDTCDE(Y)` bedeutet, dass das Datum im Format `TT.MM.JJ` ausgegeben wird. Darunter zeigt das Schlüsselwort `TIME` die aktuelle Systemzeit an. Als letztes kommt eine Überschrift aus der Variablen `KOPFTITL`, die weiss angezeigt wird.

Nun definieren wir die Fusszeilen (es ist dem System egal, in welcher Reihenfolge die Bildschirmformate in der DSPF-Teildatei eingetragen sind.

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0014.00      A          R FMTFUSS
0015.00      A                                       OVERLAY
0016.00      A            FUSSFKT1      78A  O 22  1TEXT('Funktionstasten 1')
0017.00      A                                       COLOR(BLU)
0018.00      A            FUSSFKT2      78A  O 23  1TEXT('Funktionstasten 1')
0019.00      A                                       COLOR(BLU)
0020.00      A            FUSSTXT       78A  O 24  1TEXT('Nachrichtenzeile')
0021.00      A                                       COLOR(RED)
```

Hier die Angaben zum Fussteil des Bildschirms (INTERAKT01)

Das einzig neue ist hier die Funktion OVERLAY. Diese besagt, dass das Format ausgegeben werden soll, ohne dass bereits auf dem Bildschirm dargestellte Formate gelöscht werden sollen.

Letztlich fehlt uns nur noch der Mittelteil, auch Detailbildschirm genannt. Hier ein ganz banaler:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0023.00      A          R FMTDETAIL
0024.00      A                                       CF03(03 'Ende')
0025.00      A                                       OVERLAY
0026.00      A                                    5  2'Dein Name:'
0027.00      A              #NAME          20A  B  5 15CHECK(LC)
     ******************Datenende ****************************************
```

Unser Hauptteil des Bildschirms

Als allererstes wird definiert, dass die Funktionstaste F3 verwendet werden darf (*CF03*). Wurde sie gedrückt, wird die Bezugszahl 03 (in den Klammern definiert) auf *ON gesetzt. Der Text in den Parameterklammern ist nur zur Dokumentation gut. Du könntest auch *CF03(43)* schreiben, dann wird die Bezugszahl 43 bei Verwendung der Funktionstaste 3 eingeschaltet. Dies ist aber ein schlechter Stil, man sollte aus dem Namen der Bezugszahl auch gleich die zugehörige Taste erkennen können! Ach ja: CFxx bedeutet, dass bei Verwenden einer Funktionstaste (wozu auch die Blätterntasten gehören) die aktuellen Eingaben in die Eingabefelder geschrieben und dem Programm mitgeteilt werden. Willst Du zum Beispiel, dass bei F12 die Eingaben nicht in die Variablen geschrieben werden, schreibe hier ***CA12***.

Auch dieses Format wird per OVERLAY wieder geschrieben, ohne vorhandene Formate auf dem Bildschirm zu löschen. Dann kommt eine Textkonstante an Zeile 5, Spalte 2, sowie ein Eingabefeld. Die Daten werden in die Variable #NAME (20 Zeichen alpha) geschrieben, die Feldart B gibt „***b**oth*“ an, also Ein- und Ausgabe. Hier wird oft ein Fehler gemacht, dass ein Eingabefeld als I definiert wird (***I**nput*). Ist zwar nicht falsch, nur wird der eingegebene Wert aus der Variablen nicht angezeigt, wenn das Satzformat erneut z.B. innerhalb einer Bearbeitungsschleife neu angezeigt wird.
Die Funktion CHECK(LC) sagt dem System, dass in diesem Feld auch Kleinbuchstaben (***l**ower **c**ase*) erlaubt sind.

Damit unser Bildschirm auch zu etwas nütze ist, brauchen wir ein kleines Programm. Hier ein kurzes Beispiel in RPG:

```
 FMT FX FDateiname+IPEASF.....L.....A.E/AEinh.Schlüsselwörter++++++++++++++++++
      ***************** Datenanfang ****************************************
0001.00 FINTERAKT01CF   E             WORKSTN
0002.00
0003.00 C                   EVAL      KOPFUNAM = 'Benutzer'
0004.00 C                   EVAL      KOPFJNAM = 'INTERAKT01'
0005.00 C                   EVAL      KOPFTITL = 'Testprogramm Interaktiv 01'
0006.00 C                   WRITE     FMTKOPF
0007.00 C                   EVAL      FUSSFKT1 = 'F3=Ende'
0008.00 C                   EVAL      FUSSFKT2 = *blanks
0009.00 C                   EVAL      FUSSTXT  = 'Bitte gib Deinen Namen ein.'
0010.00
0011.00  *** Hauptschleife
0012.00 C                   DOW       *IN03=*OFF
0013.00 C                   WRITE     FMTFUSS
0014.00 C                   EXFMT     FMTDETAIL
0015.00 C                   IF        #NAME = *blanks
0016.00 C                   EVAL      FUSSTXT  = 'Sei nicht so faul!'
0017.00 C                   ELSE
0018.00 C                   EVAL      FUSSTXT  = 'Hallo, ' + #NAME
0019.00 C                   ENDIF
0020.00 C                   ENDDO
0021.00 C
0022.00 C                   EVAL      *INLR = *ON
      ******************Datenende ****************************************
```

Interaktives Testprogramm 01

Das Programm sollte Dir nun keine Probleme bereiten. Es belegt die Variablen, gibt den Kopf aus. In der Hauptschleife werden Fuss (für den Nachrichtentext) und Detail ausgegeben, bis F3 gedrückt wird. Anhand der Eingabe wird ein Nachrichtentext gesetzt.

Formatierte Ausgabe

Wenn Du in einer Ausgabe ein Feld formatieren willst, zum Beispiel das Datum, benutzt Du die Funktion EDTWRD. Ändere den Hauptteil deiner Bildschirmdatei wie folgt:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0023.00      A          R FMTDETAIL
0024.00      A                                        CF03(03 'Ende')
0025.00      A                                        OVERLAY
0026.00      A                                    5  2'Dein Name:'
0027.00      A            #NAME          20A  B  5 15CHECK(LC)
0028.00      A                                    6  2'Geburtstag:'
0029.00      A            #DATUM          8Y 0B  6 15
0030.00      A                                    7  2'Eingegeben:'
0031.00      A            #DATUM2         8Y 0O  7 15EDTCDE(Y)
     ******************Datenende ***************************************
```

Formatierte Ausgabe

Hier definieren wir ein Eingabefeld #DATUM sowie ein Ausgabefeld #DATUM2. Der EDTCDE besagt, dass das #DATUM2 im Format xx.xx.xxxx ausgegeben werden soll. Beide Datumsfelder sind 8stellige numerische Variablen.

Das RPG-Programm änderst Du wie folgt:

```
 FMT *   *. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+.
0011.00  *** Hauptschleife
0012.00 C                   DOW       *IN03=*OFF
0013.00 C                   WRITE     FMTFUSS
0014.00 C                   EXFMT     FMTDETAIL
0015.00 C                   IF        #NAME = *blanks
0016.00 C                   EVAL      FUSSTXT  = 'Sei nicht so faul!'
0017.00 C                   ELSE
0018.00 C                   EVAL      FUSSTXT  = 'Hallo, ' + #NAME
0019.00 C                   ENDIF
0020.00 C                   EVAL      #DATUM2 = #DATUM
0021.00 C                   ENDDO
0022.00 C
0023.00 C                   EVAL      *INLR = *ON
     ******************Datenende ***************************************
```

Interaktives Testprogramm 2

Hier wird das eingegebene Datum einfach in das auszugebende Datum verschoben. Probiere das Programm aus und gib als Datum ein: 24121995. Beachte die Ausgabe. Probiere andere Eingaben!

Gültigkeitsprüfungen

Man kann auf Feldebene Datenprüfungen durchführen, also festlegen, welche Werte oder Bereiche zum Beispiel für ein Eingabefeld erlaubt sind. Ein Beispiel: Wir erweitern unseren Detailbereich wie folgt:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0023.00      A          R FMTDETAIL
0024.00      A                                      CF03(03 'Ende')
0025.00      A                                      OVERLAY
0026.00      A                                      UNLOCK
0027.00      A                                   5  2'Dein Name:'
0028.00      A             #NAME          20A  B  5 15CHECK(LC)
0029.00      A                                   6  2'Geburtstag:'
0030.00      A             #DATUM          8Y 0B  6 15
0031.00      A                                   7  2'Eingegeben:'
0032.00      A             #DATUM2         8Y 0O  7 15EDTCDE(Y)
0033.00      A                                   8  2'Mann/Frau :'
0034.00      A             #SEX            1A  B  8 15VALUES('M' 'F')
     ******************Datenende ***************************************
```

Manche Felder können nicht jeden Wert annehmen (INTERAKT03)

Wir haben ein neues Feld #SEX für das Geschlecht erstellt und geben mit der Funktion VALUES an, welche Werte das Feld annehmen darf. In unserem RPG-Programm tun wir nun folgendes:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0011.00  *** Hauptschleife
0012.00 C                   DOW         *IN03=*OFF
0013.00 C                   WRITE       FMTFUSS
0014.00 C                   EXFMT       FMTDETAIL
0015.00 C                   IF          #NAME = *blanks
0016.00 C                   EVAL        FUSSTXT  = 'Sei nicht so faul!'
0017.00 C                   ELSE
0018.00 C                   IF          #SEX = 'M'
0019.00 C                   EVAL        FUSSTXT  = 'Hallo, lieber ' + #NAME
0020.00 C                   ELSE
0021.00 C                   EVAL        FUSSTXT  = 'Hallo, liebe ' + #NAME
0022.00 C                   ENDIF
0023.00 C                   ENDIF
0024.00 C                   EVAL        #DATUM2 = #DATUM
0025.00 C                   ENDDO
0026.00 C
0027.00 C                   EVAL        *INLR = *ON
```

Erweiterte Prüfung (INTERAKT03)

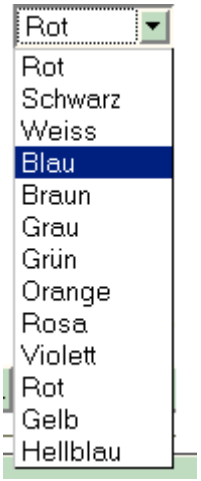
Probiers einfach mal aus. Funktioniert doch garnicht so schlecht, oder? Aaaaaber! Überlege Dir mal nun, Du schreibst ein Programm, das international verwendet werden soll. Dort hast Du einige Felder, in denen auf „Ja“ oder „Nein“ geprüft werden soll, also Inhalt „J“ oder „N“. Verklickere bitte einem Franzosen, dass er nicht „O“ (für Oui), sondern „J“ eingeben soll...

Hier kommen wir schon an die Grenzen des guten Geschmacks, sorry, an die Grenzen eines guten Datenbank-Designs. Das Thema „Mehrsprachigkeit“ wird auch noch eine eigene Stelle in diesem Manual finden, in Kürze gesagt: NIE absolut denken. Lieber einen Wert in der Datenbank haben, der je nach Situation anders dem Benutzer dargestellt wird. Ist zwar mehr Aufwand, aber das Ergebnis ist schöner :) Stell Dir einfach mal eine Software vor, bei der jeder Benutzer im laufenden Betrieb die Sprache oder gar die Mandantendatenbank wechseln kann. Ist etwas aufwändig, aber nicht unmöglich.

Interaktive Programme – Teil 2

Nun weisst Du ja schon recht gut Bescheid darüber, wie ein Programm was auf dem Bildschirm ausgibt und wie der Benutzer damit interagieren kann. Das war aber noch lange nicht alles. Wir kommen jetzt zu einem sehr wichtigen Thema: *Subfiles*. Darüber werden ganze Bücher geschrieben, also bitte nicht schlagen, wenn hier Details fehlen. Aber ohne Subfiles kann man keine schönen Verwaltungsprogramme schreiben.

Was sucht eine Datei auf dem Bildschirm?



Jeder von Euch kennt die typischen Auswahllisten in einem HTML-Browser, die da vielleicht so aussehen mögen, wie im rechten Beispiel gezeigt. Das Programm möchte einen Wert vom Benutzer ermitteln und zeigt ihm (wenn man auf den Pfeil runter klickt) eine Liste der möglichen Werte an. Die Werte wurden bei Erstellung der HTML-Seite aus einer Datei oder sonstwoher geladen. Der Benutzer sucht sich den Wert aus, den er im Formular auf der Webseite haben will, in diesem Falle klickt er auf „Blau“, einen Wert aus der Liste. Schon wird im Formular das Eingabefeld entsprechend belegt.
Ist einfach, hat aber auch schon einen gewissen Aufwand an Arbeit für denjenigen, der die Webseite programmiert.

Auf der AS/400 gibt es auch solche Auswahllisten, tipp doch einfach mal ein:

```
wrksplf df
```

Du siehst auch eine typische Auswahlliste:

```
                                 Einheit     Benutzer-          Seiten   Akt.
Opt  Datei       Benutzer        Warteschl.  daten       Stat   gesamt   Seite  Kop.
 _   QPPGMDMP    HOLLE           QEZDEBUG                HLD       12             1
 _   MAKE        HOLLE           HOLLEQ                  HLD        7             1
 _   DSPRECB     HOLLE           HOLLEQ                  HLD       10             1
 _   EDTREC      HOLLE           HOLLEQ                  HLD       25             1
 _   QPJOBLOG    HOLLE           QEZJOBLOG   MAKEJOB     HLD        6             1
 _   MAKE        HOLLE           HOLLEQ                  HLD        7             1
 _   DSPRECB     HOLLE           HOLLEQ                  HLD       10             1
 _   EDTREC      HOLLE           HOLLEQ                  HLD       25             1
 _   QPJOBLOG    HOLLE           QEZJOBLOG   MAKEJOB     HLD        6             1
 _   MAKE        HOLLE           HOLLEQ                  HLD        7             1
                                                                            Weitere ...
```

Eine typische Auswahlliste auf der AS/400

Der Benutzer kann hier einen (oder mehrere) Eintrag wählen und bearbeiten.

Wir werden im nun folgenden Projekt auch solche Bildschirme programmieren.

Unser nächstes Projekt – Das Haushaltsbuch

Einleitung

Als Zielprojekt bauen wir uns eine kleine Haushaltsbuch-Verwaltung. Die Idee dazu kam von Hannes (schönen Gruss nach Österreich), da er ein solches Programm für seine Zwecke benötigt. Also behandeln wir das Thema hier im Manual, da es bestimmt für andere Leser auch sehr interessant ist, und man dabei fast alle wichtigen Themen abdecken kann.[1]

In diesen Programmen werden einige Feinheiten der Bildschirmverarbeitung behandelt. Wenn das erledigt ist, gehen wir zu Batch-Programmen über, die sich mit größeren Logikproblemen befassen.

Um unser Haushaltsbuch-Programm kurz zu umreissen, möchte ich einfach folgende Punkte angeben. (Das ist jetzt nicht ein ausführliches Pflichtenheft!)

- Verwaltung von einzelnen Konten, sei es ein Bankkonto, ein Kreditkonto, oder ein Anlagekonto.
- Verwaltung einzelner Kategorien für Einnahmen und Ausgaben (Gehalt, Miete, Lebensmittel, Auto...)
- Einfache Eingabemöglichkeit aller für den Haushalt wichtigen Belege. Jeder Beleg soll eine Belegnummer haben, so dass geprüft werden kann, ob der Beleg schon erfasst wurde.
- verschiedene Auswertungen über Einnahmen und Ausgaben
- schneller Überblick über den Finanzstatus, sowie Rückschauen und Prognosen.
- weitere Ideen bitte an mich.

Die Pflegeprogramme für die Stammdaten (Konten, Kategorien etc) sollen einfach, aber nicht simpel sein. Das Hauptprogramm ist die Erfassung der Belege, auch hier sollte die Bedienung einfach sein, aber die verschiedenen Plausibilitätsabfragen müssen enthalten sein.
Die Auswertungen werden als einzelne Programme realisiert. Um die Anwendung später zu steuern, werden Menüs erstellt.

Datenbanktechnisch wird zunächst simpel vorgegangen, der Programmierer muss sich um die sauberen Inhalte kümmern. Später in diesem Buch fügen wir Constraints (Prüfungen) auf die Tabellen hinzu, damit Du lernst, wie man sich hier das Leben einfacher machen kann.

Zunächst wollen wir uns aber mit den Grundlagen befassen, sonst verlierst Du den Spass am Lesen und legst dieses interessante Buch beiseite. Und das wollen wir doch nicht :-)

[1] Wer auch eine Projektidee in diesem Buch behandelt sehen will, möge mich kontaktieren.

Unsere Datenbank

Unser kleines Haushaltsbuch ist eine Art Mini-Finanzbuchhaltung. Das Prinzip ist ähnlich, auch wenn nur gegen ein Konto gebucht wird. Aber auch bei solch einem kleinen Projekt muss man sauber arbeiten.

Wir werden mit folgenden Tabellen arbeiten:

KONTEN Dateiname: KONTPF (Konto, physikalische Daten)

Feldname	*Feldart*	*Beschreibung*
KONTID	num	eindeutige Nummer je Konto
KONTTXT	alpha50	Bezeichnung des Kontos
KONTART	alpha1	Kontenart (G=Giro, K=Kredit, A=Anlage)

Du siehst, alles sehr übersichtlich. Das Feld KONTID verwenden wir, damit wir erstens aus der Datei eindeutig lesen können und zweitens wir uns gleich angewöhnen, „saubere“ Schlüssel zu verwenden. Wir könnten auch KONTTXT als eindeutigen Schlüssel nehmen, aber stell Dir vor, wenn der Anwender zwei Girokonten eingeben will und als Text jedesmal „Bank“ verwendet. Wir müssten wieder prüfen, dem Anwender einen Fingerzeig geben und haben nichts gewonnen.

KATEGORIEN Dateiname: KATPF (Beleg-Kategorien, physikalische Daten)

Feldname	*Feldart*	*Beschreibung*
KATTXT	alpha50	Bezeichnung der Kategorie
KATMATC	alpha10	Matchcode zur schnellen Suche bei Eingabe eindeutig
KATTYP	alpha1	Kategorietyp (E=Einnahme, A=Ausgabe)

Auch sehr übersichtlich. Hier speichern wir für jede Einnahme- oder Ausgabeart (Gehalt, Auto, Miete, Lebensmittel, Bücher etc.) einen eigenen Satz. Dieser kann später bei der Eingabe selektiert werden. Das Feld KATMATC wird als Suchcode verwendet. Hier kann der Anwender später im Eingabeprogramm suchen. Für diese Tabelle ist KATMATC das eindeutige Feld!

PERIODEN Dateiname: PERPF (Buchungsperioden, physikalische Daten)

Feldname	*Feldart*	*Beschreibung*
PERID	alpha6	eindeutige Bezeichnung der Periode (mmjjjj)
PERTXT	alpha50	Bezeichnung der Periode

Die Periodendatei wird zu Zwecken der Übersicht und zur Prüfung des Eingabezeitraums verwendet. Zum Einen kann man dann für statistische Zwecke jede Buchung einem Zeitraum zuordnen, zum Anderen kann die Eingabe geprüft werden, ob der Zeitraum gültig ist.

BUCHUNGEN Dateiname: BUCHPF (Buchungen, physikalische Daten)

Feldname	*Feldart*	*Beschreibung*
BUCHID	num	eindeutige Nummer pro Buchung
BUCHPER	alpha6	Periode der Buchung (siehe PERIODEN)
BUCHBEL	alpha20	Belegnummer der Buchung
BUCHKAT	alpha10	Kategorie der Buchung (siehe KATEGORIEN, Matchcode)
BUCHTYP	alpha1	Buchungstyp (siehe KATEGORIEN, wird aus Geschwindigkeitsgründen hier redundant gespeichert)[I]
BUCHWERT	num	Wert der Buchung ohne Währung
BUCHTXT	alpha50	Beschreibung der Buchung
BUCHKTO	num	Buchungskonto (siehe KONTEN)

Hier legen wir unsere einzelnen Buchungssätze ab. Die Datei enthält alle relevanten Informationen, die auch später für die Auswertungen nötig sind. Wie angedeutet, nicht das perfekte Datenbankdesign, aber sehr lehrreich. Verbesserungsvorschläge von den DB-Experten sind gern gesehen und können in einen später zu schreibenden Profiteil übernommen werden.

So, nun haben wir unsere Tabellen, also sollten wir unsere Datenbankdefinitionen auch gleich anlegen. Wie bekannt, brauchts dafür erst mal eine Datei für die DDS-Quellen, sofern noch nicht vorhanden.
Für Eilige:

```
CRTSRCPF FILE(meinelib/QDDSSRC) TEXT('Meine DDS-Quellen')
```

Bevor Du aber nun die Definitionen reintippst: *STOP*! Verwende bitte immer die Möglichkeit der Referenzierung. Gewöhne Dir dies gleich an, Du wirst die Vorteile merken, wenn wir mal eine Definition eines Feldes verändern müssen.

Legen wir also zunächst eine Quelle für die Datei #REF an, in der alle verwendeten Felder und Feldtypen enthalten sind[II].

[I] Jaja, ich weiss... ist nicht wirklich die beste Methode nach Grundlage der Normalisierung. Aber Du kannst es auch gern anders machen.
[II] SQL-Profis werden dies als „Dictionary“ kennen.

Hier also der Anfang unserer Datei #REF:

```
 Spalten . . . :    1  71          Editieren                      HHBUCH/QDDSSRC
 SEU==>                                                                   #REF
 FMT PF .....A.........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
       ***************** Datenanfang ****************************************
0001.00      A           R ##REFREC
0001.01       *
0001.02       * *** <generell> ***
0001.03      A             ##PERID         6A         TEXT('Perioden-ID')
0001.04      A             ##KATMATC      10A         TEXT('Kategorie-Matchcode')
0001.05      A             ##KATTYP        1A         TEXT('KatTyp E/A')
0001.06      A             ##WERT         15S 2       TEXT('Betrag')
0001.07      A             ##KONTID        5A         TEXT('Konten-ID')
0001.08      A             ##TXT          50A         TEXT('Beschreibung')
0001.09      A             ##DATUM        10A         TEXT('Datum')
0001.10       *
0001.11       * *** KONTO ***
0002.00      A             #KONTID   R               REFFLD(##KONTID)
0003.00      A             #KONTTXT  R               REFFLD(##TXT)
0003.01      A                                       TEXT('Konto-Text')
0004.00      A             #KONTART        1A         TEXT('Kontenart')
0006.00      A
```

Anfang der Datei #REF

Zunächst fällt der Name für das Satzformat ##REFREC auf. Ist eine alte Angewohnheit. Da dieser Satz und die dort verwendeten Felder nur für Referenzzwecke existieren (also nie wirklich in einem Programm verwendet werden), fangen die Namen bei mir mit # an. Zwei # verwende ich für spezielle Zwecke: Einmal für das Satzformat der Referenzdatei, und zum anderen für Referenzfelder, die als Referenzfeld verwendet werden. Wenn Du Dich nun fragst, ob ich das hier im Wahn schreibe: nein! :-)

Schau Dir einfach mal die ersten Felder an, die ich definiert habe (deren Name auch mit ## beginnt). Ich definiere zum Beispiel ##PERID. Dieses Feld wird an mehr als einer Stelle verwendet (siehe oben). Daher lege ich hier die Grundlage fest, später wird diese für jede einzelne Datei referenziert. Warum ich das so umständlich mache? Nun:

- zum Einen kann ich in hundert Dateien ein Feld verwenden, das einen finanziellen Wert oder eben die Perioden-ID enthält. Die Vorlage definiere ich einmal ganz oben. Später in den einzelnen Dateien definiere ich darauf basierend die in der Datei verwendeten Felder. Falls ich in einer Datei mehrere Werte benötige (sagen wir mal: Soll- und Haben-Wert in einer Finanzbuchhaltung), definiere ich es dort einmal, und später in der richtigen Dateibeschreibung mehrfach, basierend auf einem Wertefeld. Somit laufe ich nicht Gefahr, ein Feld bei einer Änderung zu übersehen.
- zum Anderen dient das Ganze auch der Dokumentation. Ich sehe in der Referenzdatei bereits alle in meiner Datenbank verwendeten Felder inklusive Beschreibung. Somit muss ich mich nicht durch mehrere Dateien blättern.

Wie Du siehst, definiere ich alle Felder, die in den Dateien mehrfach vorkommen (könnten), zunächst mit einem doppelten Gartenzaun[1] als Vorlagen.

Nun fange ich mit den Dateien an, die wir später benutzen werden. Hier wird nicht wirklich der Dateiaufbau dokumentiert, sondern jedes darin verwendete Feld. Die Reihenfolge ist technisch egal, sollte aber einer gewissen Logik folgen und ungefähr dem entsprechen, wie es später in der Dateibeschreibung steht. Es kann nie schaden, wenn das Feld mit dem Primärschlüssel (der eindeutigen Unterscheidung) als erstes definiert wird.

Du siehst bei den Feldern für die Kontendatei, dass ich zwei davon auch innerhalb der Referenzdatei referenziere, und eins dann definiere. Auch das könnte man noch in den generellen Teil oben schreiben, aber es wäre für dieses Beispiel jetzt übertrieben. Mache Dich aber mit dieser Möglichkeit vertraut.
Ich gehe davon aus, dass die Definitionen kein Hexenwerk sind, also folgt gleich der nächste Teil.

Referenzdatei, Teil 2:

[1] oder wie Du das Zeichen # auch immer nennen magst.

```
 FMT A* .....A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
0008.00       * *** KATEGORIEN ***
0009.00      A            #KATMATC  R               REFFLD(##KATMATC)
0010.00      A            #KATTXT   R               REFFLD(##TXT)
0010.01      A                                      TEXT('Kategorie-Text')
0011.00      A            #KATTYP   R               REFFLD(##KATTYP)
0011.01      A                                      TEXT('Kategorie E/A')
0012.00       *
0013.00       * *** PERIODEN ***
0014.00      A            #PERID    R               REFFLD(##PERID)
0015.00      A            #PERTXT   R               REFFLD(##TXT)
0015.01      A                                      TEXT('Perioden-Text')
0016.00       *
0017.00       * *** BUCHUNGEN ***
0018.00      A            #BUCHID       10S 0       TEXT('Buchungs-ID')
0019.00      A            #BUCHPER  R               REFFLD(##PERID)
0020.00      A                                      TEXT('Buchungs-Periode')
0021.00      A            #BUCHBEL      20A         TEXT('Buchungs-BelegNr')
0022.00      A            #BUCHKAT  R               REFFLD(##KATMATC)
0023.00      A                                      TEXT('Buchungs-Kategorie')
0024.00      A            #BUCHTYP  R               REFFLD(##KATTYP)
0025.00      A                                      TEXT('Buchung E/A')
0026.00      A            #BUCHWERT R               REFFLD(##WERT)
0027.00      A                                      TEXT('Buchungswert')
0028.00      A            #BUCHTXT  R               REFFLD(##TXT)
0028.01      A                                      TEXT('Buchungstext')
0029.00      A            #BUCHKTO  R               REFFLD(##KONTID)
0030.00      A                                      TEXT('Buchungskonto')
0031.00      A            #BUCHBDAT R               REFFLD(##DATUM)
0032.00      A                                      TEXT('Belegdatum')
0033.00      A            #BUCHDAT  R               REFFLD(##DATUM)
0034.00      A                                      TEXT('Buchungsdatum')
******************Datenende ***************************************
```

Referenzdatei, Fortsetzung

Du siehst, auch hier ist kein Hexenwerk mehr enthalten. Du kannst aber hier schon Augenmerk darauf legen, dass jedes Feld auch einen Text bekommt.
Ich gehe daher auf die Definitionen nicht weiter ein, sondern springe gleich zu den Dateidefinitionen:

Die Datei KONTPF: (bitte anlegen, Typ ist PF)

```
 FMT PF .....A.........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
       ***************** Datenanfang *********************************************
0001.00      A                                      REF(#REF)
0001.01      A                                      UNIQUE
0001.02       *
0002.00      A          R KONTREC
0003.00      A            KONTID    R               REFFLD(#KONTID)
0004.00      A            KONTTXT   R               REFFLD(#KONTTXT)
0005.00      A            KONTART   R               REFFLD(#KONTART)
0006.00       *
0007.00       * Schlüssel
0008.00      A            K KONTID
      ******************Datenende ***************************************
```

Die Datei KONTEN

Du siehst in der ersten Zeile, dass ich auf die Datei #REF als Referenz zugreife. In der nächsten Zeile sagt das Funktionswort UNIQUE aus, dass der weiter unten definierte Schlüssel als Primärschlüssel eindeutig definiert ist, also jeder Schlüssel des oder der unten definierten Schlüsselfelder nur einmal vorkommen darf.
Dann definieren wir das Satzformat KONTREC und die dort verwendeten Felder. Hier werden keine Zusatzangaben zu den Feldern gemacht, auch die TEXT-Beschreibungen stammen aus der Referenzdatei.
Letztlich wird der Schlüssel für diese Datei definiert, der nur aus dem Feld KONTID besteht. Dieser Schlüssel ist (wie oben beschrieben) eindeutig, es dürfen also nur einmalige Werte in diesem Feld verwendet werden, was auch Sinn macht.

Springen wir ohne Pause (trotz der derzeitigen Temperatur von 36 Grad) weiter zur nächsten Datei:

Die Datei KATPF (auch anlegen, Typ PF):

```
 Spalten . . . :    1  71          Editieren                      HHBUCH/QDDSSRC
 SEU==>                                                                 KATPF
 FMT PF .....A.........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
       ****************** Datenanfang *****************************************
0001.00      A                                     REF(#REF)
0001.01      A                                     UNIQUE
0001.02       *
0002.00      A           R KATREC
0003.00      A             KATMATC    R          REFFLD(#KATMATC)
0004.00      A             KATTXT     R          REFFLD(#KATTXT)
0005.00      A             KATTYP     R          REFFLD(#KATTYP)
0006.00       *
0007.00       * Schlüssel
0008.00      A           K KATMATC
       ******************Datenende ***************************************
```

Die Datei KATPF

Auch hier nichts neues, daher kein weiterer Kommentar.

Die Datei PERPF:

```
 Spalten . . . :    1  71          Ansehen                        HHBUCH/QDDSSRC
 SEU==>                                                                 PERPF
 FMT PF .....A.........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
       ****************** Datenanfang *****************************************
0001.00      A                                     REF(#REF)
0001.01      A                                     UNIQUE
0001.02       *
0002.00      A           R PERREC
0003.00      A             PERID      R          REFFLD(#PERID)
0004.00      A             PERTXT     R          REFFLD(#PERTXT)
0006.00       *
0007.00       * Schlüssel
0008.00      A           K PERID
       ******************Datenende ***************************************
```

Die Datei PERPF

Die Datei BUCHPF:

```
 Spalten . . . :    1  71          Editieren                      HHBUCH/QDDSSRC
 SEU==>                                                                 BUCHPF
 FMT PF .....A.........A.Name++++++RLängeDDsF......Funktionen+++++++++++++++++
       ****************** Datenanfang *****************************************
0001.00      A                                     REF(#REF)
0001.01      A                                     UNIQUE
0001.02       *
0002.00      A           R BUCHREC
0003.00      A             BUCHID     R          REFFLD(#BUCHID)
0004.00      A             BUCHPER    R          REFFLD(#BUCHPER)
0005.00      A             BUCHBEL    R          REFFLD(#BUCHBEL)
0005.01      A             BUCHKAT    R          REFFLD(#BUCHKAT)
0005.02      A             BUCHTYP    R          REFFLD(#BUCHTYP)
0005.03      A             BUCHWERT   R          REFFLD(#BUCHWERT)
0005.04      A             BUCHTXT    R          REFFLD(#BUCHTXT)
0005.05      A             BUCHKTO    R          REFFLD(#BUCHKTO)
0005.06      A             BUCHBDAT   R          REFFLD(#BUCHBDAT)
0005.07      A             BUCHDAT    R          REFFLD(#BUCHDAT)
0006.00       *
0007.00       * Schlüssel
0008.00      A           K BUCHID
       ******************Datenende ***************************************
```

Die Datei BUCHPF

Fertig! Damit werden wir unsere Arbeit nach der Methode „Trial and error“ beginnen.[I]

[I]Heute bezeichnet man das hochtrabend als XP (eXtreme Programming). Man löst das Problem in kleinen Schritten und prüft nach jedem Schritt, ob alles ok ist, eventuell auch im Team :-) [jaja, ich weiss, dass XP etwas mehr als nur das ist!]

Vorbereitende Überlegungen

So, ich gehe davon aus, dass Du alle Dateien fehlerfrei erstellt hast. Nun müssen wir überlegen, was nun zu tun ist. Unser Projekt wird später aus zwei fetten Brocken bestehen: das Eingabeprogramm, und einige Programme für die Auswertung. Drumherum gibt es kleinere Pflegeprogramme.
Bevor wir also in das tiefe Meer (das Eingabeprogramm) springen, machen wir zunächst ein paar Plantschübungen (Datenpflege).
Unser Eingabeprogramm (das man mit Gewalt zur Eierlegenden Wollmilchsau ausbauen kann), braucht Daten, damit wir Daten eingeben können. Ohne ein paar fest definierte Kategorien wird uns das Programm später keine Eingabe zulassen. Ebenso muss eine Buchungsperiode existieren, bevor dafür Daten eingegeben werden dürfen.

Wie kriegen wir nun in diese leeren Dateien ein paar Sätze hinein?

Dateneingabe auf die einfache Art

Wir könnten uns für jede Datei ein Pflegeprogramm schreiben. Macht Sinn, ist aber nicht so ganz ohne, und am Anfang etwas stupid. Du darfst natürlich tun, was Du nicht lassen kannst, aber gehen wir zunächst mit zwei einfachen Methoden ans Werk:

Man nehme Doktor SQL

Du könntest zum Beispiel das interaktive SQL-Tool nehmen (sofern auf Deiner Maschine installiert), und Daten einfügen. Da kann man dieses nette Tool gleich besser kennenlernen. Starte das SQL-Tool mit:

```
STRSQL
```

Um nun in unsere leere Datei KATPF einen Satz einzufügen, gibt es in SQL den Befehl „INSERT INTO“. Ein Beispiel wäre:

```
INSERT INTO KATPF (KATMATC, KATTXT, KATTYP) VALUES('AUTO', 'Alles für die Karre', 'A')
```

Mit diesem Befehl gibt man an, in welche Datei (KATPF) ein Satz angefügt werden soll. Dann kommen in Klammern die Felder, die Werte erhalten sollen, und nach dem Schlüsselvort VALUES wieder in Klammern die Daten für jedes einzelne Feld, in brauchbarer Reihenfolge. Wem das Tippen und die Klammern zu viel sind, kann auch schreiben:

```
INSERT INTO KATPF  F4 DF
```

Der SQL-Befehl wird dann in Bedienerführung angezeigt.

Ach ja, versuche doch mal im SQL-Tool, mit F9 den letzten Befehl (INSERT INTO) zurück zu holen und erneut auszuführen. Das System klopft Dir auf die Finger und sagt, dass Du einen doppelten Schlüsselwert angegeben hast. Zur Erinnerung: KATMATC ist als eindeutiges Feld definiert, logisch!

Wenn Du einen Satz aus der Datei löschen willst, nimm folgendes:

```
DELETE FROM KATPF WHERE KATMATC = 'AUTO'
```

Nach entsprechender Bestätigung und Meldung ist der Satz weg. Und letztlich muss man einen Satz auch ändern können. Das geht dann wie folgt:

```
update katpf set kattxt = 'Alles fürs Auto' where KATMATC = 'AUTO'
```

Das war jetzt kein kompletter SQL-Kurs, aber fürs Gröbste reicht es.
Allerdings ist das doch auf Dauer etwas umständlich, vor allem wenn man mehrere Sätze bearbeiten will. Dafür gibt's dann noch DFU.

Man lasse arbeiten

DFU (oder Disk File Utility) ist ein einfaches, aber gar nicht so schlechtes Programm, wenn man eine Tabelle pflegen will. Man sagt einfach, welche Tabelle man bearbeiten will, beantwortet ein paar Fragen, und schon kann man Sätze einfügen, ändern, löschen. Man kann dort viele Parameter angeben, so dass man schnell den Überblick verliert. Aber das Schöne ist, dass man von DFU ein richtiges, eigenständiges Programm erstellen lassen kann, welches dann für die Pflege unserer Tabelle geeignet ist. Dieses Programm kann man dann mit CALL aufrufen.

```
STRDFU
```

springt in das Menü für DFU. Probiere zunächst den Menüpunkt `5` (Daten mit temporärem Programm fortschreiben).

```
                  Daten mit temporärem Programm fortschreiben

Auswahl eingeben und Eingabetaste drücken.

  Datendatei  . . . . . .   KATPFF        Name, F4=Liste
    Bibliothek  . . . . . .    MEINELIB    Name, *LIBL, *CURLIB
  Teildatei . . . . . .   KATPF         Name, *FIRST, F4=Liste
```

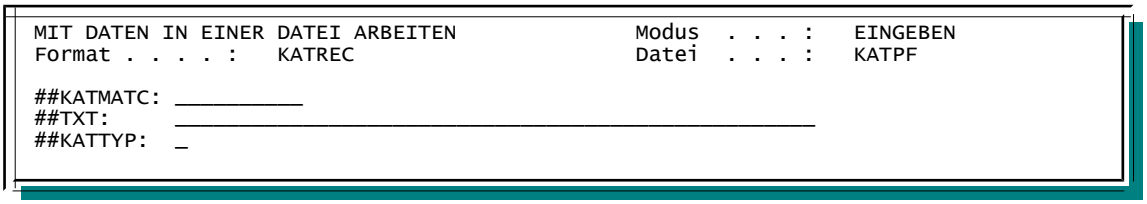
DFU will dann wissen, welche Datei Du bearbeiten willst. Trage entsprechendes ein (bei Teildatei dürfte fast immer der Eintrag „`*first`“ richtig sein) und drücke `df`.
Du landest in einem temporären Programm, das DFU für Dich gerade erstellt hast. Nun siehst Du, wie wichtig es ist, bei der Definition von Dateien auf eine saubere Beschreibung zu achten:

```
MIT DATEN IN EINER DATEI ARBEITEN              Modus  . . . :   ÄNDERN
Format . . . . :   KATREC                      Datei  . . . :   KATPF

##KATMATC: __________



F3=Verlassen            F5=Aktualisieren         F6=Format auswählen
F9=Einfügen             F10=Eingabe              F11=Ändern
```

Wie Du siehst, zeigt DFU uns im Änderungsmodus ein leeres Feld für ##KATMATC an. Wenn Du Dich wunderst, woher diese Beschreibung kommt, schau bitte mal in den Dateidefinitionen nach, wo KATMATC definiert ist. Du siehst, KATMATC beruht auf ##KATMATC in der Referenzdatei, und dort ist kein Parameter TEXT(...) angegeben. Somit wird der Feldname als Text angezeigt. Diesen Schönheitsfeher korrigieren wir nachher. Zunächst brauchen wir ein paar Sätze.

DFU zeigt Dir zunächst nur den Schlüssel für die Datei an. Wenn Du hier einen Wert eingibst, sucht DFU nach einem Satz mit dem entsprechenden Schlüssel, damit Du diesen bearbeiten kannst.
Wir wollen aber *neue* Sätze eingeben. Dafür drücke bitte einmal auf F10, so dass die Modusanzeige oben rechts auf „Eingeben“ wechselt.

Nun siehst Du auch alle drei Felder, die in unserer Datei existieren, und kannst dort Werte eintragen.



(So eine fehlende Feldbeschreibung ist schon nervig, gelle? ;-) Aber Du weisst ja, was zu tun ist.

Gib nun bitte die Werte für unsere Kategorie AUTO ein. Als Text kannst Du Dir was Schönes einfallen lassen, als Typ gib bitte 'A' ein (A für Ausgabe). Danach bestätigst Du mit DF, der Satz wird gespeichert und Du kannst den nächsten Satz eingeben. Tue dies bitte noch mit folgenden Daten:

Matchcode	*Text*	*Typ*
BENZIN	Spritkosten	A
ESSEN	Essensausgaben	A
LOHNM	Lohneinkunft Ehemann	E
LOHNF	Lohneinkunft Ehefrau	E
KINO	Ausgaben fürs Kino	A
SPENDE	Spenden von Oma	E

Du kannst selbstverständlich noch weitere Sätze eingeben, aber das sollte zunächst reichen. Verlasse DFU mit der Taste F3 und bestätige das Beenden. Du kannst gerne mit SQL prüfen, ob Deine Eingaben auch richtig in der Tabelle gelandet sind.

Nun benötigen wir noch ein paar Sätze in der Kontendatei und für die Buchungsperioden. Ich verlasse mich nun auf Dein Geschick, mit DFU umzugehen.[I] Bitte erstelle folgende Sätze:

[I]Tipp: Du kannst eine Datei direkt ändern, in dem Du eingibst: UPDDTA <datei> . UPDDTA ist das Gleiche wie der Menüpunkt 5 nach STRDFU.

Datei PERPF:

PERID	*PERTXT*
200309	September 2003
200310	Oktober 2003
200311	November 2003
200312	Dezember 2003

Datei KONTPF:

KONTID	*KONTTXT*	*KONTART*
GIROM	Girokonto Ehemann Sparkasse Nr. 4823812	G
GIROF	Girokonto Ehefrau Citybank Nr. 123921821921	G
ANLS	Anlagekonto Sparkasse Nr. 393831213	A

Wenn wir diese drei Dateien gefüllt haben, steht unsere Datengrundlage. Nun wird's heikel.

Die Datengrundlage haben wir, nun braucht es „nur noch“ das Programm, um mit diesen Daten unsere Buchungen einzugeben. Kein Problem, klickt man sich in VisualBasic in 5 Minuten zusammen. Nun... möglich, aber ob es nachher auch so einwandfrei im Netzwerk funktioniert, ist fraglich.

Programmablauf

Was muss unser Eingabe-Programm denn nun alles können? Da es wohl das meist benutzte Programm in unserem Projekt ist, sollte man sich das vorher überlegt haben. Es ist eigentlich simpel:

1. Bildschirmmaske darstellen
2. Werte eingeben
3. Werte prüfen
4. Werte in Buchungsdatei schreiben
5. gehe nach 1.

Ist ja nicht wirklich wild. Aber spätestens bei 3) kann es schon interessant werden. Und der geneigte Benutzer hätte gerne bei 1) und 2) etwas Komfort. Bei 4) darf natürlich nichts schiefgehen. Einzig 5) ist schnell gelöst. Du wirst selbst sehen, dass 2) der Punkt mit der meisten Arbeit wird, mit nur geringem Abstand folgt 3.)

Zusätzliche Funktionen im Programm sollten sein:
1. Liste der eingegebenen Buchungen anzeigen
2. Stornieren einer eingegebenen Buchung
3. Verzweigung in Auswertungsroutinen (alternativ: Aufruf über ein Menü)
4. Schneller Kassensturz auf Knopfdruck

Aber zu diesem Luxus kommen wir später in einem eigenen Workshop-Teil. Als gute Programmierer wissen wir dann, wie man in ein vorhandenes Programm schnell eine weitere Funktion einbaut.[I]

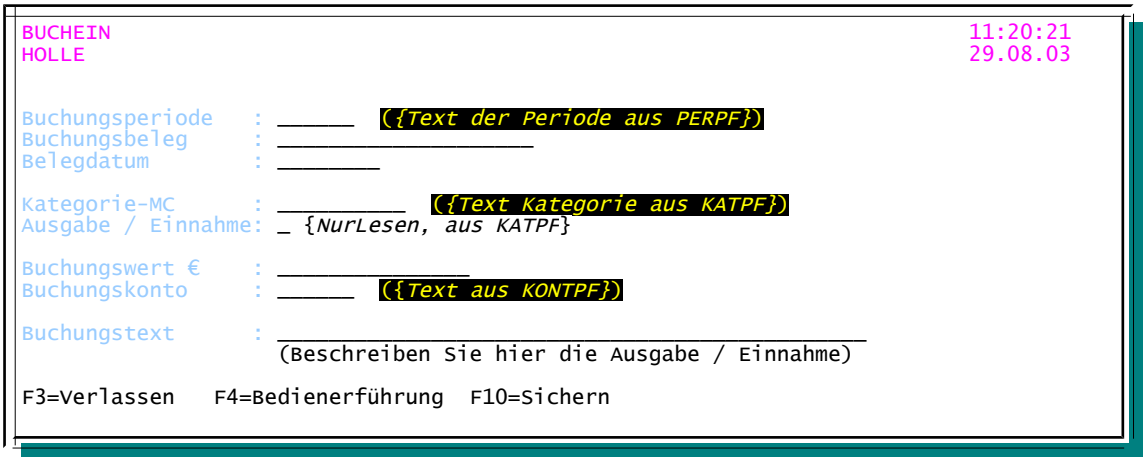
Der Bildschirm

Fangen wir mit einer Überlegung zur Optik an. So ein Eingabebildschirm sollte nicht unübersichtlich sein, auch ein unbedarfter Benutzer sollte gleich damit etwas anfangen können.

Beim Start des Programms ist es bestimmt nützlich, direkt in den Eingabemodus zu gelangen (ähnlich beim DFU). Einen Änderungs- oder Löschmodus sollte es aus Gründen der Ordnung erst gar nicht geben. Wir haben es hier ja mit einer Minimal-Buchhaltung zu tun, und in einer Buchhaltung darf nur storniert werden.
Nach dem der Benutzer die Daten eingegeben hat, sollen diese geprüft werden. Erst wenn alles plausibel ist, soll ein Satz in die Buchungsdatei geschrieben werden, ansonsten soll eine Fehlermeldung den Nutzer auf sein Vergehen hinweisen :-)

[I]Solltest Du das noch nicht wissen, ist es Ziel dieses Buches, Dir das beizubringen.

Unser Eingabebildschirm könnte ungefähr so aussehen:



Die Farbgebung ist natürlich nur ein Beispiel und in einer Textverarbeitung nicht ganz einfach zu realisieren.
Hinter den Feldern „Periode“, „Kategorie-MC“ sowie „Buchungskonto“ wird bei gültiger Eingabe das Textfeld aus der entsprechenden Datei angezeigt.
Das Feld Ausgabe / Einnahme ist ein reines Ausgabefeld und enthält je nach Buchungskategorie den Inhalt „A“ oder „E“. Am Fuss kommen die obligatorischen Funktionstasten, fertig.

Nun geht es also an die zugehörige Bildschirmdatei.

Eine feine Bildschirmdatei

Unser Bildschirm wird aus einer Menge an einzelnen Elementen bestehen. Diese habe ich jetzt noch nicht beschrieben, Du wirst bei der Definition der Bildschirmdatei sofort merken, was für eine Arbeit dahinter steckt.

Es sei Dir auf jeden Fall empfohlen, bei der Eingabe nach einigen Zeilen jeweils eine Pause zu machen und Dir die Befehle mal in Ruhe anzuschauen. Nutze auch die Tasten F4 und F1. Das fördert das Verständnis.

Wir tragen alle Definitionen in eine Quelldatei ein. Diese heisst BUCHDSP. Hier der Anfang:

```
 Spalten . . . :    1  71          Editieren                      MYLIB/QDDSSRC
 SEU==>                                                               BUCHDSP
 FMT A* .....A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
0000.01       *************************************************************
0000.02       * DATEI   : BUCHDSP
0000.03       * PROJEKT: Haushaltsbuch
0000.04       * AUTOR   : Holger Scherer
0000.05       * DATUM   : 28.08.2003
0000.06       * VERSION: 0.1
0000.07       *************************************************************
0000.08
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0000.09       * -=* GENERELLE ANGABEN *=-
0000.10      A                                     DSPSIZ(24 80 *DS3)
0000.11      A                                     REF(#REF)
0000.12      A                                     PRINT
```

Anfang der Bildschirmdatei BUCHDSP

Zunächst beginne ich einen kleinen Kommentarblock. Danach kommen generelle Angaben, die für alle Satzformate in dieser Bildschirmdatei gelten. `DSPSIZ` definiert die Auflösung des Bildschirms. Danach kommt die bekannte Bestimmung, woher Referenzfelder geholt werden sollen. `PRINT` sagt dem System, dass der Benutzer die Hardcopy-Taste verwenden darf.

Nun folgen die einzelnen Satzformate, aus denen wir den Bildschirm aufbauen.

Beginnen wir mit dem Kopfsatz. Dieser definiert die ersten zwei Zeilen und sollte nach Möglichkeit in jedem Programm und jeder Funktion gleich aussehen, damit der Benutzer nicht sofort verwirrt wird. Ich habe diese so definiert:

```
 Spalten . . . :    1  71          Editieren                      HHBUCH/QDDSSRC
 SEU==>                                                               BUCHDSP
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0001.00      A          R KOPF01REC
0001.01      A            K01USRNAM     10A  O  1  2TEXT('Benutzername')
0001.02      A                                      COLOR(TRQ)
0001.03      A                                      DSPATR(HI)
0001.04      A            K01JOBNAM     10A  O  2  2TEXT('Jobname')
0001.05      A                                      COLOR(TRQ)
0001.06      A                                      DSPATR(HI)
0001.07      A                                   1 73DATE
0001.08      A                                      COLOR(TRQ)
0001.09      A                                      DSPATR(HI)
0001.10      A                                      EDTCDE(Y)
0001.11      A                                   2 73TIME
0001.12      A                                      COLOR(TRQ)
0001.13      A                                      DSPATR(HI)
0001.14      A            K01TITEL      50A  O  1 15TEXT('Bildschirmtitel')
0001.15      A                                      COLOR(WHT)
```

Der Formatname in Zeile 0001.00 lautet „KOPF01REC“. Dies ist meine Angewohnheit, den Formatnamen zu Gruppieren, um bei grossen oder umfangreichen Bildschirmen nicht durcheinander zu kommen. Die ersten vier Stellen nenne ich „KOPF“, „BODY“ oder „FUSS“. Danach kommen zwei Stellen für eine eventuell nötige Durchnumerierung (Beachte: Null, nicht der Buchstabe O an fünfter Stelle des Namens). Danach steht „REC“ für Satzformat. Möge jeder Programmierer sein eigenes Namensschema verwenden, hauptsache, er findet sich damit zurecht. Sollte er im Team arbeiten, müssen sich aber auch die anderen Programmierer damit zurechtfinden!

Nun kommt das Feld K01USRNAM (K für Kopf, 01 als Numerierung, USRNAM als Inhaltsbezeichnung). Es mag verwirren, dass ich hier stellenweise englische und deutsche Kürzel oder Begriffe vermische. Aber im Laufe der Zeit prägen sich einige Kürzel doch stark ein, und USRNAM ist irgendwie treffender als beispielsweise BENNAM. Auch hier gilt das Gesagte über den eigenen Geschmack. Ich habe die Felder USRNAM und JOBNAM fest im Bildschirm definiert, könnte das aber auch sauberer in der Referenzdatei machen. Ich wollte für dieses Beispiel die Referenzdatei nicht zu sehr aufblähen. Bei späteren Projekten machst Du das aber besser richtig ;-)

JOBNAM enthält später den Namen des vom Benutzer verwendeten Bildschirms. Genauer gesagt wird hier der Name des Jobs eingetragen, bei interaktiven Jobs ist dies fast immer der Name des Bildschirms.

Beide besagte Felder werden später im Programm mit Inhalt gefüllt.

Die Funktionen COLOR und DSPATR sind einfach, sie sagen, in welcher Farbe und Helligkeit das Feld dargestellt werden soll.

Es folgen zwei Systemfelder DATE und TIME. Hier wird nur im Funktionsfeld geschrieben, welches Systemfeld mit welchen Attributen dargestellt werden. Es sind automatisch reine Ausgabefelder.
Beim Datumsfeld DATE habe ich zusätzlich mit der Funktion EDTCDE angegeben, dass dieses Feld mit Trennzeichen (in Deutschland ein Punkt) angezeigt werden soll. Ansonsten würde das System das Datum sechsstellig ohne Trennzeichen zeigen, nicht so toll. Beim Zeitfeld ist diese Angabe nicht nötig.[I]

Das Feld K01TITEL dient der Bildschirmüberschrift. Damit es nachher schön zentriert auf dem Bildschirm erscheint, musst Du im Programm bei der Belegung ein paar Leerstellen davortun.
Machen wir nun weiter mit dem Fuss des Bildschirms.[II]

Das Satzformat für den Bildschirmfuss ist einfach:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0029.00      A          R FUSS01REC
0031.00      A                                       OVERLAY
0032.00      A                                    22 01'F3=Verlassen'
0033.00      A                                       COLOR(BLU)
0034.00      A                                      +2'F4=Bedienerführung'
0035.00      A                                       COLOR(BLU)
0036.00      A                                      +2'F10=Sichern'
0037.00      A                                       COLOR(BLU)
0038.00      A            F1MELDUNG      78A     24 01COLOR(RED)
```

Der Fussteil

[I] Auch hier gibt es später noch viele Möglichkeiten, das eleganter zu erledigen.
[II] Wie schon geschrieben, die Reihenfolge der Definitionen in der Bildschirmdatei fast immer unwichtig. Ausnahme: Subfiles!

Einzig auffällig sind hier die Zeilen 34 und 36. Ich habe hier keine absolute Zeilen- und Spaltenangabe gemacht, sondern bei Position nur `+2` angegeben. Das bedeutet, dass der definierte Text in der gleichen Zeile mit zwei Leerstellen Abstand zum vorigen Feld angezeigt wird. Ist recht brauchbar, wenn man mal schnell in der Definition einen Text löscht oder einfügt. Man erspart sich das Spaltenzählen. Als letztes Feld habe ich noch `F1MELDUNG` definiert, eine rote Zeile in der wir Fehlermeldungen ablegen können.

Auf geht's zum wichtigsten, dem Hauptteil des Bildschirms:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0003.00      A          R BODY01REC
0003.01      A                                      BLINK
0003.02      A                                      OVERLAY
0003.03      A                                      CF03(03)
0004.00      A                                   5  1'Buchungsperiode   :'
0005.00      A                                      COLOR(BLU)
0006.00      A            B1PERID   R        B  5 21REFFLD(#BUCHPER)
0007.00      A                                      COLOR(GRN)
0008.00      A            B1PERTXT  R        O  5 28REFFLD(#PERTXT)
0009.00      A                                      COLOR(YLW)
0010.00
0011.00      A                                   6  1'Buchungsbeleg     :'
0012.00      A                                      COLOR(BLU)
0014.00      A            B1BEL     R        B  6 21REFFLD(#BUCHBEL)
0015.00      A                                      COLOR(GRN)
0015.01      A                                      CHECK(LC)
```

Nach dem Namen des Satzformates kommt zunächst die Funktion `BLINK`, die besagt, dass der Cursor blinken darf (es wird seinen Grund haben, dass man dies deaktivieren kann). Es folgt `OVERLAY`. Dadurch wird dieses Format über bereits auf dem Bildschirm dargestellte Sätze angezeigt. Würden wir dieses weglassen, wird der Bildschirm geleert, womit unsere Kopf- und Fusszeilen verschwinden.

Danach definiere ich, dass die Funktionstaste gedrückt werden darf und dies über den Indikator `*IN03` gemeldet wird.

Nun folgen zeilenweise die Feldbeschreibung jeweils als Text in der Farbe blau, sowie rechts davon stehende Felder. Für jedes Eingabe- oder Darstellungsfeld definiere ich hier temporäre Variablen in diesem Satzformat. Der Name beginnt mit „`B1`“ (für Body, der erste). Danach folgt das Suffix des Variablennamens aus der entsprechenden Datei. Dadurch ist sofort ersichtlich, welcher Feldinhalt hier abgelegt wird. Solltest Du mit der einstelligen Numerierung irgendwann mal nicht mehr auskommen, ist sowieso der richtige Zeitpunkt gekommen, um einen guten Plan für die Feldnamen zu haben.

Die Definitionen sind recht einfach. In der Zeile 0015.01 besagt `CHECK(LC)` zusätzlich, dass hier auch Kleinbuchstaben erlaubt sind.

Zum Thema Feldnamen noch eins: Versuche nach Möglichkeit immer, dieses Prinzip durchzuführen. Also: Feldnamen in einer Datei aus Prefix (Dateiname) und Suffix

Abschliessend folgt hier kommentarlos der Rest des Hauptbildschirms:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0017.00      A                                      7  1'Belegdatum        :'
0018.00      A                                         COLOR(BLU)
0020.00      A            B1BDAT    R        B  7 21REFFLD(#BUCHBDAT)
0021.00      A                                         COLOR(GRN)
0021.01
0023.00      A                                      9  1'Kategorie MatchC  :'
0024.00      A                                         COLOR(BLU)
0024.01      A            B1KATMATC R        B  9 21REFFLD(#BUCHKAT)
0024.02      A                                         COLOR(GRN)
0024.03      A            B1KATTXT  R        O  9 33REFFLD(#KATTXT)
0024.04      A                                         COLOR(YLW)
0024.05
0024.06      A                                     10  1'Ausgabe / Einnahme:'
0024.07      A                                         COLOR(BLU)
0024.08      A            B1KATTYP  R        O 10 21REFFLD(#KATTYP)
0025.00
0025.01      A                                     12  1'Buchungswert in € :'
0025.03      A            B1BUCHWERTR        B 12 21REFFLD(#BUCHWERT)
0025.04      A                                         COLOR(GRN)
0025.05                                                EDTCDE(J)
0025.06      A                                     13  1'Buchungskonto     :'
0025.07      A                                         COLOR(BLU)
0025.08      A            B1KONTID  R        B 13 21REFFLD(#BUCHKTO)
0025.09      A                                         COLOR(GRN)
0025.10      A            B1KONTTXT R        O 13 28REFFLD(#KONTTXT)
0025.11      A                                         COLOR(YLW)
0025.12
0025.13      A                                     15  1'Buchungstext      :'
0025.14      A                                         COLOR(BLU)
0025.15      A            B1BUCHTXT R        B 15 21REFFLD(#BUCHTXT)
0025.16      A                                         COLOR(GRN)
0025.17      A                                         CHECK(LC)
0025.18      A                                     16 21'(Beschreiben Sie hier +
0025.19      A                                         die Ausgabe / Einnahme)'
```

Fertig. :-) Bitte diese Bildschirmdatei umwandeln und eventuelle Fehler prüfen. Wir werfen uns gleich ins Zeug und machen uns an das Hauptprogramm, aber... wie üblich... gaaanz langsam.

Das Hauptprogramm

Zunächst wollen wir wissen, wie unsere Bildschirmdatei nun auf dem Bildschirm aussieht. Denn so eine umgewandelte Bildschirmdatei kann man sich nicht ohne Weiteres anzeigen lassen. Aber das macht nichts. Wir beginnen mit unserem Hauptprogramm, binden die Bildschirmdatei ein, belegen ein paar Variablen mit mehr oder weniger sinnvollen Werten und zeigen die Bildschirmsätze an. Das reicht für den ersten Überblick.

Erster Anfang

Wechsle also in Deine Sourcendatei QRPGLESRC (falls nicht vorhanden: mach Dir eine mit CRTSRCPF). Lege eine neue Teildatei mit dem Namen BUCHEIN an und fülle sie mit den folgenden Zeilen:

```
FMT F   FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0001.00 FBUCHDSP   CF   E             WORKSTN
0002.00
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0002.01 C                   EVAL      K01USRNAM = 'Holle'
0002.02 C                   EVAL      K01JOBNAM = 'BUCHEIN'
0002.03 C                   EVAL      K01TITEL  = '          Buchungen im +
0002.04 C                             Haushaltsbuch eingeben'
0002.05 C                   EVAL      B1PERID = '200309'
0002.06 C                   EVAL      B1PERTXT= 'September 2003'
0003.00 C                   WRITE     KOPF01REC
0003.01 C                   WRITE     FUSS01REC
0003.02 C                   EXFMT     BODY01REC
0003.03
0004.00 C                   EVAL      *inLR = *ON
        ******************Datenende ***************************************
```

Das sieht ja sehr übersichtlich aus. In Zeile 001.00 wird insere Bildschirmdatei eingebunden (denke daran: Dateiart ist `C` für „combined“, Kombiniert Ein/Ausgabe; Verwendung ist `F` für „full procedural“, komplett selbst gesteuert).

Im Anweisungsteil belegen wir einfach ein paar der im Bildschirm definierten Variablen. Danach wird mit WRITE der Kopfsatz und der Fusssatz geschrieben. Letztlich wird das Satzformat BODY01REC ausgeführt, das heisst, erst geschrieben, dann gelesen.
Warum? Einfach: Kopf und Fuss enthalten nur Ausgabevariablen. Diese schreiben wir auf den Bildschirm. Im Hauptteil sind aber auch Eingabefelder. Also müssen wir den Satz erst auf den Bildschirm schreiben, dann die Eingaben des Benutzers lesen, damit wir sie in unseren Feldern stehen haben. Der Befehl dazu heist EXFMT (execute format). Probiere doch mal aus, was passiert, wenn Du EXFMT durch WRITE ersetzt. Richtig, das Programm schreibt alle Satzarten auf den Bildschirm und beendet sich dann. Und nach Programmbeendigung siehst Du wieder den vorigen Bildschirm (SEU etc.). Also merke: Bei jedem Bildschirm, den man programmiert, hat man oft diverse Satzarten, die nur geschrieben (WRITE) werden, und ein steuerndes Satzformat, das geschrieben und gelesen wird (meist in einer Schleife). Spasseshalber kannst Du ja auch noch aus experimentellen Gründen alle WRITE durch EXFMT ersetzen.

Wandele Dein Programm um und Teste ein wenig. Es ist Dein zweites RPG-Programm mit Bildschirmdateien, also sollte es kein Problem mehr darstellen.

Füllen wir auch dieses Programm mit etwas mehr Leben.

Initialisierungen

Jedes Programm sollte vor seiner eigentlichen Arbeit ein paar Initialisierungen durchführen. Das sind vorbereitende Befehle, die den restlichen Ablauf vorbereiten. Bei unserem Programm ist das recht einfach. Wir haben im Kopfteil des Bildschirms zwei Felder definiert, die den Benutzer und den Jobnamen darstellen sollen. Da wir diese beiden Werte nicht fest im Programm festlegen können, müssen wir sie beim Programmstart beim System erfragen.

Dies geht recht einfach. Im Betriebssystem gibt es eine Menge Datenstrukturen, das ist eine Sammlung von Feldern mit verschiedenen Inhalten. Jeder Job hat auch eigene Datenstrukturen, eine davon ist die so genannte Programmstatusdatenstruktur (dieses lange Wort hat IBM erfunden). In dieser DS[I] stehen verschiedene interessante Werte.

Wir müssen nun in unserem Programm ein paar Felder definieren, die Werte aus dieser DS erhalten sollen. Das geht so: man definiere eine Datenstruktur im Programm (also eine Sammlung von Feldern) und durch eine spezielle Angabe erreichen wir, dass diese Struktur vom System gefüllt wird:

```
FMT D   DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0002.01 DSYSDS            SDS
0002.02 D xJOBNAM                 244    253
0002.03 D xUSRNAM                 254    263
```

Um diese Zeilen einfach eingeben zu können, gib zunächst mal nach der Zeile mit der F-Bestimmung als Zeilenbefehl „FD“ ein. Das bedeutet, das System soll Dir eine Formatierungszeile vom Typ D (Data definitions) anzeigen. Danach wechselst Du mit dem Zeilenbefehl „I“ in den Einfügemodus.

In Zeile 002.01 definieren wir eine Datenstruktur mit dem Namen „SYSDS“. Dieser Name ist frei gewählt, da wir aber auf die Programmstatus-DS zugreifen, macht der Name „SYSDS“ einen Sinn. Du könntest diese DS aber auch „HARRY“ nennen. Da wir nur auf Teile dieser Struktur zugreifen, benötigen wir den Namen nicht.

In Spalte „T“ (die beim prompten mit „Art der Datenstruktur“ überschrieben ist, das T ist total verwirrend, daher wohl auch „T“) muss ein „S“ eingegeben werden. Diese besagt, dass die definierte Datenstruktur ihre Werte aus der Programmstatus-DS erhalten soll. Ansonsten kann man auch leere DS definieren.
Und unter „Ds“ muss man auch „DS“ eingeben, damit das System überhaupt weiss, dass diese D-Zeile eine Datenstruktur beginnt. Denn nun kann man in den folgenden Werten die einzelnen Felder dieser Struktur angeben.
Stell Dir eine Datenstruktur als eine lange Zeichenkette vor. Nun sage ich in den folgenden Zeilen, dass das Feld „xJOBNAM“ mit den Stellen 244 bis 253 aus der Datenstruktur gefüllt wird, und das Feld „xUSRNAM“ mit den Stellen 254 bis 263. Da durch Angabe von „S“ im Feld „Art der Datenstruktur“ das System die Struktur beim Programmstart automatisch füllt, haben wir unsere gewünschten Werte in den beiden Feldern. Im Anhang E findest Du alle Informationen, die man aus der Programmstatus-Datenstruktur herausholen kann.

[I] Ich kürze „Datenstruktur“ ab hier mit „DS“ ab.

Ich habe in Zeilen 002.02 und .03 die Namen der Unterfelder dieser Struktur um eine Spalte nach rechts eingetragen. Du könntest sie aber auch direkt nach dem „D“ beginnen. Ich mache das nur wegen der Optik.

Ändere nun die Zweisung der beiden Felder K01USRNAM und K01JOBNAM im Programm auf die Felder in der Datenstruktur, und schau:

```
0002.05 C                   EVAL      K01USRNAM = xUSRNAM
0002.06 C                   EVAL      K01JOBNAM = xJOBNAM
```

Nach dieser Änderung sieht man im Programm gleich, wer an welchem Bildschirm damit arbeitet. Hat doch was, oder? Das ist immer nützlich, wenn ein Benutzer ein Problem hat und beim Usersupport anruft. Mit der Angabe des Benutzernamens und des Bildschirmnamens kann der Support direkt auf den Job zugreifen (z.B. `WRKUSRJOB <BENUTZERNAME>`).

So, nun geht es zum nächsten Schritt. Die Felder für die Angabe der Buchungsperiode haben wir fest vobelegt. Das lassen wir nun auch vorerst so, eine Auswahl der Periode bauen wir später ein. Machen wir uns zunächst eine Eingabeschleife. Ich schlage vor, der Benutzer soll so lange Daten eingeben und bearbeiten können, bis er die Taste <DF> drückt. Dann werden alle Felder geprüft, ob die benötigten Felder einen Inhalt haben und ob dieser einen Sinn macht. Ist etwas faul, gibt es eine Meldung auf dem Bildschirm, stimmt alles, wird der Satz geschrieben und dem Benutzer wieder eine leere Maske präsentiert.

Wie wir in unserem ersten Programm gelernt haben, kann EXFMT eines Bildschirmformats einige Indikatorvariablen zurückmelden, abhängig davon, welche Sondertaste oder Funktionstaste gedrückt wurde. Bei <DF> können wir das aber nicht feststellen. Wir müssen also in der Schleife, die unser Programm durchläuft, ein Kennzeichen definieren, wann die Schleife beendet wird. Dazu nimmt man am einfachsten ein selbst definiertes Feld, das anzeigt ob die Schleife beendet werden soll. Sprich:

```
PROGRAMMANFANG
SETZE „BILD1ENDE“ AUF „FALSCH“
SOLANGE „BILD1ENDE“ = „FALSCH“
     BILDSCHIRM AUFBAUEN
     BENUTZEREINGABEN HOLEN
     EINGABEN PRÜFEN. WENN EINGABEN OK, DANN SETZE „BILD1ENDE“ AUF
     „WAHR“
WIEDERHOLE SCHLEIFE
SATZ SCHREIBEN
...
```

Das wäre ein grober Überblick über den Programmablauf. Aus Gründen der Übersicht bauen wir das Programm aber auch gleich in Subroutinen auf. Das sieht dann so aus:

Zunächst brauchen wir eine Variable, die uns anzeigt, dass Bildschirm 1 beendet werden soll[I]. Diese Variable wird später ein einer Bearbeitungsschleife abgefragt.

Füge diese Zeile unter die vorhandenen D-Zeilen an:

```
FMT D   DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0002.04 D#B1ENDE          S              1A   INZ('N')
```

Damit erhalten wir eine einstellige Zeichenvariable namens `#B1ENDE`, die mit dem Wert „N“ vorbelegt wird.

Nun bauen wir unser Hauptprogramm etwas um:

```
0002.06 C                   EXSR      INIT
0002.13 C                   EXSR      BILD1
0004.00 C                   EVAL      *inLR = *ON
```

Das Hauptprogramm

Das ist ein ganz einfaches Hauptprogramm, es beendet sich auch nach Bestätigung des Bildschirms durch Datenfreigabe gleich vor lauter Scham wieder.

[I]Wir haben eigentlich nur einen Bildschirm in unserem Programm, aber trotzdem gönnen wir ihm eine Nummer.

Saubere Programme durch Unterroutinen

Nun folgen die Unterroutinen (diese müssen in RPG nach dem Setzen von `*INLR` definiert werden): Zunächst die (vorübergehende) Initialisierung einiger Variablen:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0006.00  *** SUBROUTINEN ***
0006.01  *   ===========
0006.02
0006.03  * ____________________________________________________________________
0006.04  * INIT : Initialisierungen
0007.00 C     INIT          BEGSR
0007.01 C                   EVAL      K01USRNAM = xUSRNAM
0007.02 C                   EVAL      K01JOBNAM = xJOBNAM
0007.03 C                   EVAL      K01TITEL  = '          Buchungen im +
0007.04 C                             Haushaltsbuch eingeben'
0008.00 C                   ENDSR
```

Hier werden ein paar Variablen initialisiert

Eine Schleife, die unseren Bildschirm so lange darstellt, bis der Anwender keine Lust mehr hat:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0010.01  * ____________________________________________________________________
0010.02  * BILD1: Bildschirm 1 darstellen und bearbeiten
0010.03 C     BILD1         BEGSR
0011.00 C                   DOW       #B1ENDE = 'N'
0012.00 C                   WRITE     KOPF01REC
0013.00 C                   WRITE     FUSS01REC
0014.00 C                   EXFMT     BODY01REC
0014.01 C                   EXSR      CHKF1
0019.00 C                   ENDDO
0019.01 C                   ENDSR
```

Die Schleife zum Zeigen des Bildschirms 1

Eine kleine Prüfung, welche Funktionstaste der Anwender in Bildschirm 1 gedrückt hat:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0021.00  * ____________________________________________________________________
0022.00  * CHKF1: Funktionstasten von Bild1 bearbeiten
0023.00 C     CHKF1         BEGSR
0023.01  * F3 gedrückt? -> Ende *
0023.02 C                   IF        *in03 = *ON
0023.03 C                   EVAL      #B1ENDE = 'J'
0023.04 C                   ENDIF
0024.00 C                   ENDSR
        ******************Datenende ***************************************
```

Und hier werden Funktionstasten geprüft

So, prüfe mal das Programm. Du wirst nur mit F3 rauskommen. Das ist aber auch gewünscht.

Nachdem Du damit experimentiert hast, wird es ernst. Wir müssen nun zusätzlich prüfen, ob Eingaben vorhanden sind und ob diese auch sinnvoll sind. Die Routinen werden ja auch durchlaufen, wenn der Benutzer auf DF drückt.
Wenn die Eingaben ok sind, dann muss zum Einen `#B1ENDE` auf „`J`“ gesetzt werden, damit die Bildschirmschleife beendet wird. Weiterhin muss eine neue Variable `#B1OK` auf „`J`“ gesetzt werden, damit (noch ungeschriebene) Routinen wissen, dass Daten zu speichern sind.

Also erst mal definieren wir uns unsere Variable, die darstellt, dass der Anwender was brauchbares eingegeben hat:

```
FMT D   DName++++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0002.05 D#B1OK             S               1A   INZ('N')
```

Variable zum Anzeigen ob die Eingaben ok sind

Nun brauchen wir eine neue Prüfroutine.

Füge bitte in Routine `BILD1` nach dem Aufruf von `CHKF1` folgende Zeilen hinzu:

```
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0014.03 C                   IF        #B1ENDE = 'N'
0014.04 C                   EXSR      CHKEIN1
0014.05 C                   ENDIF
```

Aufruf weiterer Prüfungen.

Wir rufen unsere Prüfroutine nur auf, wenn der Benutzer keine Taste wie F3 gedrückt hat. Wäre ja blöd, der Anwender will das Programm beenden und wir belästigen die CPU noch mit Datenprüfungen. :-)

Hier nun die Routine `CHKEIN1` (Minimalvariante):

```
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0027.00  * CHKEIN1: Prüfen der Eingaben von Bildschirm1
0028.00 C     CHKEIN1       BEGSR
0028.01  * Aufwändige Variante, aber übersichtlich:
0028.02 C                   EVAL      #B1OK = 'J'
0028.03
0028.04  * ?PERIODE?
0028.05 C                   IF        #B1OK = 'J'
0028.06 C                   IF        B1PERID = *blanks
0028.07 C                   EVAL      #B1OK = 'N'
0028.08 C                   EVAL      F1MELDUNG = 'Periode angeben!'
0028.09 C                   ENDIF
0028.10 C                   ENDIF
0028.11
0029.00 C                   ENDSR
```

Prüfroutine, ein Anfangswert

Viel passiert hier an sich nicht. Zunächst gehen wir (im Zweifel immer für den Angeklagten) davon aus, dass der Benutzer alles richtig gemacht hat. Nun werden einzelne Felder geprüft. Sobald ein Feld nicht gültig gefüllt ist, werden die folgenden Felder nicht weiter geprüft und die Meldung belegt. Somit wird immer nur ein Fehler ausgegeben. So wird der Anwender Schritt für Schritt zum Erfolg gezwungen :)

In Zeile `0028.05` wird geprüft, ob noch alles im grünen Bereich ist (siehe oben). Wenn ja, wird geprüft, ob das Eingabefeld für die Buchungsperiode leer ist. Falls ja, liegt hier ein Fehler vor. Das OK-Kennzeichen wird auf „`N`“ gesetzt und das Meldungsfeld belegt. Probier's aus!

Nun müssen wir aber noch mehr machen. Wenn das Periodenfeld belegt ist, muss geprüft werden, ob in der Datei `PERPF` ein passender Satz hierfür existiert. Wenn ja, muss aus dem Satz das Beschreibungsfeld gelesen werden, wenn nein: *Fehler!* Du merkst, Eingabeprüfungen für einen Bildschirm können sehr aufwändig sein.

Zunächst brauchen wir also die PERPF-Datei in unserem Programm. Folgende Zeile am Anfang bei den F-Bestimmungen zufügen:

```
FMT F   FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0001.01 FPERPF     IF   E           K DISK
```

Als Dateiart reicht hier „I“ für Input, wir wollen ja nur etwas lesen. Beachte, dass bei „Art der Satzadressierung“ ein „K“ eingetragen ist. Sonst klappt der Zugriff über einen Suchschlüssel nicht. Du erhälst dann beim Unwandeln eine Fehlermeldung beim CHAIN-Befehl.

Dann definieren wir uns einen Key (IBM-Deutsch: Zugriffsschlüssel), mit dem wir in der Periodendatei suchen:

```
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0002.07 C     *LIKE         DEFINE    PERID         kyPERID
0002.08 C     kyPERREC      KLIST
0002.09 C                   KFLD                    kyPERID
```

Zugriffsschlüssel für PERPF

Zunächst wird mit dem DEFINE-Befehl ein neues Feld definiert. In Faktor1 steht „*LIKE“, das bedeutet, wir kopieren uns ein bereits vorhandenes Feld („is like“). Faktor2 gibt an, welches Feld wir kopieren wollen (PERID kommt aus PERPF), und im Ergebnis steht der Name des neuen Feldes. Danach definieren wir uns die Keyliste „kyPERREC“. Unter dieser Keyliste wird nur ein Keyfeld namens „kyPERID“ definiert. Es macht auf jeden Fall Sinn, für jedes Feld, was man in einer KLIST-Definition verwendet, vorher mit DEFINE eine Kopie zu erzeugen. Man könnte auch direkt als KFLD den Namen eins in der Datei vorkommenden Feldes verwenden, allerdings kommt man dann etwas durcheinander, wenn nach erfolglosem Lesen eventuell das Feld nicht richtig belegt ist.

Ich benenne Keylisten und Keyfelder immer ky... (wobei der Rest des Namens dem Satzformat oder Feld entstammt). Das ist eine Angewohnheit. Wer seine eigenen Regeln verwenden will, soll sich nicht abhalten lassen :)

Lass Dich nicht verschachteln!

Unsere Prüfroutine für die Buchungsperiode sieht inklusive Suchen so aus:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0028.05  * ?PERIODE?
0028.06 C                   IF        #B1OK = 'J'
0028.07 C                   IF        B1PERID = *blanks
0028.08 C                   EVAL      #B1OK = 'N'
0028.09 C                   EVAL      F1MELDUNG = 'Periode angeben!'
0028.10 C                   EVAL      B1PERTXT = *blanks
0028.11 C                   ELSE
0028.12 C                   EVAL      kyPERID = B1PERID
0028.13 C     kyPERREC      CHAIN     PERPF                              99
0028.14 C                   IF        *in99 = *OFF
0028.15 C                   EVAL      B1PERTXT = PERTXT
0028.16 C                   ELSE
0028.17 C                   EVAL      B1PERTXT = *blanks
0028.18 C                   EVAL      F1MELDUNG = 'Periode unbekannt!'
0028.19 C                   EVAL      #B1OK = 'N'
0028.20 C                   ENDIF
0028.21 C                   ENDIF
0028.22 C                   ENDIF
0029.00 C                   ENDSR
```

Du siehst, es wird schnell interessant. Schon drei ineinander verschachtelte `IF`-Abfragen. Das kann man bestimmt noch etwas schöner gestalten. Daher habe ich mal etwas Farbe eingebaut.

Rote Linie: Die gesamte Prüfung wird nur durchgeführt, wenn bisher keine Fehler gefunden wurden. Macht zwar bei der ersten Prüfung keinen Sinn, die anderen Prüfungen werden aber genauso abgekapselt.

Blaue Linie: Entweder ist das Eingabefeld leer, dann wird gemeckert, dass dies nicht erlaubt sei. Ansonsten wird geprüft, ob die Eingabe einge gültige Periode enthält. Hierzu wird das Schlüsselfeld aus unserer Keyliste belegt, und dann unter Angabe dieser Keyliste mit dem Befehl `CHAIN` in der Datei gesucht. Beachte dort die Angabe der Anzeigevariablen 99 in der Spalte „Ho“. `CHAIN` setzt diese auf `*AN`, wenn kein passender Satz gefunden wurde.

Grüne Linie: Wenn ein Satz gefunden wurde, das Textfeld belegen, ansonsten Fehlermeldung.

Probiere die neue Routine mal aus, das Ergebnis ist ansehnlich. Ist natürlich einiges an Tipparbeit, und mit noch etwas mehr Aufwand kann man sich den Aufwand sparen (grins, gute Formulierung?). Nein, es gibt komplexere Lösungen, so dass man das Prüfen vieler Felder aus vielen Dateien in einer Routine erledigen kann. Aber das lohnt sich nicht für unser kleines Programm.

Damit die Prüfroutine nicht ins Extreme gerät, prüfen wir die Felder Buchungsbeleg, Belegdatum und Buchungswert nur auf Inhalt. Der Matchcode für die Kategorie sowie das Buchungskonto müssen in den entsprechenden Dateien vorhanden sein. Das Feld Buchungstext muss nicht gefüllt sein.

Also brauchen wir noch die Dateien für Kategorie und Konto:

```
FMT D   DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter+++++++++++++++++
0001.02 FKATPF     IF   E           K DISK
0001.03 FKONTPF    IF   E           K DISK
```

Dateidefinitionen an die F-Zeilen anfügen!

Sowie passende Keylisten:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0002.13 C     *LIKE          DEFINE    KATMATC        kyKATMATC
0002.14 C     kyKATREC       KLIST
0002.15 C                    KFLD                     kyKATMATC
0002.16
0002.18 C     *LIKE          DEFINE    KONTID         kyKONTID
0002.19 C     kyKONTREC      KLIST
0002.20 C                    KFLD                     kyKONTID
```

Die Keylisten auch am Anfang der C-Zeilen einfügen

Wenn wir das haben, können wir unsere Prüfroutine vervollständigen (viele Zeilen!). Los geht's:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0028.23
0028.24  * ?BUCHUNGSBELEG?
0028.25 C                    IF        #B1OK = 'J'
0028.26 C                    IF        B1BEL = *blanks
0028.27 C                    EVAL      #B1OK = 'N'
0028.28 C                    EVAL      F1MELDUNG = 'Belegnummer angeben!'
0028.29 C                    ENDIF
0028.30 C                    ENDIF
0028.31
0028.32  * ?BELEGDATUM?
0028.34 C                    IF        #B1OK = 'J'
0028.35 C                    IF        B1BDAT = *blanks
0028.36 C                    EVAL      #B1OK = 'N'
0028.37 C                    EVAL      F1MELDUNG = 'Belegdatum angeben!'
0028.38 C                    ENDIF
0028.39 C                    ENDIF
0028.40
0028.31
0028.32  * ?BELEGDATUM?
0028.34 C                    IF        #B1OK = 'J'
0028.35 C                    IF        B1BDAT = *blanks
0028.36 C                    EVAL      #B1OK = 'N'
0028.37 C                    EVAL      F1MELDUNG = 'Belegdatum angeben!'
0028.38 C                    ENDIF
0028.39 C                    ENDIF
```

Prüfung der einzelnen Felder auf Inhalt und Sinn

Keine Sorge, die Programme stehen auf meiner Maschine bereit, und ich werde auch die Quelltexte zum Herunterladen anbieten. Bevor ich Beschwerden wegen wunder Finger bekomme... Es wird auch einen Anhang geben, wo die Programme (fast) kommentarlos am Stück abgebildet werden.

Hier geht es weiter mit den Prüfungen:

```
0028.41  * ?KATEGORIE?
0028.42 C                   IF        #B1OK = 'J'
0028.43 C                   IF        B1KATMATC = *blanks
0028.44 C                   EVAL      #B1OK = 'N'
0028.45 C                   EVAL      F1MELDUNG = 'Kategorie angeben!'
0028.46 C                   EVAL      B1KATTXT = *blanks
0028.47 C                   ELSE
0028.48 C                   EVAL      kyKATMATC = B1KATMATC
0028.49 C     kyKATREC      CHAIN     KATPF                              99
0028.50 C                   IF        *in99 = *OFF
0028.51 C                   EVAL      B1KATTXT = KATTXT
0028.52 C                   EVAL      B1KATTYP = KATTYP
0028.53 C                   ELSE
0028.54 C                   EVAL      B1KATTXT = *blanks
0028.55 C                   EVAL      B1KATTYP = *blanks
0028.56 C                   EVAL      F1MELDUNG = 'Kategorie unbekannt!'
0028.57 C                   EVAL      #B1OK = 'N'
0028.58 C                   ENDIF
0028.59 C                   ENDIF
0028.60 C                   ENDIF
```

Fortsetzung der Prüfroutinen. Bitte die Verschachtelungen zu entschuldigen

Für die Profis und alten Hasen unter uns:
Man könnte die Bezugszahl am `CHAIN`-Befehl auch weg lassen und folgendes schreiben:

```
 C     kyPERREC      CHAIN     PERPF
 C                   IF        %found(PERPF)
 C                   EVAL      B1PERTXT = PERTXT
...
```

Wie man ohne Bezugszahl prüft

Das sieht etwas freundlicher aus, vor allem kommt man so von den (etwas altmodischen) Bezugszahlen weg. Allerdings habe ich unter V4R5 schon gelegentlich Probleme mit `%FOUND()` gehabt und von anderen Programmierern gehört. Man sollte es also mit einer gewissen Vorsicht verwenden! Ich überlasse es Dir, in diesem Programm die nötigen Änderungen durchzuführen :-)

Wie die Routine für den Buchungsbetrag aussieht, kannst Du Dir bestimmt schon denken:

```
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0028.62  * ?BUCHWERT?
0028.63 C                   IF        #B1OK = 'J'
0028.64 C                   IF        B1BUCHWERT = *zero
0028.65 C                   EVAL      #B1OK = 'N'
0028.66 C                   EVAL      F1MELDUNG = 'Buchungswert angeben!'
0028.67 C                   ENDIF
0028.68 C                   ENDIF
```

Prüfung des Buchungswerts

Man beachte, dass ich in Zeile `0028.64` nicht auf `0` sondern auf `*ZERO` prüfe. Hat beides das gleiche Ergebnis, aber `*ZERO` liest sich meines Erachtens besser.

Den Rest muss ich bestimmt nicht weiter kommentieren. Ebenso wenig wie die folgenden Zeilen. Hier der Rest der Kontoprüfung sowie Abschluss der gesamten Prüfroutine:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0028.70  * ?KONTO?
0028.71 C                   IF        #B1OK = 'J'
0028.72 C                   IF        B1KONTID = *blanks
0028.73 C                   EVAL      #B1OK = 'N'
0028.74 C                   EVAL      F1MELDUNG = 'Konto angeben!'
0028.75 C                   EVAL      B1KONTTXT = *blanks
0028.76 C                   ELSE
0028.77 C                   EVAL      kyKONTID = B1KONTID
0028.78 C     kyKONTREC     CHAIN     KONTPF
0028.79 C                   IF        %FOUND(KONTPF)
0028.80 C                   EVAL      B1KONTTXT = KONTTXT
0028.81 C                   ELSE
0028.82 C                   EVAL      B1KONTTXT = *blanks
0028.83 C                   EVAL      F1MELDUNG = 'Konto unbekannt!'
0028.84 C                   EVAL      #B1OK = 'N'
0028.85 C                   ENDIF
0028.86 C                   ENDIF
0028.87 C                   ENDIF
0028.88
0028.89  *ENDE
0029.00 C                   ENDSR
```

Du siehst, es ist hauptsächlich Tipparbeit. Wenn Du in einem Programm viele Felder hast, die Du gegen viele Dateien prüfen musst, wird das schon wirklich viel. Und es wird unflexibel. Bei grossen Projekten erstellt man zum Beispiel eine Art Schlüsseldatei. Diese Datei stellt eine Sammlung aller möglichen Werte aller Dateien dar, und man kann die Prüfroutine vereinfachen, wenn man nur noch auf diese Daten prüft. Allerdings muss diese Schlüsseldatei immer stimmig sein.
Wie man das macht.... das wird ein eigenes Kapitel, denn es ist nicht ohne!

Nun haben wir also auch unsere Prüfroutine fertig. Diese bauen wir nun in den Hauptteil des Programms ein. Du erinnerst Dich an meine Ausführungen einige Seiten vorher, an unsere DO-Schleife? Fein, nach dem Aufruf von CHKF1 (wo wir die F3-Taste überwachen), folgt nun dieses hier:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0014.02  * Funktionstasten prüfen
0014.03 C                   EXSR      CHKF1
0014.04
0014.05  * Daten prüfen
0014.06 C                   IF        #B1ENDE = 'N'
0014.07 C                   EXSR      CHKEIN1
0014.08 C                   ENDIF
```

Es macht ja keinen Sinn, nochmal alle Felder zu prüfen, wenn der Benutzer F3 zum Verlassen unseres Programms gedrückt hat (sparen wir der CPU Arbeit). Daher die Abfrage.

Nun teste Dein Programm bis hierhin. Wenn es zu Deiner Zufriedenheit funktioniert, fehlt nur noch das Wichtigste: Das Abspeichern der eingegebenen Daten.
Wie das aussieht? Lies' weiter!

Eindeutiges Schreiben

Unsere Prüfroutine stellt also sicher, dass die Daten (in gewissem Rahmen) brauchbar sind. Daher müssen wir uns in der Schreibroutione nicht darum kümmern.
Diese Schreibroutine hat die Aufgabe, einen neuen Satz in unserer Datei BUCHPF anzulegen. Dieser Satz muss aber vollständig sein. Unsere Datei enthält das Feld BUCHID, das eine eindeutige Zahl für jeden Satz in der Datei enthalten soll[I].

Es gibt nun verschiedene Ansätze, wie man sicherstellt, dass dieses Feld einen eindeutigen Wert enthält. In SQL gibt es beispielsweise Möglichkeiten, dass das System selbst einen numerischen Feldinhalt hochzählt, oder man benutzt Identities, die in OS/400 ab V5R2 implementiert wurden. Man kann (oder muss) sich auch selbst um die Eindeutigkeit kümmern. Es folgt ein klassisches Beispiel in RPG, nicht perfekt, aber fürs Erste ganz interessant, und Du wirst als RPG-Programmierer in vielen alten Programmen diese Methodik finden. Und später lernst Du auch, wie man das dann anders macht.

Die Erhöhung unseres eindeutigen Feldwertes geschieht ganz einfach: Wir suchen (sortiert nach dem fraglichen Feld BUCHID) den letzten Satz in der Datei. Dieser enthält (aufgrund unserer Sortierung) den höchsten bisher verwendeten Wert. Diesen erhöhen wir um eins und verwenden ihn für unseren neuen Satz.
Und wenn es noch gar keinen Satz in unserer Datei BUCHPF gibt? Fangen wir eben mit Eins an...

Ich höre die Zwischenrufe der mitlesenden Profis. Ja, das ist unelegant, fehlerträchtig und überhaupt... Aber lasst uns bei den einfachen Methoden anfangen. Es soll der unbedarfte Leser nicht sofort erschlagen werden.

Beginnen wir also eine neue Unterroutine:

```
 FMT C  CL0N01Factor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0032.00  * SCHREIBE: Satz wegschreiben
0033.00 C     SCHREIBE      BEGSR
0033.07 C                   EVAL      kyBUCHID = *HIVAL
0033.08 C     kyBUCHREC     SETLL     BUCHPF
0033.09 C                   READP     BUCHPF                                 99
0033.11  * Wenn wir einen Satz finden (*IN99 = *OFF) -> Satznummer erhöhen
0033.12  * Ansonsten mit Satznummer 1 anfangen.
0033.13  * (alte, unelegante, aber lehrreiche Methode)
0033.14 C                   IF        *IN99 = *OFF
0033.15 C                   EVAL      kyBUCHID = BUCHID + 1
0033.16 C                   ELSE
0033.17 C                   EVAL      kyBUCHID = 1
0033.18 C                   ENDIF
```

Zunäst belegen wir unser Keyfeld (mit dem wir in der Datei suchen, siehe oben) mit dem Wert „*HIVAL“. Das bedeutet unter RPG: „benutze den für dieses Feld höchstmöglichen Wert!“. Hätte das Feld kyBUCHID eine Definition von „5stellig ohne Nachkommastellen“, wäre das dann 99999. Machen wir dies mit einem Feld von 8 Stellen, davon zwei Nachkommastellen, wäre es dann 999999,99. Für Neugierige: „*HIVAL“ hat ein Gegenstück, das sich „*LOVAL“ nennt.
In Zeile 33.08 sehen wir einen neuen Befehl namens SETLL. Ausgeschrieben heisst er „Set lower limit“. Damit sagen wir dem System, es soll in der Datei BUCHPF den Dateizeiger auf den ersten Satz stellen, in dessen Schlüsselfelder mindestens die Werte haben wie im Schlüssel kyBUCHREC angegeben. Da das dort definierte, einzige Schlüsselfeld den Inhalt *HIVAL hat, bedeutet das, wir landen auf dem letzten Satz der Datei, besser ausgedrückt, wir springen somit ans Ende der Datei. SETLL liest aber nicht, sondern positioniert nur. Daher liefert SETLL auch keinen Fehler.

[I]Wir brauchen dieses Feld nicht zwingend in diesem Programm, aber ich habe den Fall mit eingebaut, zu Lernzwecken...

Ach ja, bevor ich es vergesse, hier die Definition für unseren Key:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0002.23 C     *LIKE          DEFINE    BUCHID         kyBUCHID
0002.24 C     kyBUCHREC      KLIST
0002.25 C                    KFLD                     kyBUCHID
```

So, nun sind wir nach dem `SETLL` am Ende, äh, unserer Datei. Was nun? Es folgt der Befehl `READP`. Ausgesprochen „Read previous“ (lies vorherigen). Dieser Befehl liest also – ausgehend vom aktuellen Standort des Dateizeigers – den vorigen Satz.

Nach unserer Logik also den Satz mit dem höchsten Vorkommnis von `BUCHID`. Verwirrt? Naja, nicht so wirklich, man muss nur physikalisch denken. Als SQLer würde man vielleicht schreiben:

```
SELECT MAX(BUCHID) FROM BUCHPF
```

und erhält auch den höchsten Wert in `BUCHID` aus der Datei. Der Optimizer würde dann genau das gleiche machen wie unser RPG-Programm: Einen Schlüssel auf das Feld `BUCHID` suchen, und den letzten Satz holen[I].

Unser `READP` setzt die Bezugszahl „Nicht gef.“ (hier: `99`) auf `*ON`, wenn kein Satz gelesen werden konnte, ergo kein Satz in der Datei existiert. Dann beginnen wir die Zählerei mit Eins, ansonsten erhöhen wir den Wert in `BUCHID` und zwischenspeichern diesen im nicht mehr benötigten Feld unserer Keyliste.

Im Grunde ist diese Methode also recht einfach. Problematisch kann es nur werden, wenn durch Zufall und Parallelverarbeitung zwei Jobs genau zum gleichen Zeitpunkt mit `SETLL` und `READP` den letzten Satz ermitteln. Wie man das umgeht: später!

Vervollständigen wir zunächst unsere Schreibroutine:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0033.19  * Felder belegen
0033.20 C                    CLEAR                    BUCHREC
0033.21 C                    EVAL      BUCHID     = kyBUCHID
0033.22 C                    EVAL      BUCHPER    = B1PERID
0033.23 C                    EVAL      BUCHBEL    = B1BEL
0033.24 C                    EVAL      BUCHKAT    = B1KATMATC
0033.25 C                    EVAL      BUCHTYP    = B1KATTYP
0033.26 C                    EVAL      BUCHTXT    = B1BUCHTXT
0033.27 C                    EVAL      BUCHKTO    = B1KONTID
0033.28 C                    EVAL      BUCHBDAT   = B1BDAT
0033.29 C                    EVAL      BUCHWERT   = B1BUCHWERT
0033.30 C                    IF        BUCHTYP = 'A'
0033.31 C                    EVAL      BUCHWERT = (BUCHWERT * -1)
0033.32 C                    ENDIF
0033.40 C                    WRITE     BUCHREC
0033.41  * Detailbild leeren
0033.42 C                    CLEAR                    BODY01REC
0033.43 C                    EVAL      F1MELDUNG = 'Satz wurde geschrieben.'
0033.44 C
0034.00 C                    ENDSR
```

Wir leeren also zunächst das Satzformat `BUCHREC` (da wir es vorher eventuell gelesen haben), belegen dessen Felder mit denen aus unserem Bildschirm. Danach wird `BUCHREC` geschrieben und das Satzformat unseres Bildschirms `BODY01REC` geleert.

[I]Wenn es für das gesuchte Feld keinen Index gibt, wird automatisch einer erstellt, zur Strafe :-)

Das interessante Konstrukt in den Zeilen 033.30 bis 033.32 hat Faulheitscharakter. Wenn der Buchungstyp eine Ausgabe ist, wird der Buchungswert negativ in der Datei abgelegt. Das vereinfacht uns später die Auswertung per SQL ungemein :-)

Abschliessend lassen wir uns eine sinnvolle Meldung für die Fusszeile einfallen und beenden die Subroutine.

Damit unsere Schreibroutine auch ausgeführt wird, müssen wir wieder zurückgehen in unsere Hauptschleife, und folgendes hinter den Aufruf unserer Prüfroutine CHKEIN1 einbauen:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0014.10  * Wenn alles ok, Satz schreiben!
0014.11 C                   IF        #B1ENDE = 'N'
0014.12 C                             AND #B1OK = 'J'
0014.13 C                   IF        *IN10 = *ON
0014.14 C                   EXSR      SCHREIBE
0014.15 C                   ELSE
0014.16 C                   EVAL      F1MELDUNG = 'Daten ok, bitte mit der +
0014.17 C                             Taste F10 abspeichern.'
0014.18 C                   ENDIF
0014.19 C                   ENDIF
```

Zunächst wird geprüft, dass nicht F3 gedrückt wurde (in dem Falle setzen wir ja #B1ENDE auf „J“), und sichergestellt, dass die Eingaben im Bildschirm brauchbar sind (#B1OK). Da wir erst abspeichern wollen, wenn der Anwender F10 gedrückt hat, prüfen wir diese Taste auch noch. Wenn *ON (also gedrückt), wird unsere Schreibroutine aufgerufen, ansonsten wird nur eine Meldung am Fuss ausgegeben.

Stop! Woher weiss das System überhaupt, dass wir F10 drücken dürfen? Klar: In unserer Bildschirmdatei BUCHDSP müssen wir in das Satzformat BODY01REC noch folgende Zeile einbauen:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0003.04      A                                      CF10(10)
```

Ein guter Platz wäre direkt unterhalb der Zeile für die F3-Taste.

Nun wandle zuerst die Bildschirmdatei, dann das Programm um.

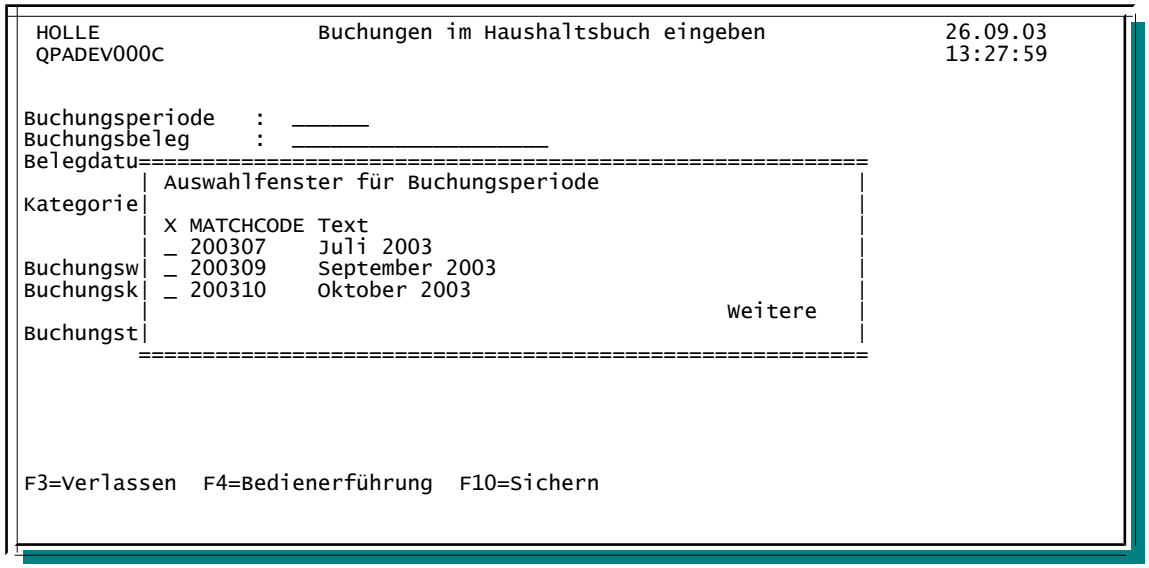
Experimentiere nun ein wenig mit dem Programm, und mach Dich mit den Befehlen vertraut. Vor allem aber mit den nötigen Schritten zur Prüfung der Felder, und dem Schreiben des Satzes. Es geht gleich hart weiter!

Das Programm ist schon mal prima. Wir können buchen wie die Profis und all unsere Haushaltskosten und Einnahmen hinterlegen. Aber! Hast Du alle Matchcodes für die Kategorien oder die Konten im Kopf? Vielleicht möchtest Du andere Kürzel für die Buchungsperioden verwenden, hast im Eifer des Gefechts aber vergessen, wie die gerade benötigte Periode heisst?

Hier kommt die Taste F4 ins Spiel. Du kennst sie von vielen Bildschirmen, zum Beispiel bei Befehlen kannst Du Dir damit die möglichen Werte für einen Parameter anzeigen lassen. Genau dieses wollen wir in unserem Haushaltsbuch-Programm verwenden.

Wenn Du mit dem Cursor auf dem Eingabefeld für die Buchungsperiode stehst und F4 drückst, soll eine Liste aller vorhandenen Periondeneinträge (aus der entsprechenden Datei) erscheinen. Darin soll der Benutzer blättern können, und bei Bedarf eine davon auswählen.

Klingt trivial, ist es aber nicht ganz. Ich stelle mir das optische Ergebnis ungefähr so vor:



Sieht in schwarzweiss nicht so schön aus, wie es nachher in Farbe erscheinen wird. Aber Du weisst Bescheid. Das oben gezeigte Fenster soll erscheinen, wenn im Feld „Buchungsperiode“ die Auswahltaste F4 gedrückt wird. In diesem Fenster werden alle in der Periodendatei vorhandenen Sätze angezeigt. Vor jedem Satz ist ein einstelliges Eingabefeld. Wenn der Anwender dort ein X oder eine 1 einträgt und mit DF bestätigt, soll der markierte Matchcode übernommen und im Hauptbildschirm eingetragen werden. Existieren in der Periodendatei mehr Sätze als in das Fenster passen, muss natürlich auch geblättert werden können!

Bevor Du Dir jetzt Gedanken über das Fenster und die ganzen Aus- und Eingabezeilen machst: Nicht so hastig, OS/400 nimmt Dir etwas Arbeit ab.

Grundlage für diesen Hilfebildschirm ist ein Fenster. Dieses kann man in einer Bildschirmdatei mit einem Befehl definieren. Viel interessanter sind die Zeilen, die später Matchcode und Text der Periode enthalten, und uns vor jedem Satz ein Eingabefeld präsentieren. Du könntest diese Zeilen per Hand programmieren...

Du kannst es aber auch bleiben lassen, denn es gibt:

Subfiles – Die Datei die keine ist?

Irgendjemand bei IBM muss einen guten Tag gehabt haben, als er Subfiles erfunden hat. Denn Subfiles machen dem (Bildschirm-)Programmierer das Leben leichter.

Denke mal an typische Verwaltungs- und Eingabeprogramme. Oft gibt es dort eine Situation, in der Du eine gewisse Menge gleichartiger Zeilen auf dem Bildschirm hast. Nehmen wir unser Beispiel im Hilfefenster. Die Periodendatei kann einige bis einige -zig Sätze enthalten. Diese sollen nach Möglichkeit auf dem Bildschirm präsentiert werden, aber nicht alle auf einmal, sondern immer nur einige.
Der Benutzer soll dann blättern, und einen gewünschten Satz markieren.

Subfiles ermöglichen noch mehr: man kann alle Felder als Ein-/Ausgabefelder deklarieren und den Benutzer dort Angaben machen lassen, man kann Zeilen ausblenden, Zeilen über mehrere Bildschirmzeilen erstrecken und... Stop! Es gibt ganze Bücher über Subfiles, also will ich hier zunächst nicht ausschweifen.

Subfiles musst Du Dir wie eine normale Datei vorstellen, ein Objekt, das mehrere gleichartige Sätze enthält. Allerdings wird diese Datei nicht irgendwo auf einer Platte oder einem Band gespeichert, sondern nur für die Darstellung und Bearbeitung auf dem Bildschirm temporär verwendet. Aber die Art und Weise, wie Du als Programmierer mit dem Subfile umgehst, ähnelt stark der Behandlung einer normalen Datei. Man kann Sätze lesen und schreiben, das Subfile leeren etc.

Wie wird ein Subfile definiert?

Ein Subfile besteht immer aus der Definition eines Steuersatzes und der Definition der Datensätze. Der Steuersatz (genauer: Das steuernde Satzformat in einer Bildschirmdatei) ist dafür da, generelle Verhaltensweisen des Subfiles festzulegen. Das Subfile selbst (genauer: Das Subfile-Format in einer Bildschirmdatei) enthält Informationen, wie die Datensätze aussehen sollen.

Wenn Du also ein Subfile auf dem Bildschirm haben willst, musst Du immer mindestens zwei weitere Satzformate in Deiner Bildschirmdatei unterbringen. Dabei werden schon fortgeschrittenere Angaben in der Bildschirmdatei benötigt, so dass die Definition und das Arbeiten mit einem Subfile am Anfang doch schon für Stirnrunzeln sorgt. Aber wenn man ein paar mal Subfiles programmiert hat, ist es gar nicht mehr so schlimm. Stell Dir einfach vor, Du müsstest das per Hand machen, das ist viel Schlimmer ;-)

Da unser Subfile aus zwei Satzformaten besteht, muss man im Grunde auch beide Satzformate auf den Bildschirm bringen. Aber weil wir ja ein steuerndes Satzformat haben, nimmt es uns etwas Arbeit ab.

Betrachte noch einmal den Bildschirm auf der vorigen Seite. Wir haben (innerhalb des Fensters) eine Titelzeile, einige Spaltenüberschriften sowie die Datenzeilen selbst. Titelzeile und Spaltenüberschriften gehören zum steuernden Satzformat, die Datenzeilen in das Subfile-Satzformat. Das ist schon mal die halbe Miete.

Bauen wir uns das Subfile Schritt für Schritt auf. Ich zeige hier eine einfache, aber schon etwas verwirrende Version unseres Steuerformates und erkläre die einzelnen Zeilen:

Schau Dir einfach die folgenden Zeilen aus der Bildschirmdatei BUCHDSP kurz an und folge dann den Beschreibungen:

Fangen wir zunächst mit dem Anfang des Subfile selbst an, es ist einfach und übersichtlich:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0110.00      A          R SFL1                          SFL
0111.00      A            #SFL1RN         9S 0H
0112.00      A            #SFL1WAHL       1A  B  4  2VALUES('X' '1' ' ')
0113.00      A            #SFL1MATC       4A  O  4  4
0114.00      A            #SFL1TEXT      40A  O  4 12
```

Unser Subfile – simpel und einfach

Wir definieren also in unserer Bildschirmdatei ein neues Satzformat namens SFL1. Dies wird in Zeile 0110.00 durch SFL in der Funktionsspalte dem Compiler deutlich gemacht.

Nun folgen vier Felder. Vier? Moment! Auf dem Bildschirm haben wir doch nur drei Felder definiert (Auswahlfeld, Matchcode und Text). Ein Subfile benötigt aber zur Steuerung beim Lesen und Schreiben noch eine Satznummer (wie jede normale Datenbankdatei auch immer eine Satznummer hat, auch wenn man sie nicht immer sieht). Diese Satznummer wird in einem versteckten Feld innerhalb des Subfiles abgelegt.

Schau Dir Zeile 0111.00 an. Dort definiere ich das Feld #SFL1RN neunstellig numerisch (Typ 'S') ohne Nachkommastellen. Als Feldart ist hier „H“ für „hidden“ (versteckt) eingetragen. Somit existiert das Feld zwar, wird aber nie angezeigt. Du als Programmierer kannst aber ganz normal damit arbeiten.

Danach folgt in 0112.00 das Feld #SFL1WAHL, welches ein einstelliges Alphafeld darstellt, das die Auswahl des Benutzers („Ja, die Zeile meinte ich!“) enthält. Daher als Feldart auch „B“ für „both“ (beides, Ein- und Ausgabe). Zusätzlich habe ich im Funktionsbereich definiert, welche Werte hier überhaupt erlaubt sind. Dabei haben sich „X“ und „1“ als häufig verwendet herausgestellt. Das Leerzeichen muss auch unbedingt als gültige Eingabe definiert sein, sonst würde das System später bei jeder Zeile im Subfile, in der keine Eingabe erfolgte, dieses Feld anmeckern!

Die beiden anderen Felder #SFL1MATC und #SFL1TEXT sind recht einfach erklärt, es sind zwei Ausgabefelder. Punkt.

Die Zeilennummer „4“ (Spalte „Zei“) unserer drei Felder haben ihren Grund: Ich werde das Subfile (im Steuerformat) so definieren, dass es in einem Fenster erscheint. Innerhalb dieses Fensters müssen wir relativ die Positionen angeben, also beginnend mit der oberen, linken Position innerhalb des Fensters. Daher fangen die Daten des Subfiles in Zeile 4 an, da darüber Überschrift und Titel erscheinen sollen (Zeile 2 und 3).

Ein Hinweis zur Zeilennummer: Wir definieren gleich den Rest für das Subfile, darunter auch ein Fenster. Unser Subfile soll ja nicht ständig auf dem Bildschirm stehen. Also wird bei Bedarf ein Fenster geöffnet. Alle Positionierungsangaben müssen somit relativ innerhalb des Fensters angegeben werden. Daher verwende ich auch die Zeile 4, die nichts mit der absoluten Zeile auf dem Bildschirm zu tun hat.

Aus mehr besteht unser Subfile nicht. Die Arbeit steckt im steuernden Subfileformat. Schau Dir mal das hier an:

```
 FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0116.00      A          R CTLSFL1                     SFLCTL(SFL1)
0117.00      A                                        CF03(03)
0118.00      A                                        CF12(12)
0119.00      A                                        OVERLAY
0120.00      A  91                                    SFLINZ
0121.00      A  93                                    SFLCLR
0122.00      A N93                                    SFLDSP
0123.00      A N93                                    SFLDSPCTL
0124.00      A                                        SFLSIZ(0999)
0125.00      A                                        SFLPAG(0007)
0125.10      A N93                                    SFLEND(*MORE)
0126.00      A                                        WINDOW(6 5 12 68)
0127.00      A            S1TITEL       40A  O  1  2COLOR(WHT)
0128.00      A                                   3  2'X'
0129.00      A                                      COLOR(WHT)
0130.00      A                                   3  4'MATCH'
0131.00      A                                      COLOR(WHT)
0132.00      A                                   3 12'Beschreibungstext'
0133.00      A                                      COLOR(WHT)
```

Das Steuernde Satzformat für unser Subfile

Sieht schon mal „interessant“ aus, oder?[I] Lass Dich nicht von den vielen Parametern und unbekannten Kürzeln abhalten. Es hat alles seinen Sinn.

Fangen wir einfach (wie gewohnt) oben an:

In Zeile `0116.00` beginnt ein neues Satzformat in der Bildschirmdatei. Ich benenne steuernde Satzformate für Subfiles immer `CTLxxxx`, wobei `xxxx` gleich `SFL` und eine Nummerierung darstellt. Zusätzlich wird in der Funktionsspalte mit `SFLCTL()` angegeben, dass das hier definierte Satzformat `CTLSFL1` ein Steuersatzformat für das Subfile SFL1 ist. Dieses muss (wie geschehen) *vor* dem Steuerformat definiert sein[II]! Die Reihenfolge in der DDS-Datei ist hier also wichtig. Zunächst definierst Du das Subfile, und danach (am besten direkt in Folge) das dazugehörige Steuersatzformat.

Die Zeilen `0117` und `0118` sollten bekannt vorkommen. Hier definierst Du, dass die Funktionstasten `3` und `12` erlaubt sind, und gibst in Klammern an, welche Bezugszahl[III] entsprechend gesetzt werden soll.

Auch die Funktion `OVERLAY` in der nächsten Zeile sollte kein Zauberwerk mehr für Dich darstellen. Es besagt nur, dass dieses Satzformat (also unser Fenster mit dem Subfile) dargestellt werden soll, ohne dass der bereits vorhandene Bildschirminhalt gelöscht wird.

[I] Man könnte auch sagen: wüst, ein Subfile benötigt aber immer einige grundlegende Angaben...
[II] Das ist ganz wichtig. Ich habe schon merkwürdige Effekte gesehen, wenn man Subfile-Formate in falscher Reihenfolge oder unvollständig definiert, also gilt: Aufpassen!
[III] Bezugszahl/Anzeigevariable/Indikator meinen immer das gleiche. Nicht verwirren lassen, ich will nur zur Auflockerung mal ein anderes Wort verwenden.

Interessant wird es jetzt in Zeile 0120.00. Hier siehst Du gleich links in der Zeile den Begriff „91“. Wenn Du Dir die Formatierungszeile (Typ: DP) oben anschaust, siehst Du:

```
FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
```

Formatierungszeile DP

Das erste A steht für die Zeilenart, in der Du immer ein ***A*** einträgst. Danach folgt ein weiteres A, welches ich vorerst ignoriere. Nun folgen drei Gruppen N01, N02 und N03.

Diese drei Gruppen dienen dazu, über Bezugszahlen diese Zeile zu konditionieren, also bedingt zu steuern. In unserer ersten Zeile steht unter „01“ die Bezugszahl „91“. Wenn in einem der drei beschriebenen Gruppenfelder eine Bezugszahl eingetragen wird, so wird die betreffende Zeile nur dann ausgeführt, wenn diese Anzeigevariable auf *ON steht. Stellt man ein N vor die Bezugszahl in die passende Spalte, muss diese auf *OFF stehen, damit die Zeile beachtet wird. In unserem Fall gilt also, dass die Funktion SFLINZ nur ausgeführt wird, wenn die Bezugszahl 91 auf *ON steht.
Somit kann man über Bezugszahlen das Verhalten von Bildschirm- und Druckerdateien steuern.

Da es *drei* Gruppen an jedem Zeilenanfang gibt, kannst Du auch bis zu drei Bezugszahlen angeben. Diese drei Angaben müssen zutreffen, damit die Zeile beachtet wird. Beispiel:

```
FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
            A  30 31N32                              SFLDSP
```

Konditionierung einer Zeile mit 3 Bezugszahlen

Diese Zeile wird nur beachtet, wenn Bz. 30 und 31 eingeschaltet sind (kein N davor), *und* Bz. 32 ausgeschaltet ist (N davor). Schon verwirrend, denn bei intensivem Einsatz dieser Möglichkeit kann man jede Bildschirmdatei unleserlich machen!

Damit es nicht langweilig wird, kann man auch noch mehrere Zeilen zu einer logischen Zeile verknüpfen. Dafür ist das zweite A (also links von der ersten Gruppe) zuständig. Noch ein herbes Beispiel:

```
FMT DP .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
            A  30 31N32
            AAN42 43                                 SFLDSP
```

So sollte man nicht programmieren

Diese Zeile (die aus zwei Zeilen besteht) wird nur beachtet, wenn Bz. 30, 31, 43 auf *ON stehen, sowie Bz. 32 und 42 auf *OFF. Es gibt Programmierer, die früher fast jede Zeile so konditioniert haben, besonders in viel genutzten Druckdateien. Nur kein Mensch weiss, was diese Programme machen. Also: Sparsam einsetzen!

Jetzt habe ich Dich genug verwirrt...

Zurück zu unserem Programm. In Zeile 0120 wird also die Funktion SFLINZ nur ausgeführt, wenn die Bezugszahl 91 eingeschaltet ist. Warum, siehst Du gleich. Aber was macht eigentlich SFLINZ? Naja, ganz einfach, es steht für **SubFiLeINitialiZe**. Wir initialisieren also unser Subfile am Anfang. Macht Sinn!

Analog auch in Zeile 0121: Die Funktion SFLCLR besagt, dass alle Sätze in der Subdatei gelöscht werden sollen und wird nur ausgeführt, wenn Bezugszahl *IN93 eingeschaltet ist.

In der nächsten Zeile `0122` wird die Funktion `SFLDSP` nur ausgeführt, wenn `*IN93` ausgeschaltet ist. `SFLDSP` stellt das Subfile auf dem Bildschirm dar.

Zeile `0123` schliesslich, die ebenfalls nur aufgerufen wird, wenn `*IN93` ausgeschaltet ist, zeigt auch noch die Felder des steuernden Satzformates dar.

Somit haben wir vier wichtige Funktionen definiert, die man bei jeder Subdatei benötigt. Dass sie mit Bezugszahlen konditioniert sind, ist wichtig, und Du solltest Dir auch die Reihenfolge beim Aufbau eines Subfiles merken:

1) Subfile initialisieren
2) Subfileinhalt löschen
3) Subfile eventuell nach Wunsch mit Daten füllen (im Programm)
4) Subfile anzeigen und steuerndes Satzformat anzeigen
5) Eingaben aus dem Subfile abfragen

Nachdem wir schon mal ein paar Grundlagen des Subfiles festgelegt haben, müssen jetzt noch ein paar optische Aspekte eingerichtet werden.

Man werfe bitte einen Blick auf die Zeilen `0124` und `0125`. Hier geben wir an, wieviele Sätze maximal im gesamten Subfile enthalten sein dürfen (**`SubFiLeSIZe`**), und wieviele Sätze gleichzeitig auf einer Seite angezeigt werden sollen (**`SubFiLePAGe`**).

Den ersteren Parameter solltest Du zur Schonung der Systemressourcen gut bedenken. Wenn Du ausschliessen kannst, dass in dem Subfile jemals mehr als 50 Einträge enthalten sein können, gib hier einen entsprechenden Wert an. OS/400 muss ja für jeden Anwender und jedes Subfile Speicher reservieren, und wenn nun 1000 Anwender Dein Programm verwenden, kann es sein, dass sinnlos Speicherbereiche belegt werden.

Die Angabe der Seitengrösse gehört zum Design und hängt von unserer Fenstergrösse ab. Die Steuerung übernimmt OS/400 automatisch, wenn Dein Subfile mehr Sätze hat als in `SFLPAG()` angegeben, kann der Anwender blättern.

Dass der Anwender dies kann, wird ihm in der Zeile `0125.10` mitgeteilt. Die Funktion `SFLEND()`, die zwingend durch eine Bezugszahl konditioniert werden muss, teilt dem System mit, wie dem Anwender gezeigt werden soll, dass es weitere Sätze im Subfile gibt. `*MORE` besagt, dass das System sprachabhängig „`Weitere...`“ anzeigen soll. Du kannst diese Zeile ja testweise mal weg lassen.

Schliesslich, letztlich und fehlenderweise haben wir in Zeile `0126.00` nun noch unser Fenster, in dem wir unser gesamtes Subfile darstellen wollen. Die Funktion `WINDOW()` hat diverse Parameter, wir verwenden hier nur die Zeile und Spalte auf dem Bildschirm, an dem die linke obere Ecke des Fensters liegen soll (hier: Zeile 6, Spalte 5), sowie die Höhe des Fensters in Zeilen und die Breite in Spalten.

So, nun haben wir also eine Menge Arbeit hinter uns, Dein Kopf raucht, und es fehlen nur noch wenige Schritte. Die restlichen Zeilen definieren die Überschriften des Steuersatzformates. Beachte bei der Positionierung von Zeile und Spalte, dass dies innerhalb des definierten Fensters gilt. Wir verwenden hier die Zeilen 1 und 3, nicht absolut auf dem Bildschirm, sondern vom Fenster.

Wie sieht das Subfile aus?

Nun ist die Definition unseres Subfiles fertig, und Du kannst es kaum erwarten, das Ding auf dem Bildschirm zu sehen. Langsam reiten, Cowboy! Bis wir unser Subfile gefüllt haben, geht auch noch etwas Zeit ins Land. Damit Du nicht ganz ungeduldig wirst, fügen wir hier einen kleinen Subfile-Test ein, damit Du auch Ergebnisse siehst:

Speichere Deine Bildschirmdatei `BUCHDSP` ab, wandele sie mit `14` um und korrigiere die Tippfehler.

Dann erstelle in `QRPGLESRC` das folgende RPGLE-Programm namens `SFLTEST01`:

```
 Spalten . . . :    6  76          Editieren                    HHBUCH/QRPGLESRC
 SEU==>                                                             SFLTEST01
 FMT FX FDateiname+IPEASF.....L.....A.E/AEinh.Schlüsselwörter+++++++++++++++++++
        ********** Datenanfang *****************************************
0001.00 FBUCHDSP   CF   E             WORKSTN SFILE(SFL1:#SFL1RN)
0001.10
FMT C   CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0003.00 C                   EVAL      #SFL1RN = 0
0004.00 C                   EVAL      *IN91 = *ON
0005.00 C                   WRITE     CTLSFL1
0006.00 C                   EVAL      *IN91 = *OFF
0007.00 C                   EVAL      *IN93 = *ON
0008.00 C                   WRITE     CTLSFL1
0009.00 C                   EVAL      *IN93 = *OFF
0010.00
0010.10 C                   DOW       #SFL1RN < 30
0011.00 C                   EVAL      #SFL1RN = #SFL1RN + 1
0012.00 C                   EVAL      #SFL1MATC = %CHAR(#SFL1RN)
0013.00 C                   EVAL      #SFL1TEXT = 'eine Zeile'
0014.00 C                   WRITE     SFL1
0014.10 C                   ENDDO
0015.00
0016.00 C                   EXFMT     CTLSFL1
0017.00
0018.00 C                   EVAL      *INLR = *ON
```

Unser kleiner Subfiletest

Dieses Programm benutzt unser Subfile, füllt es mit Inhalt und zeigt es an.

Experimentiere damit mal ein wenig rum, und schau Dir genau an, was die einzelnen `WRITE`-Befehle machen und in welcher Reihenfolge sie aufgerufen werden.

Bei der Dateidefinition in Zeile `0001` siehst Du, dass wir im Bereich der Schlüsselwörter angeben müssen, dass wir ein Subfile verwenden. Ohne diese Angabe bekommst Du Probleme beim Zugriff auf einzelne Sätze im Subfile. Als Parameter musst Du den Subfilenamen und die Variable angeben, die über ihren Inhalt den Zugriff auf einzelne Sätze steuert. Du erinnerst Dich, diese Variable haben wir im Satzformat `SFL1` definiert.

In Zeile `0004` und `0005` wird das Subfile initialisiert (über `*IN91`) .

Zeilen `0007` und `0008` löschen alle Sätze, die im Subfile sein mögen.

Zeile `0010` bis `0015` stellen eine kleine Schleife da, um 30 Sätze in das Subfile zu bringen. Für jeden Satz musst Du die Felder belegen und in die Subdatei `SFL1` schreiben.

Nachdem wir dies gemacht haben, wird das Subfile mit `EXFMT` auf den Bildschirm gebracht.

So, und nun: Ausprobieren und erfreuen!

Nachdem Du nun weisst, wie das Subfile definiert wird und wie es später ungefähr aussieht, geht es nun an den aufwändigen Teil: F-Taste abfragen und je nach Feld das Subfile füllen und eine Auswahl des Benutzers auswerten. Dazu braucht es aber noch einiges an Vorbereitung.

F4 - Was darfs denn sein?

Zunächst einmal müssen wir in unserem Hauptbildschirm die Taste F4 erlauben und abfragen. Dann müssen wir feststellen, wo sich der Cursor befand, als der Anwender die F4 Taste gedrückt hat. Man könnte hier die Cursorposition abfragen, aber das macht keinen Sinn. OS/400 erleichtert es uns ein wenig, in dem wir vom Displayfile den Feldnamen zurückerhalten können.

Editiere das Displayfile BUCHDSP. Springe in das Satzformat BODY01REC. Baue die folgend rot markierten Zeilen ein:

```
FMT DP  .....AAN01N02N03A.Name++++++RLängeDDsFZeiPosFunktionen+++++++++++++++++
0035.00      A          R BODY01REC
0036.00      A                                      RTNCSRLOC(*RECNAME +
0037.00      A                                      &#CSRREC &#CSRFLD  +
0038.00      A                                      &#CSRPOS)
0039.00      A                                      BLINK
0040.00      A                                      OVERLAY
0041.00      A                                      CF03(03)
0042.00      A                                      CF04(04)
0043.00      A                                      CF10(10)
0044.00      A            #CSRREC       10A  H
0045.00      A            #CSRFLD       10A  H
0046.00      A            #CSRPOS        4S 0H
0047.00      A                                   5  1'Buchungsperiode   :'
```

So erhalten wir die Cursorposition von unserer Bildschirmdatei

Die System-Funktion RTNCSRLOC() liefert die Position und das aktuelle Feld des Cursors zurück, nachdem das Satzformat verlassen wurde (sei es durch eine Funktionstaste oder DF). Die drei in diesem Aufruf verwendeten Felder müssen wir aber auch im Satzformat definieren (siehe unten).
Übrigens siehst Du hier schön, wie man mit dem Pluszeichen eine Zeile verlängern kann, wenn die Angaben im Funktionsbereich nicht in eine Zeile passen.
Schliesslich muss noch die Taste F4 erlaubt werden.
Voilà, schon können wir in unserem Programm etwas damit anfangen! Bevor ich aber jetzt einzelne Schritte erkläre und Du nur noch im Quellcode hin- und herspringst, ohne mitzukommen, werde ich einen vorbereitenden Block bringen.

Wir werden die F4 Funktionalität für drei Felder einrichten:

- Buchungsperioden
- Buchungskategorien
- Buchungskonten

Das bedeutet, wir müssen zumindest aus jeder dieser drei Dateien lesen. Stelle also sicher, dass Du in den FSpezifikationen folgende Dateien hast:

```
FMT F   FDateiname+IPEASFSlän+LSlän+AIE/AEinh.Schlüsselwörter++++++++++++++++++
0001.00 FBUCHDSP    CF   E             WORKSTN
0001.10 F                                      SFILE(SFL1:#SFL1RN)
0001.20 FPERPF      IF   E           K DISK
0001.30 FKATPF      IF   E           K DISK
0001.40 FKONTPF     IF   E           K DISK
0001.50 FBUCHPF     UF A E           K DISK
```

Diese Dateien brauchts

Weiterhin kann es nicht schaden, wenn wir ein temporäres Feld haben, welches nach der Auswertung der F4-Taste vermerkt, welchen Satz der Benutzer ausgewählt hat:

```
FMT D   DName++++++++++ETDsVon++++Bi/L+++IDG.Schlüsselwörter++++++++++++++++++
0002.15 D#SFLWAHL         S             10A   INZ(*blanks)
```

Dies an die D-Zeilen anfügen

Weiterhin benötigen wir ein paar Zugriffsschlüssel, mit denen wir auf die gewünschten Dateien zugreifen. Man muss eigentlich nicht viel dazu sagen, da die Schlüssel recht einfach sind:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0002.35  *  [PERIODEN]
0002.45 C     *LIKE         DEFINE    PERID         kyPERID
0002.55 C     kyPERREC      KLIST
0002.65 C                   KFLD                    kyPERID
0002.75
0002.85  *  [KATEGORIEN]
0002.95 C     *LIKE         DEFINE    KATMATC       kyKATMATC
0003.05 C     kyKATREC      KLIST
0003.15 C                   KFLD                    kyKATMATC
0003.25
0003.35  *  [KONTEN]
0003.45 C     *LIKE         DEFINE    KONTID        kyKONTID
0003.55 C     kyKONTREC     KLIST
0003.65 C                   KFLD                    kyKONTID
0003.75
0003.85  *  [BUCHUNGEN] zur Ermittlung der nächsten Nummer
0003.95 C     *LIKE         DEFINE    BUCHID        kyBUCHID
0004.05 C     kyBUCHREC     KLIST
0004.15 C                   KFLD                    kyBUCHID
```

Unsere Schlüssel zum Erfolg!

Du siehst, mit dem Namensschema kann man sich recht schnell denken, wofür welcher Key und welches Feld ist. Je nach Grösse des Projekts ist es zwar schwierig, ein festes Schema streng durchzuhalten, aber: da muss man eben durch! Jede Abweichung mach das Schema wertlos[I].

Da wir bereits ein funktionierendes Programm mit einem sauberen Grundgerüst haben, müssen wir nur noch erweitern:

Schau in die Routine `CHKF1`. Ein Tipp: Bevor Du nun blätterst wie ein Weltmeister: SEU hilft Dir! Springe mit der Taste F10 in die SEU-Befehlzeile. Tippe nun ein: `chkf1  12` und drücke F16. Der Cursor wird direkt auf den Anfang der Routine `CHKF1` gesetzt. Warum? F16 sagt dem SEU, er soll nach dem suchen, was in der Befehlszeile steht. Allerdings wird als Suchtext nur die Zeichen bis zur ersten Leerstelle akzeptiert. Nach dieser Leerstelle kann man als weiteren Parameter die Spalte angeben, in der gesucht wird. Also sucht SEU nach allen Vorkommnissen von „`CHKF1`“ in Spalte 12. Wenn Du nun genau schaust, ist Spalte 12 die Spalte, in der Faktor1 anfängt. Und wir haben nicht viele Stellen, in denen im Faktor1 der Name unserer Subroutine steht :)

Füge der Routine `CHKF1` folgende Zeilen hinzu:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0023.14  * F4 gedrückt?
0023.24 C                   IF        *in04 = *ON
0023.34 C                   EXSR      PROMPT1
0023.54 C                   ENDIF
```

Aufruf einer weiteren Prüfroutine wenn F4 gedrückt wird

[I]Ausser, die Ausnahme wird konsequent durchgeführt. Ich hatte einen Kunden, der alle Felder mit deutschen Kürzeln benannt hat. Nur die Kundennummer hiess immer CUST ;-)

Wir rufen also eine neue Routine auf, wenn F4 gedrückt wird.

In dieser Routine muss das getan werden, was eben getan werden muss, wenn der Benutzer mit F4 die Eingabemöglichkeiten für ein Feld sehen will.

Beispielhaft für das Feld der Buchungsperioden zeige ich den Weg:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0025.40 C     PROMPT1       BEGSR
0025.50  * Abhängig vom gewünschten Feld...
0025.60 C                   IF        #CSRFLD = 'B1PERID'
0025.70  * PERIODE *
0025.80 C                   EVAL      S1TITEL = 'Auswahl Perioden'
0025.90 C                   EXSR      INITSFL
0026.00 C                   EXSR      FILSFLPER
0027.50 C                   EXFMT     CTLSFL1
0027.60 C                   EXSR      CHKSFL
0027.70 C                   IF        #SFLWAHL <> *blanks
0027.80 C                   EVAL      B1PERID = #SFLWAHL
0027.90 C                   ENDIF
0028.00 C                   ENDIF
```

Behandlung der F4 Taste für das Feld B1PERID

Zunächst prüfen wir die Variable #CSRFLD, unser Displayfile informiert uns, wo F4 gedrückt wurde. Wenn also es sich um das Eingabefeld für die Buchungsperiode handelt, machen wir weiter.
Ich habe die dafür nötigen Schritte bereits in einzelne Routinen ausgelagert, so dass diese später mehrfach benutzt werden können (man ist ja faul).

Nachdem ich die Variable für die Fensterüberschrift belegt habe, rufe ich die Routine INITSFL auf:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0030.90  * INITSFFL - Subfile initialisieren (für alle)
0031.00 C     INITSFL       BEGSR
0031.10 C                   EVAL      *in91 = *ON
0031.20 C                   WRITE     CTLSFL1
0031.30 C                   EVAL      *in91 = *OFF
0031.40 C                   EVAL      *in93 = *ON
0031.50 C                   WRITE     CTLSFL1
0031.60 C                   EVAL      *in93 = *OFF
0031.70 C                   EVAL      #SFLWAHL = *blanks
0031.80 C                   ENDSR
```

INITSFL – Subfile vorbereiten – Für alle Dateien verwendbar

Diese Routine erledigt die üblichen Arbeiten, um das Subfile vorzubereiten. Wie wir oben in unserem Testprogramm schon gesehen haben, muss das Steuersatzformat zunächst mit eingeschalteter *IN91, und dann mit eingeschalteter *IN93 aufgerufen werden. Danach wird unser temporäres Feld #SFLWAHL geleert.

Danach kommt die einzige dateiabhängige Routine der Subfile-Behandlung: Das Subfile ist mit allen möglichen Werten aus der Periodendatei füllen. Auch das ist an sich banal:

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0035.70 C     FILSFLPER     BEGSR
0035.80 C                   EVAL       #SFL1RN = 0
0035.90 C                   EVAL       kyPERID = *LOVAL
0036.00 C     kyPERREC      SETLL      PERREC
0036.10 C                   READ       PERREC                                 99
0036.20 C                   DOW        *IN99 = *OFF
0036.30 C                   EVAL       #SFL1WAHL = *blanks
0036.40 C                   EVAL       #SFL1MATC = PERID
0036.50 C                   EVAL       #SFL1TEXT = PERTXT
0036.60 C                   EVAL       #SFL1RN = #SFL1RN + 1
0036.70 C                   WRITE      SFL1
0036.80 C                   READ       PERREC                                 99
0036.90 C                   ENDDO
0037.00 C                   ENDSR
```

FILSFLPER – Füllen des Subfiles mit allen Perioden-Werten

Hier wird einfach der Suchkey mit dem niedrigstmöglichen Wert belegt, die Datei mit `SETLL` an den Anfang gebracht und dann Satz für Satz gelesen. Die Felder aus der Datei werden in das Subfile geschrieben. Beachte, dass hier `#SFL1RN` jeweils um eins erhöht wird, da wir damit den zu Schreibenden Satz im Subfile anzeigen.

Blicken wir wieder zurück in unsere Routine `PROMPT1`: Nach dem wir das Subfile gefüllt haben, wird mit `EXFMT` das Steuersatzformat ausgegeben, und somit auch der Inhalt des Subfiles. Wenn der Benutzer die Bearbeitung des Subfiles beendet (Datenfreigabe oder erlaubte Funktionstaste), springen wir wieder in eine Routine: `CHKSFL`. Dort wird geprüft, ob der Anwender im Subfile eine Zeile markiert hat, wenn ja, wird deren Matchcodefeld zurückgegeben.[I]

```
 FMT C  CL0N01Faktor1+++++++Opcode&ExtFaktor2+++++++Ergebnis++++++Län++D+HoNiGl
0037.40 C     CHKSFL        BEGSR
0037.50 C                   EVAL       #SFLWAHL = *blanks
0037.80 C                   READC      SFL1                                   99
0037.90 C                   DOW        *IN99 = *OFF AND #SFLWAHL = *blanks
0038.00 C                   IF         #SFL1WAHL <> *blanks
0038.10 C                   EVAL       #SFLWAHL = #SFL1MATC
0038.20 C                   ENDIF
0038.50 C                   READC      SFL1                                   99
0038.60 C                   ENDDO
0038.60 C                   ENDSR
```

Dies ist eine relativ einfache Leseroutine. Zunächst wird unsere temporäre Variable `#SFLWAHL` geleert. Nun lesen wir den ersten *geänderten* Satz im Subfile mit dem Befehl READC (read changed). Der Indikator 99 im Feld „Gl“ wird gesetzt, wenn kein geänderter Satz gefunden wurde.
Dann beginnt unsere Suchschleife. Diese läuft, solange der Indikator 99 zeigt, dass ein Satz gelesen werden konnte, und solange wir unser Zwischenfeld `#SFLWAHL` noch nicht mit einem Matchcode belegt haben. In diesem Falle sind wir noch auf der Suche :-)
Es wird dann geprüft, ob das Feld `#SFL1WAHL` aus dem Subfile eine Auswahl enthält. In diesem Falle wird unser Zwischenfeld `#SFLWAHL` mit dem Matchcode des markierten Satzes aus dem Subfile belegt. Dann beenden wir unsere Schleife mit dem Lesen des nächsten geänderten Satzes.

Diese Routine liest also solange veränderte Sätze aus dem Subfile, bis ein markierter Satz gefunden wurde oder das Ende des Subfiles erreicht wurde. Der Inhalt des Matchcode-Felds aus dem Subfile wird zurückgeliefert. Auch diese Routine können wir für alle drei benötigten Dateien verwenden, da hier nur mit Feldern im Subfile gearbeitet wird.

[I]Markiert der Anwender mehrere Zeilen, wird nur die erste Zeile beachtet.

Nun haben wir also erfolgreich das Feld `#SFLWAHL` gefüllt, oder – wenn der Anwender keine Auswahl gemacht hat – es mit Leerzeichen überschrieben. Dieser Feldinhalt wird nun der Einfachheit halber in das Bildschirm-Eingabefeld übertragen (bei der Auswahl von Periode also in `B1PERID`). Wir können nämlich davon ausgehen, dass der Matchcode existiert. Der Inhalt des Eingabefeldes wird später sowieso noch einmal geprüft und der zu diesem Matchcode zugehörige Text angezeigt.

Da Dir nun garantiert der Kopf schwirrt, folgt nun ein Diagramm des Ablaufs.

Zunächst die Routine `PROMPT1`, die aufgerufen wird, wenn man F4 drückt:

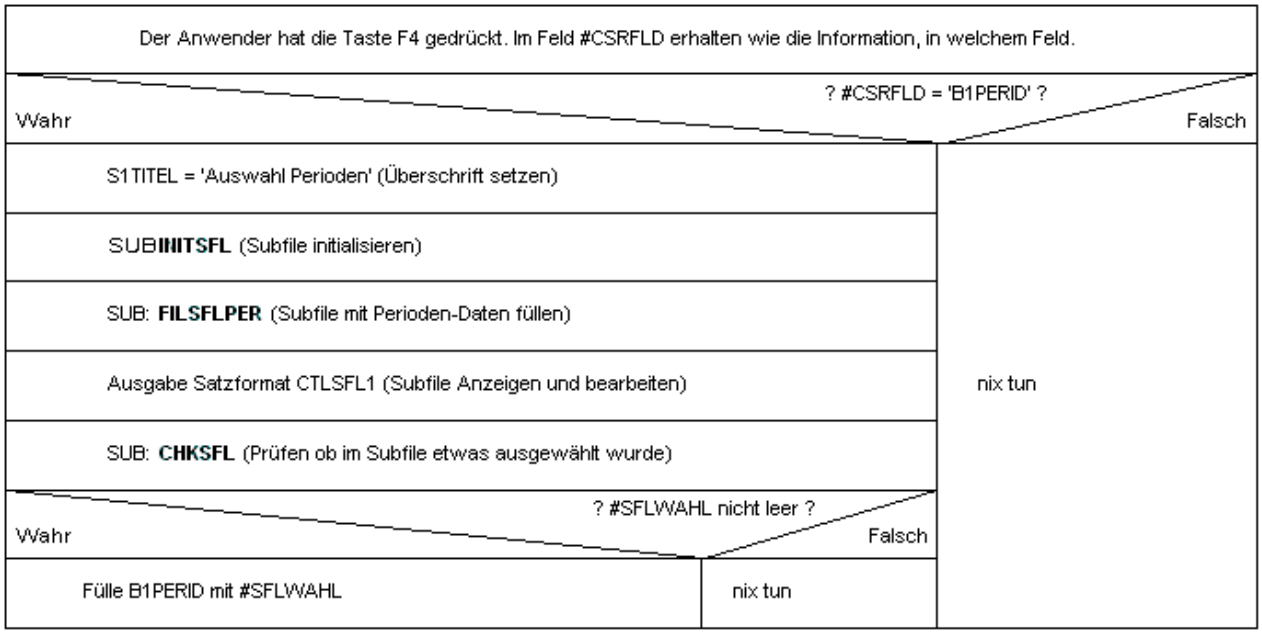


ABB. 1 - Grobübersicht F4

Nun die recht einfache Routine `INITSFL`:

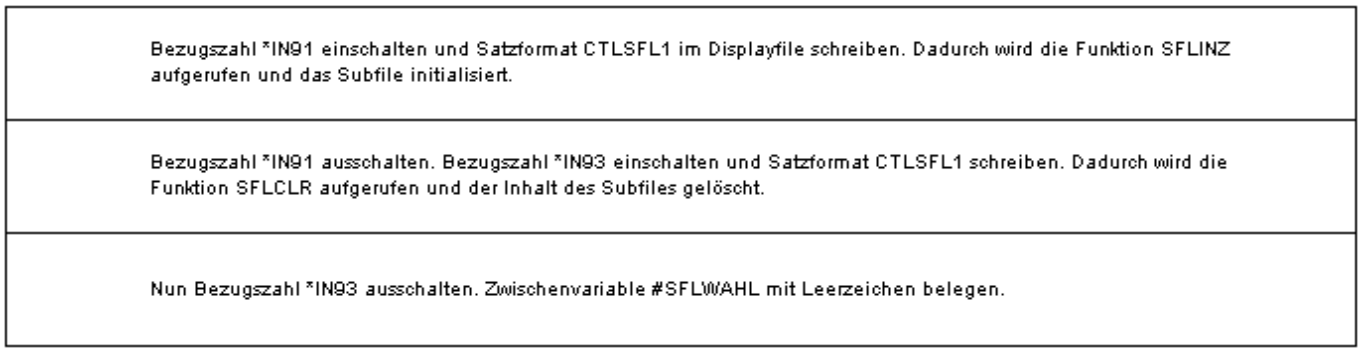


ABB. 2 - Welche Schritte beim Initialisieren des SFL gemacht werden.

Das war der einfache Teil ;-)

Nun folgt die Routine FILSFLPER:

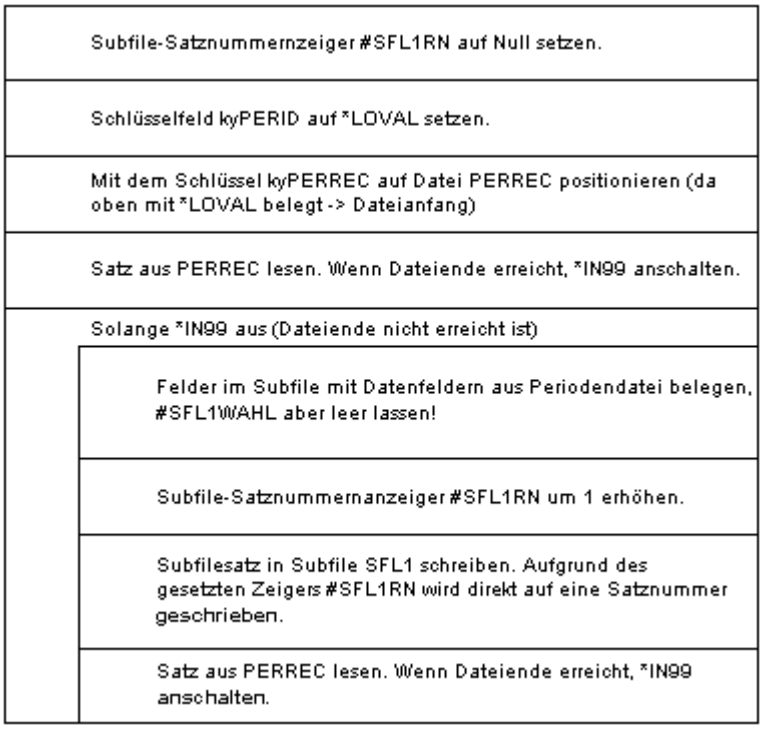


ABB. 3 - Mit FILSFLPER die Periodendaten ins Subfile schreiben

Wie Du siehst, auch nicht unbedingt schwer. Ein paar vorbereitende Befehle und eine kleine Schleife...

Schliesslich noch CHKSFL, das prüft, ob etwas eingegeben wurde:

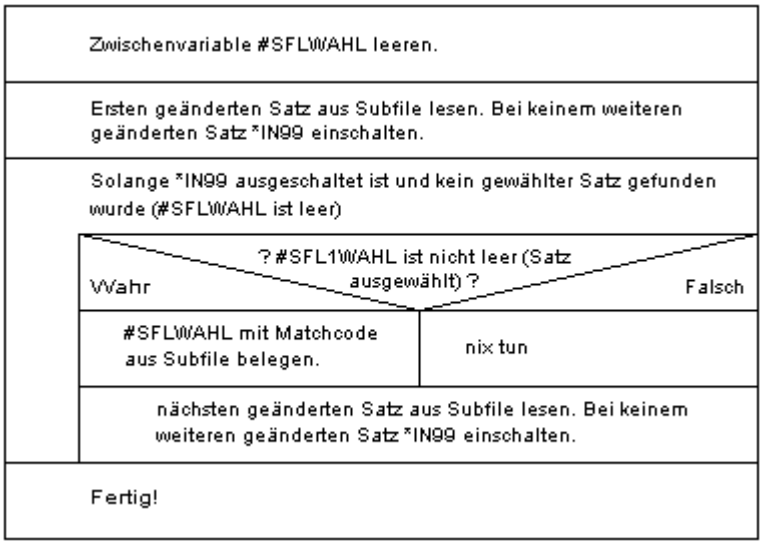


ABB. 4 - Prüfen ob im Subfile was gewählt wurde

Du siehst, das ist alles kein Hexenwerk. Würde man alles in einer Routine unterbringen, könnte man das zum Einen nicht für mehrere Felder verwenden, zum Anderen blickt man nicht durch. Somit ist auch unsere F4-Behandlung fertig. Wir brauchen nur Füllroutinen für die einzelnen Dateien, sowie in PROMPT1 für jedes Feld eine Abfrage. Als Denksportaufgabe kommt nun der komplette Quelltext des Programms, wovon einige Zeilen nicht erklärt sind. Aber mit Hilfe der Promptfunktion wirst Du durchblicken. Um Dir das Abtippen zu ersparen, kannst Du das Programm runterladen (siehe Anhang).

Nun beginnt die schwierige Phase eines Programmierers: **Testen**! Du hast nun ein lauffähiges Programm, das hoffentlich ohne grosse Fehler das Erwartete erledigt. Prüfe dies, in dem Du möglichst viele und unterschiedliche Sätze eingibst. Diese können wir später auch für die Auswertungsroutinen und den Weg dorthin gebrauchen.

Ich überlasse es Deiner Phantasie, wieviele und welche Sätze Du eingibst, es sollten nur so mindestens zwanzig bis 30 sein.

SQL-Auswertungen

Wenn Du ein paar Werte (bitte auch aus mehreren Perioden) eingegeben hast, kannst Du eine erste, grobe Auswertung vornehmen. Starte hierzu das interaktive SQL mit:

```
STRSQL
```

und gib dann folgende Befehlszeile ein:

```
SELECT BUCHPER, BUCHTYP, SUM(BUCHWERT) FROM BUCHPF GROUP BY BUCHPER,
BUCHTYP
```

Du erhälst eine Summierung der Aus- und Eingaben, gruppiert nach der Periode und der Buchungsart (Ausgabe, Einnahme).

Um ein monatliches Ergebnis Deiner Haushaltsführung auf einen Blick zu sehen, reicht folgender Befehl:

```
SELECT BUCHPER, SUM(BUCHWERT) FROM BUCHPF GROUP BY BUCHPER
```

Hier werden pro Periode einfach alle Buchwerte addiert. Da wir Ausgaben negativ ablegen, werden diese abgezogen und Du siehst auf einen Blick, ob am Ende des Geldes noch Monat übrig ist oder umgekehrt :)

Hier eine Gesamtübersicht, welche Kategorien welche Ausgaben oder Einnahmen hatten:

```
SELECT BUCHKAT, SUM(BUCHWERT) FROM BUCHPF GROUP BY BUCHKAT
```

Gegeben bei einem Anfangsstand von Null, siehst Du hiermit den aktuellen Kontostand:

```
SELECT SUM(BUCHWERT), BUCHKTO FROM BUCHPF GROUP BY BUCHKTO
```

Alle Befehle können natürlich auch in Kleinbuchstaben eingegeben werden. Aus diesen Abfragebeispielen werden wir im nächsten Update bei Bedarf und Anfrage ein paar kleine Auswertungsprogramme und Druckroutinen schreiben.

Aufgabenstellung bis zum nächsten Update:

Wenn Dein Programm funktioniert, wird unsere kleine Buchhaltung erweitert. Baue in alle nötigen Tabellen und Dateien ein weiteres Feld „`Empfänger`“ ein. Dieser Empfänger ist bei Ausgaben der Zahlungsempfänger. Diese Information wird bei jeder Buchung gespeichert. Lege hierfür eine Stammdatei wie für die Konten oder die Kategorien an, und ermögliche eine F4-Auswahl auf dem Eingabebildschirm.

Bei Fragen schreibe mir einfach eine Mail, die Auflösung kommt in der nächsten Ausgabe :)

Weiterhin mach Dir schon ein mal Gedanken über Druckprogramme und Auswertungen. Wenn Du bestimmte Auswertungen in einem Programm haben willst, aber Du Probleme beim Programmieren hast, maile mir Deine Ideen und ich versuche, diese in die nächste Ausgabe mit aufzunehmen.

Anhang 1: Alt aber gut

Schon öfter habe ich Anfragen von Lesern bekommen, wie man denn am günstigsten an eine eigene AS/400 kommt. Der Gebrauchtmarkt ist in Deutschland leider nicht ganz so florierend wie in Nordamerika, aber man findet doch gelegentlich ein Schnäppchen. Da der potentielle Gebrauchtkäufer aber meist nicht einmal weiss, was er da kauft (er will sich mit der Thematik ja erst beschäftigen), möchte ich hier einige kleine Hinweise zum Suchen und Kaufen geben. Potentielle Verkäufer bitte nicht ärgern, aber die Wahrheit ist manchmal recht böse :-)

Was kaufen?

Technologiebereiche

Zunächst einmal der erste warnende Finger. Es gibt grob gesagt zwei Gruppen von AS/400. Die älteren Systeme mit CISC-Prozessor(en), sowie die neuen RISC-Systeme. Die älteren sind meist im beigen Gehäuse, die RISC-Systeme sind immer schwarz. Es gibt aber auch einige schwarze Geräte mit CISC-Prozessor (400, 500er Modelle)
Der große Unterschied hier ist, dass bei den CISC-Modellen bei Betriebssystem V3R2M0 Schluss ist. Das bedeutet: Support von IBM ist hierfür seit Jahren eingestellt, es fehlen viele moderne Funktionen zur Vernetzung und neueste Internet-Protokolle. Aber mit diesen Systemen kann man trotzdem ILE RPG Programmieren, sich mit dem System vertraut machen und auch heute noch komplexe Anwendungen laufen lassen. Selbstverständlich ist auch eine ältere AS/400 Netzwerk- und TCP/IP-fähig, wenn eine Netzwerkkarte eingebaut ist. Die neuen RISC-Modelle sind aber meist wesentlich schneller, und die neuen Betriebssystemversionen unterstützen natürlich einige tolle neue Sachen. Viele der Beispiele in diesem Buch kann man auch mit den alten CISC-Modellen nutzen.

Betriebssysteme

Nein! Keine Chance! *Vergesst* es! Auf einer CISC-Maschine kann man *kein* Linux laufen lassen, bisher hat es keiner geschafft, und meines Erachtens wird es auch nie klappen, denn: Den CISC-Prozessoren fehlt eine kleine Eigenschaft, die Linux unbedingt benötigt: den Speicherschutz zum hardwaremässigen Abschotten der einzelnen Tasks untereinander. Wirklich, sowas haben diese CPUs nicht, da das Betriebssystem dies selbst regelt. Also bitte legt diese Idee beiseite.

Die CISC-Maschinen laufen nur unter OS/400 bis max. Version V3R2M0. Die IBM-Lizenzbedingungen sehen vor, dass diese OS-Versionen unabhängig von der Hardware lizensiert werden. Daher erwirbst Du mit einer alten CISC-Maschine auch keine OS/400 Lizenz. Nach meinen Informationen sieht IBM dies aber recht locker, sofern die Maschine nicht für professionellen Geschäftsbetrieb verwendet wird, sondern zum privaten Lernen. Auf jeden Fall solltest Du darauf achten, dass der Verkäufer Dir Bänder mit dem Betriebssystem zum installieren gibt, und evtl. aktuelle Patches. Wenn man kein Betriebssystem hat, und auf der Maschine keins installiert ist oder Du das Kennwort nicht kennst, taugt das System höchstens als Blumenvasenständer oder Hocker oder Wohnzimmerzierde, je nach Größe. Solltest Du irgendwo ein Angebot für ein System ohne Betriebssystem sehen und keinen kennen, der Dir das System netterweise kopiert (pssst!), lass die Finger davon. Ebenso sollte man den Verkäufer nach dem MULIC, bzw. SLIC-Band fragen (je nach Maschinentyp auch nötig). Bei vielen Modellen enthält dieses Band grundlegende Routinen, die nach totalem Plattencrash installiert werden müssen und für jedes Gerät erstellt wurden (da auch die Seriennummer darauf ist). Also auch hier nachfragen.

Bei den RISC-Maschinen sieht es schon besser aus. Die erste OS-Version war V3R6M0, das Pendant zu V3R1M0 auf den CISC-Maschinen. Allerdings nicht wirklich toll, auch V3R6M0 ist nur eine Vorstufe. Man sollte darauf achten, mindestens V4R1M0, besser V4R2M0 zu bekommen. Entgegen einiger Aussagen hat mir erst neulich ein IBMer ausdrücklich erklärt, dass bei V4 das OS immer zur Hardware gehört, also bei Verkauf auch mitgeliefert werden sollte (Ausnahme: Kunde hat auf V5 aktualisiert). Dieses System wird immer auf CD-ROM ausgeliefert, das macht die Installation etwas angenehmer. Wer also eine RISC-Maschine ohne Betriebssystem angeboten bekommt, sollte ganz die Finger davon lassen oder wissen, was er tut. Denn in dem Falle wurde die Lizenz auf ein grösseres System mit übernommen, und die angebotene ältere Hardware hat keine Softwarelizenz. Das kann teuer werden, wenn Du eine Lizenz (sofern noch erhältlich) nachkaufen musst. Nach neuesten Informationen wird V4 nicht mehr verkauft, also muss man sich auch hier andersweitig behelfen...

Systeme

Was also kaufen wir nun? Der Bereich AS/400-Hardware ist so komplex, dass es von IBM einen Stapel Bücher, Konfiguratorsoftware und viele elektronische Publikationen gibt, die will ich an dieser Stelle nicht ersetzen. Welche Steckkarte in welches Gehäuse passt, sprengt hier den Rahmen, einige notwendige Sachen erwähne ich aber.
Nehmen wir einfach mal ein paar fiktive (bzw. von realen Angeboten abgeleitete) Maschinen an:

- *Der einfachste Fall*: Dir bietet jemand „eine AS/4002 an. Mehr Informationen gibt's nicht. Das ist so originell wie „Ich verkaufe ein Auto“. Was, welcher Zustand, wie etc. sind nicht angegeben. Hier hilft entweder entschiedenes Nachfragen, oder, wenn der Verkäufer angibt, sich damit nicht auszukennen, bitte um alle Zahlen, die vorne oder hinten auf dem System oder in eventuellen Dokumentationen stehen. Ansonsten: kauf nicht, denn Du wirst im Zweifelsfall nur Blech kaufen.

- *Überteuerte Maschinen*: Vorweg: Fast alle CISC-Modelle in beigem Gehäuse sollte man nur zum Kilopreis von höchstens einem Euro pro Kilogramm kaufen. Sorry, liebe Verkäufer, aber mehr sind die Kisten nicht wert. Da sie aber oft schwer sind, kriegt man immerhin das Geld für ein Abendessen raus. Ein Beispiel: AS400 Typ 9404 (mit Kennzeichnung Modell meist B10, D10,E02 etc.) Das ist ein kleines bis mittleres Maschinchen mit relativ geringer Leistung. Reicht für 2-3 Leute zum Programmieren und experimentieren, oder für kleinere Softwarepakete. Die Modelle B10/D10 sind ungefähr so groß wie ein Schreibtisch-Unterschrank (ca. 45cm breit, 80cm hoch und 90cm tief) und wiegen 80-100 Kilogramm (!). Also schon ein Transportproblem. Darin sind meist 8 oder mehr Megabyte RAM. Mehr ist gut, unter 16MB machts keinen Spass. Üblicherweise wurden dort 320MB oder 640MB Festplatten eingebaut, meistens gehen nur 4 davon in die Kiste rein. Also bis 1,2 oder 2,4 GB, das ist aber nicht wirklich viel. Wird Dir ein Modell D10 mit >3 GB angeboten, dann Vorsicht: Es ist wohl eine Maschine mit Erweiterungsbox, also doppelt so breit und doppelt so schwer! Neben den Bändern für das Betriebssystem sollte ein Bandlaufwerk dabei sein (manche Kisten haben keins, weil ausgebaut für andere Zwecke, dann Finger weg). Beispielhafter Wert für eine 9404 D10 mit 16MB RAM, viermal 320MB Festplatte, und einer Netzwerkkarte: max. 100 Euro (zuzüglich Transport). Man kann etwas mehr ausgeben, wenn ein Twinax-Bildschirm oder 32MB RAM drin sind, aber viel höher sollte man nicht gehen. Bei eBay habe ich solche Kisten schon für >1000€ gesehen, das ist nicht nur Wucher, sondern Phantasie. Es gibt auch noch Systeme vom Typ 9402, die sind Leistungsmäßig knapp drunter, dementsprechend gilt der Preis. Modelle vom Typ Fxx (nicht F04) können sehr gross (Kühlschrank) sein und sehr schwer (>200kg) werden, und lohnen daher kaum.

- *Seltene und kleine Maschinchen*: Hierzu zählen einige Modelle, die platzsparend, da klein sind. Natürlich von Privatanwendern sehr beliebt. Beispiele: Modell 150, 170, 250, 270, 200, 400, 236, 436, 600 und P03. Die P03 ist die kleinste AS/400 auf dem Markt und wurde gerne für Messevorführungen eingesetzt. Sie ist so groß wie ein Aktenordner für DIN A4-Papier, nur etwas tiefer (ca. 8cm breit, 40cm hoch, 55cm tief, 6kg Gewicht). Drin ist ein gemütlicher CISC-Prozessor, 8-56MB RAM und 1-4GB Festplatte. Ein wirklich schnuckeliges Teil auch für unterwegs, und reicht zum Programmieranfang.
  Die anderen genannten Modelle sind so groß wie ein PC-Tower (mit leichten Abweichungen). Die 2xx ist ein CISC-Modell mit geringerer Leistung (bis 100€ würde ich zahlen), alle anderen sind RISC-Modelle und daher teurer. Für Einsteiger tut es eine gebrauchte 400 auf jeden Fall (je nach Ausstattung schon diverse bis viele Hundert Euro wert, aber auch so gross wie ein Schreibtisch-Rollschrank), oder die etwas teurere aber leistungsmäßig nicht flottere 150, wie sie in meinem Büro steht und per Standleitung erreichbar ist. Für diese Maschine kann man je nach Ausstattung aber noch 1000-3000€ hinblättern. Besonderheit bei der 150: Bis System V4R5M0 benötigt man keine Lizenzschlüssel für die Software. Wer also ein Betriebssystem hat, kann es (und zwar nur auf der 150) mit allen Optionen installieren.
  Die 170 ist je nach Ausstattung schon eine recht flotte Kiste, die viele große CISC-Kisten blass aussehen läßt. Bis zu 170GB Festplatte und >1GB RAM gehen da rein, die Preislage geht, je nach Glück, von ca. 1500€ bis in den fünfstelligen Euro-Bereich. Auf solchen Kisten, die wie ein schwarz angemalter PC aussehen, können auch schon mal über 200 Anwender Buchhaltung und Warenwirtschaft benutzen, Leistung gibt's genug. Von den Modellen 250 und 270 reden wir lieber nicht. Diese sind auf dem Gebrauchtmarkt selten zu kriegen und wenn, dann für den Privatanwender, der sich damit beschäftigen will, zu teuer. Die 270 ist ein Wolf im Schafspelz, so groß wie ein PC-Tower, etwas schwerer, aber ich habe einen Kunden, der auf einer solchen Maschine knapp 1000 Anwender versorgt, inklusive eMail, Warenwirtschaft, Statistiken etc, und die Maschine versteckt sich unter dem Schreibtisch des PC-Administrators.

- *Unrealistisch*: Die Modelle 270, 800, 810, 820, 825, 840, 890 sind für den Privatanwender preislich nicht zu empfehlen, außer, du hast am Wochenende sowieso gerade keine Lust, wieder mit Deinem Privatjet nach Amerika zu jetten ;-) und weisst sonst nicht, wohin mit den 20.000€ bis diversen Millionen.

- *Räumlich kritisch*: Die „big irons“ des Marktes. Das sind die Modelle 320, die 500er, 620er und 700er Serie. Das sind Maschinen, auf denen der Mittelstand seine Anwendungen laufen läßt. Leistungsmäßig von mittel bis „wow“, kriegt man manchmal schon eine 320er hinterhergeworfen. Der Grund: So groß wie ein Kühlschrank (oder auch mehr), das Gewicht wird auch Schwarzenegger vom Transport abhalten (250kg sind üblich), daher will die keiner haben, viele landen auf dem Schrott, auch wenn Einzelteile oft verwertbar sind. Bedenke, dass in vielen Häusern der Boden nur bis 150kg oder 200kg je Quadratmeter ausgelegt sind, das können die Dinger locker überbieten.

- *Sinnlos*: Kauf keine Einzelteile, wenn Du nicht genau weisst, was Du davon gebrauchen kannst und ob es in Deine vorhandene Maschine passt. (wollte ich nur mal am Rande erwähnen)

- *Unbrauchbar*: Kaufe keine Modelle 236, oder 436 mit SSP-System, bei denen kein OS/400 dabei ist. Das sind Systeme, die für den Betrieb alter S/36-Software gebaut wurden. Meist ist gerade bei diesen Angeboten kein OS dabei. Wenn explizit von OS/400 gesprochen wird, kann man für einen moderaten Preis darüber reden. Die 236er kann sowieso nur das alte SSP (System/36). Auf die 436er kann man OS/400 installieren, wenn mindestens 64MB RAM eingebaut sind.

Zubehör

So eine nackte AS/400 ist recht langweilig. Man hat ja auch kein Motorrad ohne Sitz, ohne Tank und ohne Lenker, oder? Zu einer AS/400 gehört mindestens eine Konsole. Das ist der Bildschirm, der für den normalen Betrieb nicht benötigt wird, aber für Wartungsarbeiten im so genannten „manuellen Modus“. Ohne Konsole kannst Du Deine neue Maschine zwar hochfahren, aber evtl. ist sie nicht für Netzwerkbetrieb konfiguriert (sofern sie überhaupt eine Netzwerkkarte hat), und dann nützt Dir das System nur als Stromverbraucher.
Bei fast allen Systemen ist eine Twinaxkarte für die Konsole drin, meines Wissens haben nur wenige Modelle 150 und 170 sowie gelegentlich alte CISC-Modelle eine serielle Schnittstelle für die Konsole. Dafür benötigt man dann aber auch besondere Software, die sollte auf jeden Fall dabei sein (nennt sich Client Access). Ansonsten, im Twinax-Falle, ist es recht einfach. Wenn Du nur über Twinax (weil kein Netzwerk) arbeiten willst, solltest Du einen Twinaxbildschirm mit Farbdarstellung nehmen, ansonsten tuts auch ein Schwarzweiss-Bildschirm. Diese Dinger findet man oft, natürlich von IBM, aber auch von Memorex o.ä. Bei IBM heissen die Geräte 3196 (s/w), 3197 (Farbe), 3487, 3488, 3489 etc. Die Geräte unterscheiden sich neben dem Bildschirm auch in der Anzahl von gleichzeitigen Sitzungen, und ob man daran direkt einen Drucker anschließen kann. Als Konsole und Arbeitsbildschirm funktionieren alle, im Zweifel nachfragen. Bei Memorex heissen die Geräte 1477, 1478, 1489 etc. Interessant sind Modelle (IBM 3488, Memorex 1489 etc), die nur aus der Steuerbox (auch Pizzabox genannt) und der Tastatur bestehen. Daran kann man fast immer einen normalen VGA-Monitor anschließen und spart Platz. Eine Tastatur sollte aber bei einem Terminal immer dabei sein, da diese meist einen etwas eigenen Anschluß haben und 24 Funktionstasten sowie einige Sondertasten haben, und eine PC-Tastatur meist nicht daran funktioniert.
Für ein S/W-Terminal sollte es ein Zehner (Euro) tun, gute Farbterminals oder Steuerboxen sollten höchstens 50 bis 100€ (mit Bildschirm) kosten.
Mehr Spaß hat man mit einer Twinax-Karte für den PC. Aber Achtung, am besten eine IBM-Karte auftreiben, dann klappts oft auch mit ClientAccess unter Windows (die Software sollte auch dabei sein). Ich habe hier eine IDEA-Twinaxkarte, die nur unter DOS-Läuft, aber zum Glück im DOS-Fenster von Win98. Damit spart man sich auf jeden Fall den Platz für die eigene Konsole. Immerhin ist die Karte Baujahr 1990 und läuft noch :-)

Natürlich darf auch Kabel nicht fehlen. Neben einem Stromkabel (jaja...) benötigt man für seine Twinax-Konsole auch einen entsprechenden Controller. Dies ist eine Box (bei kleinen Kisten auch eine Art Y-Kabel), die direkt an das System angeschlossen wird und 2 bis 8 Twinax-Anschlüsse hat (diese runden Dinger mit 2cm Durchmesser und zwei Pins). Ein passendes Kabel benötigt man für die Konsole. Wer Netzwerken will, benötigt ebenfalls Kabel. Ethernet ist einfach, an der AS/400 sollte eine entsprechende RJ45- oder BNC-Buchse sein, die man entsprechend mit dem PC verbindet. Wer TokenRing in seiner AS/400 hat (oft erkennbar am grünen Punkt auf der Karte oder dem 9poligen Anschluß mit Zusatzstecker, oder gar an diesen alten, klobigen Typ1-Steckern, muß aufpassen. Für TokenRing benötigt es einen Ring-Leitungsverteiler, zwei Geräte (AS/400 und PC) kann man nicht direkt zusammenstöpseln. So ein Ringleitungsverteiler, oft auch Hub genannt, bekommt man gelegentlich für ein paar Euros.

Bänder! Ja, ich weiß, Backup ist was für Feiglinge. Wer ein Bandlaufwerk in seiner AS/400 hat (wer hat das nicht?), sollte sich auch um ein paar leere Bänder kümmern, um sein System mal sichern zu können. QIC-Bänder von 150-2500MB kriegt man relativ günstig.

Das wärs erst mal, damit kann man dann schon ganz gut arbeiten.

Bezugsquellen

Wie gesagt, der Markt in Deutschland ist leider nicht so groß wie in Amerika. Vor allem sollte man nicht bei einem so genannten Broker kaufen. Nicht, dass ich diese Händler schlecht machen will, aber diese

handeln fast nur mit größeren Gebrauchtsystemen, die für den geschäftlichen Betrieb geeignet und daher teuer sind. Ausserdem wollen die Firmen auch von etwas Leben. Der Privatmann, der sich nur etwas mit der Materie beschäftigen will, ist bei Online-Auktionatoren wie eBay (http://www.ebay.de) besser dran. Hier (http://www.as400forum.de) gibt es auch einen kleinen Gebrauchtmarkt, aber die Maschinen sind auch etwas größer. Fragen schadet nicht! Wer noch günstige Bezugsquellen gefunden hat, möge sie mir mailen. Die Adressen von kommerziellen Anbietern möchte ich an dieser Stelle nicht veröffentlichen, bevor es Ärger gibt. Für Fragen bin ich aber zu haben.

Preisspiegel

Folgende Preise für Hardware sind ungefähr realistisch (Stand: Mai 2004)

- AS/400 Modell 150 (RISC), CPU 2269 oder 2270, 20/35CPW, 192MB RAM, 4*4GB Festplatte, Ethernet-Adapter, Lizenz V4Rx
  ==> ca. 800-1000 Euro
- AS/400 Modell 170 (RISC), CPU 2289 oder 2290, 50-70CPW, 128MB RAM, 2*4GB Festplatte, Ethernet-Adapter, Lizenz V4Rx
  ==> ca. 1200 Euro
- AS/400 Modell 400 (RISC) CPU 223x, 26CPW, 224MB RAM, 4*4GB Festplatte, Ethernet-Adapter, ohne Lizenz
  ==> ca. 700 Euro
- Ethernetkarte 10MBit für AS/400 Modell 150
  ==> ca. 150 Euro (werden momentan hoch gehandelt da selten)
- Festplatte 4GB 6607 für AS/400 (evtl. mit Rahmen)
  ==> ca. 30-50 Euro
- Festplatte 8GB 6713 für AS/400 (evtl. mit Rahmen)
  ==> ca. 9 Euro (unbekannte Quelle)
  ==> ca. 150 Euro (bekannte Quelle)
  ==> bis zu 300 Euro (von einem Broker aus Wartungsvertrag)

Gerade bei Festplatten ist der Preis seeehr (!) schwankend, man muss aber ein wenig auf die Quelle achten. Mir sind schon Festplatten angeboten worden, die durch Tricks von einer PC-Platte auf eine AS/400-Platte umgewandelt wurden. Für sowas gibt's von IBM keinerlei Garantie.

Wahrscheinlich sind die Preise morgen sowieso wieder anders, daher diese Angaben nur als Richtwerte.

Schau mal rein!

Wenn Du als potentieller Käufer einer AS/400 vor derselben stehst und Dir nicht sicher bist, was genau da drin ist (viele Verkäufer wissen das auch nicht), hier ein paar Tips:

Melde Dich als QSECOFR an und probiere folgende Befehle durch[1]:

`WRKDSKSTS` – Mit Plattenstatus arbeiten

```
                          Mit Plattenstatus arbeiten                BLACKBOX
                                                            20.10.03  17:20:59
Abgelaufene Zeit:   00:00:00

Ein-          Größe  %be-    E/A   Anford.  Lese   Schrb.  Lesen  Schrb  %ver-
heit  Typ      (M)   legt   Anf.  Größe(K)  Anf.   Anf.    (K)    (K)   wend.
   1  6713    8589  48,6    0,0       0,0    0,0    0,0     0,0    0,0      0
   1  6713    8589  48,6    0,0       0,0    0,0    0,0     0,0    0,0      0
   3  6713    8589  48,5    0,0       0,0    0,0    0,0     0,0    0,0      0
   3  6713    8589  48,5    0,0       0,0    0,0    0,0     0,0    0,0      0
```

Ansicht des Plattenstatus

Hier siehst Du ein System mit vier Festplatten. Für jede Festplatte wird eine Zeile angezeigt. Schau in der Spalte „Einheit“, dort findest Du jeweils für zwei Platten die gleiche Nummer. Dies bedeutet, dass auf dem System die Festplattenspiegelung eingeschaltet ist. OS/400 sieht eine Einheit Nummer 1, im System sind dafür zwei Festplatten eingebaut.

Beachte die Spalte „Größe(M)“. Hier wird die netto für das System verwendbare Grösse angezeigt. Erwartest Du zum Beispiel ein System mit 4*8GB Festplatten (wie oben angezeigt), siehst aber nur 4 Festplatten mit der Grösse von ca. 6441 MB, ist das kein Betrug, sondern ein eingebauter RAID-Controller. Dieser verwendet von den vier physikalisch vorhandenen Festplatten eine für die Abspeicherung redundanter Informationen, so dass eine beliebige der vier Platten ausfallen kann. Dadurch reduziert sich die nutzbare Kapazität. Rechne selbst: drei nutzbare Festplatten a 8589MB = 25767MB. Da dem OS/400 aber immer die Anzahl physikalisch vorhandener Festplatten angegeben wird, teilst Du diese 25767MB durch vier und erhälst 6441MB, die pro Platte netto verfügbar sind.
Also keine Angst! Wichtig ist der Plattentyp, hier eine Auswahl:

Typ	*Kapazität*
6602	1GB
6604	2GB
6607	4GB
6613	8.5GB (7.200 Umdrehungen/Minute)
6617	8.5GB (10.000 Umdrehungen/Minute)
6618	17GB (10.000 Umdrehungen/Minute)
67xx	Eine Platte im Einschubrahmen für PCI-Systeme (170, 600 etc)
68xx	Eine Platte im Einschubrahmen für SPD-Systeme (500, 700 etc)

[1]Wenn es keine Möglichkeit gibt, sich als QSECOFR am Verkaufsobjekt anzumelden, frage nach einem Hardware-Ausdruck, den vielleicht jemand erstellt hat, oder lass die Maschine aufschrauben. Man kauft ja keine Katze im Sack!

Wieviel Hauptspeicher hat die Kiste? Ganz einfach zu ermitteln:

```
WRKHDWRSC *PRC
```

Zeigt alle Prozessor-relevanten Daten an:

```
                          Mit Prozessorressourcen arbeiten
                                                          System:   BLACKBOX
Auswahl eingeben und Eingabetaste drücken.
  7=Ressourcendetails anzeigen

Opt  Ressource       Typ-Modell  Status                Text
     CEC01            9401-150   Betriebsbereit        Hauptkartengehäuse
     PN01             2468-001   Betriebsbereit        Systemsteuerkonsole
     MP02             2270-000   Betriebsbereit        Systemprozessorkarte
     SP01             6756-002   Betriebsbereit        Serviceprozessorkarte
     BCC01                       Betriebsbereit        Buserweiterung
       MS01                      Betriebsbereit        64MB Hauptspeicherkarte
       MS02                      Betriebsbereit        64MB Hauptspeicherkarte
       MS03                      Betriebsbereit        64MB Hauptspeicherkarte
```

Ausgabe der Prozessorinformationen

Du siehst hier eine AS/400 Typ 9401 Modell 150 (siehe unter Ressource CEC01, Hauptkartengehäuse). In diesem System ist unter anderem als Mpxx einen Systemprozessor Typ 2270 eingebaut. Weiterhin siehst Du drei Einträge Msxx (Main Storage) mit jeweils 64MB. Also hat das System 192MB Hauptspeicher.

Den Eintrag PN01 ignorieren wir einfach, der ist für den Steuerprozessor (vorne am Gehäuse, wo das Display 01 BN zeigen sollte). Da kann man nicht viel manipulieren.

Wichtig ist auf jeden Fall, den Eintrag für die Systemprozessorkarte (oder bei grösseren Systemen mehrere) zu beachten. Denn daraus erkennst Du, wie schnell das System ist, ob es CISC (alt) oder RISC (neu) ist, und ob Du mit dem System Spass haben wirst. Daher folgt auf Wunsch einiger Leser und eBay-Beobachter ab der nächsten Seite eine lange Liste mit den vielen CPU-Typen, die IBM unter anderem eingebaut hat (und Angaben zur verfügbaren Leistung)

Beachte auch die Spalte „Leistung CPW“. Die CPW[1]-Angabe hier ist ein Messinstrument, um AS/400 Systeme (grob) vergleichen zu können. CPW1.0 ist die Leistung der ersten Modelle gewesen. Weiterhin kann der angegebene CPW-Wert nur erreicht werden, wenn das System möglichst viel RAM und möglichst viel Festplatten hat. Ein System mit beispielsweise 15CPW-Kapazität, die die CPU liefern könnte, aber nur 64MB Hauptspeicher und einer 4GB-Festplatte wird schweinelangsam sein, so im Bereich von 3-4 CPW.

Weiterhin unterscheiden sich die CPW-Leistungen nach der eingesetzten Betriebssystemversion. Beim Wechsel der RISC-Modelle von V3R6 bis V5R2 optimiert IBM immer einiges in den Systemroutinen. Die hier angegebenen Werte gelten bei RISC-Systemen immer für V4R5 aufwärts, bei CISC-Systemen für V3R2.

Ist in der Spalte „Leistung“ in Klammern ein Wert „xxx int.“ angegeben, bedeutet dies, dass das System eine Batch- (oder Client/Server-) Leistung hat, die dem ersten Wert enspricht. Für interaktive Programme (Green Screen) steht die Leistung in Klammern zur Verfügung.
Bei den neuen Modellen ab 250 gibt es Kombinationsmöglichkeiten „Gesamtleistung / Interaktivleistung“. Ich habe dies entsprechend notiert und in der letzten Spalte die Prozessorgruppe Pxx angegeben. Je höher die Prozessorgruppe, umso teurer die Software!

[1]CPW = Commercial Processing Workload, ein IBM-Vergleichswert für OS/400-Systeme.

CPU-Typ	Leistung CPW	Typ	in welchem AS/400 Modell z.B.?	Release / Pxx
2030	7.3	CISC	200	bis V3R2
2031	11.6	CISC	200	bis V3R2
2032	16.8	CISC	200	bis V3R2
2040	11.6	CISC	300	bis V3R2
2041	16.8	CISC	300	bis V3R2
2042	21.1	CISC	300	bis V3R2
2043	33.8	CISC	310	bis V3R2
2044	56.5	CISC	310 (zwei CPUs)	bis V3R2
2050	67.5	CISC	320	bis V3R2
2051	120.3	CISC	320 (zwei CPUs)	bis V3R2
2052	177.4	CISC	320 (vier CPUs)	bis V3R2
2102	16.3	RISC	436	bis V4R5
2104	20.6	RISC	436	bis V4R5
2106	27.4	RISC	436	bis V4R5
2130	13.8	RISC	400	bis V4R5
2131	20.6	RISC	400	bis V4R5
2132	27.0	RISC	400	bis V4R5
2133	35.0	RISC	400	bis V4R5
2140	21.4	RISC	500	bis V5R2
2141	30.7	RISC	500	bis V5R2
2142	43.9	RISC	500	bis V5R2
2143	81.6	RISC	510	bis V5R2
2144	111.5	RISC	510	bis V5R2
2150	148.0	RISC	530	bis V5R2
2151	188.2	RISC	530	bis V5R2
2152	319.0	RISC	530 (zwei CPUs)	bis V5R2
2153	598.0	RISC	530 (vier CPUs)	bis V5R2
2162	650.0	RISC	530 (vier CPUs)	bis V5R2
2269	20.2 – 27.0	RISC	150 Typ #0181, #0183, #0191, #0193	bis V5R2
2270	20.2 – 35.0	RISC	150 Typ #0182, #0184, #0192, #0194	bis V5R2
2129	22.7	RISC	600	bis V5R2
2134	32.5	RISC	600	bis V5R2
2135	45.4	RISC	600	bis V5R2
2136	73.1	RISC	600	bis V5R2
2175	50.0	RISC	620	bis V5R2
2179	85.6	RISC	620	bis V5R2
2180	113.8	RISC	620	bis V5R2

CPU-Typ	Leistung CPW	Typ	in welchem AS/400 Modell z.B.?	Release / Pxx
2181	210.0	RISC	620	bis V5R2
2182	464.3	RISC	620 (zwei CPUs)	bis V5R2
2237	319.0	RISC	640	bis V5R2
2238	583.3	RISC	640 (zwei CPUs)	bis V5R2
2239	998.6	RISC	640 (vier CPUs)	bis V5R2
2240	1794.0	RISC	650 (acht CPUs)	bis V5R2
2243	2340.0	RISC	650 (zwölf CPUs)	bis V5R2
2188	3660.0	RISC	650 (acht CPUs)	bis V5R2
2189	4550.0	RISC	650 (zwölf CPUs)	bis V5R2
2159	75.0 (16.0 int)	RISC	170	V4R3 - V5R3
2160	114.0 (23.0 int)	RISC	170	V4R3 - V5R3
2164	210.0 (29.0 int)	RISC	170	V4R3 - V5R3
2176	319.0 (39.0 int)	RISC	170	V4R3 - V5R3
2183	319.0 (65.0 int)	RISC	170	V4R3 - V5R3
2289	50.0 (15.0 int)	RISC	170	V4R3 - V5R3
2290	73.0 (20.0 int)	RISC	170	V4R3 - V5R3
2291	115.0 (25.0 int)	RISC	170	V4R3 - V5R3
2292	220.0 (30.0 int)	RISC	170	V4R3 - V5R3
2385	460.0 (50.0 int)	RISC	170	V4R3 - V5R3
2386	460.0 (70.0 int)	RISC	170	V4R3 - V5R3
2388	1090.0 (70.0 int)	RISC	170 (zwei CPUs)	V4R3 - V5R3
2407	30.0 (10.0 int)	RISC	170 (Dedicated Server for Domino)[I]	V4R4 bis...
2408	60.0 (15.0 int)	RISC	170 (Dedicated Server for Domino)	V4R4 bis...
2409	120.0 (20.0 int)	RISC	170 (Dedicated Server for Domino)	V4R4 bis...
2295	50.0 (15.0 int)	RISC	250	P05
2296	75.0 (20.0 int)	RISC	250	P05
2248	150.0	RISC	270 (Int: FC#1517 = 25.0)	P05
2250	370.0	RISC	270 (Int: FC#1518 = 30.0)	P10
2252	950.0	RISC	270 (Int: FC#1519 = 50.0)	P05
2253	2000.0	RISC	270 (Int: FC#1520 = 70.0)	P20
2431	465.0	RISC	270 (Int: FC#1518 = 30.0)	P10
2432	1070.0	RISC	270 (Int: FC#1519 = 50.0)	P10
2434	2350.0	RISC	270 (Int: FC#1520 = 70.0)	P20
2395	370.0	RISC	820 (Int: FC#1521 - #1524)[II]	P10-P20
2396	950.0	RISC	820 (Int: FC#1521 - #1525)	P20-P30
2397	2000.0	RISC	820 (Int: FC#1521 - #1526) (zwei CPUs)	P20-P30

[I]Die DSD-Modelle sind für Domino-Serverbetrieb optimiert und nicht für C/S oder Interaktiv-Betrieb zu empfehlen!
[II]Interaktivleistung nach Wahl: #1521 = 35.0 / #1522 = 70.0 / #1523 = 120.0 / #1524 = 240 / #1525 = 560.0 / #1526 = 1050 / #1527 = 2000

CPU-Typ	Leistung CPW	Typ	in welchem AS/400 Modell z.B.?	Release / Pxx
2398	3200.0	RISC	820 (Int: FC#1521 - #1527) (vier CPUs)	P30-P40
2435	600.0	RISC	820 (Int: FC#1521 - #1524)	P10-P20
2436	1100.0	RISC	820 (Int: FC#1521 - #1525)	P20-P30
2437	2350.0	RISC	820 (Int: FC#1521 - #1526)	P20-P30
2438	3700.0	RISC	820 (Int: FC#1521 – #1527)	P30-P40
2487	20200 – 29300	RISC	890 (16 bis 24 CPUs) verschiedenste Typen	
2488	29300 – 37400	RISC	890 (24 bis 32 CPUs) verschiedenste Typen	

Die letzten beiden Einträge habe ich nur mal ausser Konkurrenz mit eingebracht, um zu verdeutlichen, bis in welche Regionen die Leistung skaliert werden kann. Bezahlen kann das sowieso kaum jemand. Die letzte Maschine in der Liste, eine 890er mit #2488 CPU fängt bei ca. 1.5MIO Euro erst an (mit 16 CPUs, 16*1.5MB L2-Cache und 128MB L3-Cache, 16GB Hauptspeicher und zwei Platten zum Starten :-)

So, mit dieser Liste wirst Du auf jeden Fall die richtige Maschine ergattern, auch wenn sie nicht vollständig ist.

Mit dem Befehle

```
WRKHDWRSC *STG
```

siehst Du alle Ressourcen, die mit Speicher zu tun haben (Storage).

```
                        Mit Speicherressourcen arbeiten
                                                          System:   BLACKBOX
Auswahl eingeben und Eingabetaste drücken.
  7=Ressourcendetails anzeigen   9=Mit Ressource arbeiten

Opt  Ressource       Typ-Modell  Status               Text
     CMB02            6756-002   Betriebsbereit       Combined function IOP
       DC12           6713-050   Betriebsbereit       Steuereinheit für Platt
       DC09           6713-050   Betriebsbereit       Steuereinheit für Platt
       DC10           6713-050   Betriebsbereit       Steuereinheit für Platt
       DC11           6713-050   Betriebsbereit       Steuereinheit für Platt
       DC01           6390-001   Nicht gefunden       Bandsteuereinheit
       DC05           6321-002   Betriebsbereit       Optische Steuereinheit
       DC08           6385-001   Betriebsbereit       Bandsteuereinheit
```

Speicherressourcen eines Systems

Du siehst hier bereits bekannte Einträge der Festplatten. Zusätzlich ein Eintrag für ein CD-Laufwerk (IBM-Jargon „Optische Steuereinheit“) sowie zwei Einträge für ein Bandlaufwerk. Eines davon war auf dem System konfiguriert, ist aber derzeit nicht angeschlossen.

Kommunikative Informationen findest Du mit:

```
WRKHDWRSC *CMN
```

Dies zeigt eine Liste aller Kommunikationsanschlüsse (serielle Leitungen, Netzwerkkarten etc) an:

```
                          Mit DFV-Ressourcen arbeiten
                                                              System:   BLACKBOX
Auswahl eingeben und Eingabetaste drücken.
  5=Mit Konfigurationsbeschreibungen arbeiten   7=Ressourcendetails anzeigen

Aus-
wahl  Ressource       Typ   Status                Text
      CMB02           6756  Betriebsbereit        Combined function IOP
        LIN03         2723  Betriebsbereit        LAN-Adapter
          CMN04       2723  Betriebsbereit        Ethernet-Anschluß
        LIN02         2721  Betriebsbereit        DFV-Adapter
          CMN02       2721  Betriebsbereit        DFV-Anschluß
          CMN03       2721  Betriebsbereit        DFV-Anschluß
        LIN01         2720  Betriebsbereit        DFV-Adapter
          CMN01       2720  Betriebsbereit        DFV-Anschluß
        LIN04         2850  Nicht gefunden        E/A-Adapterkarte für Datei-S
          CMN05       6B00  Nicht gefunden        Virtueller Anschluß
        LIN05         285A  Nicht gefunden        LAN-Adapter
```

Kommunikationsanschlüsse

Hier siehst Du auf einen Blick folgendes: Das System hat mehrere Karten, die mit Kommunikation zu tun haben. Diese sind immer als LINxx beschriftet. Unterhalb jeder Karte wird der Anschluss CMNxx angezeigt. Somit siehst Du zum Beispiel die Karte LIN03 (ein Ethernet-Adapter) mit einem Anschluss Nummer CMN04. Darunter siehst Du eine Karte LIN02 mit zwei Anschlüssen CMN02 und CMN03. Der Typ #2721 besagt, dass es eine serielle Karte ist.

Unten erkennst Du ein LIN04 nebst Zubehör, das nicht gefunden wurde. Dies ist aber keine Kommunikationskarte, sondern ein IPCS (integrated PC Server). Also eine Steckkarte mit einem PC drauf, der als Netzwerkserver unter Windows, Novell Netware oder OS/2 dient. IBM bezeichnet das auch irgendwie als Kommunikationshardware.
In diesem Beispiel ist diese Karte konfiguriert, konnte aber nicht gefunden wurden. Man hat sie wohl ausgebaut.

Interessante Features:

Feature	***Beschreibung***
2721	Karte mit zwei seriellen Anschlüssen für Modem oder Direktverbindung
2723	Ethernetkarte 10MBit
2724	TokenRing Karte 4/16 Mbit
2728	Ethernetkarte 10/100MBit
2720	Karte mit Twinaxanschluss für Konsole und einer seriellen Leitung
2850	IPCS (eingebauter PC-Server)

Mit diesen Informationen kannst Du Dir schon mal einen recht guten Überblick verschaffen.

Wenn Du nicht selbst an die Maschine kommst, gibt es noch den sogenannten Konfigurationsausdruck. Was das ist? Eine zweiseitige Übersicht über die in der Maschine eingebaute Hardware. Dieser sieht zum Beispiel so aus: (sorry, etwas klein geraten)

```
     HARDWARE SERVICE MANAGER                        SYSTEM CONFIGURATION LIST       BLACKBOX    20.10.03  18:51:55   PAGE
SYSTEM TYPE / MODEL / FEATURE CODE / SERIAL NUMBER / RELEASE : 9401 / 150 / 2270 / 44-03D4A   / V4R5M0 (1)
                                                                    - - LOCATION - -
                                                 SERIAL     FRAME DEVICE CARD  RESOURCE   PART
DESCRIPTION                    TYPE-MODEL                    NUMBER      ID    POS.  POS.    NAME     NUMBER       LOGICAL ADDRESS
SYSTEM                         9401-150                      44-03D4A                        CEC01
 SYSTEM UNIT                   9401-150                      44-03D4A    1                   CEC01
 SYSTEM BUS                                                  00-0000000  1                   LB01                  1/  1/ /   - /  /  /  /  /
  CONTROL PANEL                2468-001                      00-0000000  1             P1    PN01       0000021F5772
  MULTIPLE FUNCTION PLANAR     2270-000          * <         10-8068042  1              1    MP02       0000090H9004 LOCATION =>              0
  MULTIPLE FUNCTION IOP        6756-002          * <         10-8068042  1              1    CMB02      0000090H9004 1/  1/0/  1- /  /  /  /  /
  SERVICE PROCESSOR            6756-002                      10-8068042  1              1    SP01
   COMMUNICATIONS PORT         2723-001                      10-0167120  1              1A   CMN04      0000023L4193 1/  1/0/  1-1/  /14/ 0/ 0/
   COMMUNICATIONS IOA          2723-001                      10-0167120  1              1A   LIN03      0000023L4193 1/  1/0/  1-1/  /14/ 0/  /
   COMMUNICATIONS CHANNEL      605A-001                      00-0167120  1              1A   CHN01                   1/  1/0/  1-1/  /14/ 0/ 0/
   COMMUNICATIONS IOA          2721-001                      53-7148779  1              1B   LIN02      0000021H8100 1/  1/0/  1-1/  /14/ 1/  /
   COMMUNICATIONS PORT         2721-001                      53-7148779  1              1B   CMN02      0000021H8100 1/  1/0/  1-1/  /14/ 1/ 0/
   COMMUNICATIONS PORT         2721-001                      53-7148779  1              1B   CMN03      0000021H8100 1/  1/0/  1-1/  /14/ 1/ 1/
   COMMUNICATIONS IOA          2720-001                      53-9085435  1              1C   LIN01      0000090H9189 1/  1/0/  1-1/  /14/ 2/  /
  COMMUNICATIONS PORT          2720-001                     53-9085435  1              1C   CMN01      0000090H9189 1/  1/0/  1-1/  /14/ 2/ 0/
  WORKSTATION IOA              266C-001                     53-9085435  1              1C   CTL03      0000090H9189 1/  1/0/  1-3/  /14/10/  /
  COMMUNICATIONS IOA           285A-003            ?        53-8244038  1              1D   LIN05      0000090H9214 1/  1/0/  1-1/  /14/ 5/  /
  COMMUNICATIONS IOA           2850-011            ?        53-8242112  1              1E   LIN04      0000061H0127 1/  1/0/  1-1/  /14/ 4/  /
  VIRTUAL PORT                 6B00-001            ?        10-8242112  1              1E   CMN05      000008193654 1/  1/0/  1-1/  /14/ 4/15/
  MAIN STORAGE                                64            00-00000    1              1U   MS01                    LOCATION =>              1
  MAIN STORAGE                                64            00-00000    1              1V   MS01                    LOCATION =>              1
  MAIN STORAGE                                64            00-00000    1              1W   MS02                    LOCATION =>              2
  MAIN STORAGE                                64            00-00000    1              1X   MS02                    LOCATION =>              2
  MAIN STORAGE                                64            00-00000    1              1Y   MS03                    LOCATION =>              3
  MAIN STORAGE                                64            00-00000    1              1Z   MS03                    LOCATION =>              3
 DISK UNIT                     6713-050               *     68-01FD306  1     1              DD003      59H6611      1/  1/0/  1-2/  / 0/ 1/ 0/
 DISK CONTROLLER               6713-050                     68-01FD306  1     1              DC12       59H6611      1/  1/0/  1-2/  / 0/ 1/  /
 DISK UNIT                     6713-050                     68-01880FB  1     2              DD007      59H6611      1/  1/0/  1-2/  / 0/ 2/ 0/
 DISK CONTROLLER               6713-050                     68-01880FB  1     2              DC09       59H6611      1/  1/0/  1-2/  / 0/ 2/  /
 DISK UNIT                     6713-050                     68-0189AD6  1     3              DD006      59H6611      1/  1/0/  1-2/  / 0/ 3/ 0/
 DISK CONTROLLER               6713-050                     68-0189AD6  1     3              DC10       59H6611      1/  1/0/  1-2/  / 0/ 3/  /
 DISK UNIT                     6713-050                     68-018E7D7  1     4              DD008      59H6611      1/  1/0/  1-2/  / 0/ 4/ 0/
 DISK CONTROLLER               6713-050                     68-018E7D7  1     4              DC11       59H6611      1/  1/0/  1-2/  / 0/ 4/  /
  TAPE UNIT                     6385-001          %          00-40673    1     5              TAP07                   1/  1/0/  1-2/  / 0/ 7/ 0/
  TAPE CONTROLLER              6385-001                     00-40673    1     5              DC08                    1/  1/0/  1-2/  / 0/ 7/  /
  OPTICAL STORAGE UNIT         6321-002          %          00-00000    1     6              OPT02      0000093H8055 1/  1/0/  1-2/  / 0/ 6/ 0/
  OPTICAL CONTROLLER           6321-002                     00-00000    1     6              DC05                    1/  1/0/  1-2/  / 0/ 6/  /
     HARDWARE SERVICE MANAGER                        SYSTEM CONFIGURATION LIST       BLACKBOX    20.10.03  18:51:55   PAGE
- - - - - LEGEND - - - - -
SPECIAL CHARACTER INDICATORS:
  * - LOAD SOURCE DISK UNIT
  % - POTENTIAL ALTERNATE IPL UNIT
  = - MOST RECENTLY USED ALTERNATE IPL UNIT
  & - ITEM IS NOT AN IBM PRODUCT (OEM)
  < - WORK STATION CONSOLE IOP
  ? - NON-REPORTING RESOURCE
NOTE:  THESE SPECIAL CHARACTER INDICATORS APPEAR NEXT TO THE TYPE-MODEL OF THE RESOURCE FOR WHICH THEY APPLY TO.
OPERATING SYSTEM INDICATOR:
  (1) - OS/400
  (2) - SSP
NOTE:  THIS OPERATING SYSTEM INDICATOR APPEARS NEXT TO THE RELEASE INFORMATION.

LOGICAL ADDRESS FORMAT = A/B/C/D-E/F/G/H/I/J
```

Damit sieht man wirklich alles auf einmal. Bitte den Verkäufer, so einen Ausdruck zu erstellen und Dir zu mailen. Wie? Sag ihm folgendes:

1) als QSECOFR anmelden
2) STRSST eingeben und DF drücken
3) Auswahl 1 (Start a service tool)
4) Auswahl 7 (Hardware Service Manager)
5) Dann Taste F6 drücken
6) Nun die Formatoptionen mit DF bestätigen
7) Mit zweimal F3 und DF das SST wieder verlassen
8) Mit wrksplf in die Druckausgaben schauen, da wird er diesen Ausdruck finden
9) Fertig!

Beachte eine Ungereimtheit: Die Speichermodule unter „`Main Storage`“ werden hier mit 6 Stück angegeben, das ist auch richtig, das das System 6*32MB Module enthält. In der Mitte des Ausdrucks steht aber die Zahl „`64`“. Es sind aber nicht 384, sondern 192MB Speicher eingebaut. Verlasse Dich lieber auf die Ausgabe von `WRKHDWRSC *PRC`

Anhang 2: Auf mein Kommando!

Auf besondere Anfrage gibt es hier einen kleinen Ausflug über die Administration einer AS/400 und Vorgehensweisen, die weniger den Programmierer, sondern einen Systemverwalter beschäftigen. Aus verständlichen Gründen ist dieser Teil etwas theoretischer gehalten als der Programmierteil. Zum Einen kannst Du nicht alle hier vorgestellten Befehle auf meiner AS/400 nachvollziehen, zum Zweiten sollte man genau wissen, was man tut. OS/400 fragt im Gegensatz zu Windows nicht nach, wenn der Administrator den Befehl zum Runterfahren gibt ;-) Details wird es in einem eigenen Buch geben.

Bei den hier besprochenen Kommandos gilt noch mehr, was schon für die im Programmierteil dargestellten Befehle erläutert wurde: Was ich hier schreibe und per Bildschirmfoto darstelle, kann sich von dem System, auf dem Du arbeitest, mehr oder weniger unterscheiden. Der Zweck ist gleich, die Vorgehensweise auch, aber es können Felder fehlen oder zusätzliche Felder vorhanden sein, oder die Beschriftung differiert. Ich schreibe dieses Manual mit Hilfe mehrerer Systeme, einmal mit einem CISC-System unter V3R2M0, einmal mit einem RISC-System unter V4R5M0 und mit einer Testmaschine unter V5R2M0. Zwischen diesen Betriebssystemversionen gibt es schon große Unterschiede, aber Du wirst flexibel genug sein, diese festzustellen und zu sehen, was auf Deinem System anders ist. Insbesondere sei für alle hier beschriebenen Befehle der beherzte Einsatz der F1-Taste nahegelegt. Zum Einen auf die Bildschirmüberschrift (fast immer in der Mitte der obersten Zeile), und zum anderen in jedem einzelnen Feld. Wenn Du ein Menü vor Dir hast, geht es auch auf den Menüeinträgen. Viele Fragen, die ich so per eMail kriege, kann man auch damit schon beantworten.

Wenn Du das Glück hast, an einer AS/400 mit Scheff-Rechten arbeiten zu können, wird es in vielen Fällen Deine eigene Maschine sein, und auf Grund der Gebrauchtmarktpreise gehe ich hier von einem älteren System aus. Daher beginne ich mit der plakativen Beschreibung dieser Version. Teile der Beschreibung entstammen einem älteren Buch von Frank Soltis – er möge mir das Kopieren verzeihen.

Wer darf was?

... und wer warum nicht? Eine einfache Frage mit doch nicht immer so einfachen Antworten. Wie auch auf anderen Systemen, sind fast alle Rechte an bestimmte Benutzerprofile gebunden, beziehungsweise von IBM so vorbelegt. Ausserdem gibt es unter OS/400 verschiedene Stufen der Sicherheit, unter der das Betriebssystem läuft. Das Sicherheitskonzept der AS/400 ist - weil es von Haus aus in das Betriebssystem eingebaut ist - vollständig implementiert, aber auch so fein und flexibel definiert, dass man ein grosses Buch nur zu dem Thema schreiben kann. Bitte nicht schlagen, wenn etwas hier fehlt.

Jede AS/400 kennt mindestens zwei für die Verwaltung des Systems notwendige Benutzer:

- QSECOFR
- QSYSOPR

Diese Benutzer sind von IBM immer eingerichtet und haben vorgegebene Arbeitsgebiete. Natürlich kann man deren Rechte auch auf andere Benutzerprofile übertragen, aber da sollte man schon wirklich wissen, was man tut. Man kann sich recht schnell die Finger verbrennen. Und wer eine gebrauchte AS/400 aufgetrieben hat, auf der zwar ein Betriebssystem installiert ist, aber das QSECOFR-Kennwort unbekannt ist, hat keine große Freude an dem System, da der einzige Nutzen der Verbrauch von Elektronen und vielleicht die Nutzung als Blumenvasenständer ist ;-) Hierauf sollte man also achten (Achtung, das war ein Wink mit dem Zaunpfahl). Wer Bänder mit dem Betriebssystem hat, kommt schon viel weiter.

QSECOFR

Bedeutet ausgeschrieben „Security Officer“. Da das Benutzerprofil von IBM erstellt wurde, fängt der Name mit dem Buchstaben Q an. QSECOFR ist der Boss, er darf alles. Unix-Freunde erkennen ihn als „root“ wieder, Windows-NT / 2000 – Benutzer kennen diesen Benutzer als „Administrator“. Dieses Profil sollte man nur verwenden, wenn man Sicherheitsaufgaben zu erledigen hat, wie z.B. Rechte vergeben, Systemkonfigurationen vornehmen etc. Das Kennwort von QSECOFR ist nach einer Neuinstallation auf „QSECOFR“ eingestellt und sollte sofort geändert werden, wenn auch andere Personen Zugriff auf das System haben. Vor allem sollte man das Kennwort dann für sich behalten, und es sollte nicht zu einfach gewählt und öfter geändert werden. Ich habe schon Systeme bei Kunden gesehen, bei denen das Kennwort für QSECOFR nur durch Hinzufügen eines zusätzlichen Buchstabens geändert wurde. Der Verantwortliche beim Kunden war natürlich blaß, als er dies von einem externen Programmierer nach einigen Minuten gezeigt bekam. Es soll sogar produktive Systeme geben, bei denen das QSECOFR-Kennwort überhaupt nicht geändert wurde. *Vorsicht!* Nicht auf dem nächsten System ausprobieren, das Dir unter die Finger kommt und vielleicht produktiv läuft. Du könntest Dir großen Ärger einhandeln.

QSYSOPR

Er ist der kleine Bruder von QSECOFR und hat weniger Rechte, was das Steuern des Systems und der Konfiguration betrifft. Allerdings ist er der Herr des Systems, was die Steuerung von Druckern, Bandlaufwerken und Jobs betrifft. Dieses Benutzerprofil ist mehr dafür gedacht, tägliche Aufgaben zum Steuern der Systemabläufe zu erledigen.

Wie sich die beiden Benutzer unterscheiden, werden wir gleich sehen.

Q...was?

Wie Du vielleicht schon festgestellt hast, gibt es auf einer AS/400 noch weitere Benutzerprofile, deren Namen mit Q beginnt. Diese haben fast immer weniger Rechte als die beiden obengenannten, sind aber auch für spezielle Aufgaben eingerichtet. Einige Dienste und Programme beziehen ihr Recht direkt von einem Benutzerprofil, bzw. die Programme laufen unter Verwendung eines solchen Profils. Somit kann man die Rechte schön eingrenzen und ungewollte Rechteübernahme verhindern. Weiterhin sind fast all diese Profile so eingestellt, dass sie sich nicht interaktiv anmelden können.

Woher nimmst Du Dir das Recht?

Jedes Objekt auf der AS/400 wird mit Rechten versehen, die es auf andere Objekte ausüben kann, sowie mit Rechten, die Benutzerprofile auf dieses Objekt haben. Letzteres ist natürlich der wichtige Teil. Die Rechteprüfung läuft *unterhalb* des Betriebssystems, daher kommt man da auch nicht so einfach drumherum. Wer unbedingt dem Betriebssystem (Benutzer QSYS) alle Rechte entziehen will, mag seinen Spaß daran haben. Beginnen wir mit den Grundlagen.

Anhang 3 : Am Anfang war das Recht

Viele Betriebssysteme wie Windows kennen keine tief verwurzelten Rechte. Die Grundlagen von Windows kommen von DOS, und DOS kannte keine Rechte. Jeder durfte alles, es herrschte sogenannte Anti-Autorität, wie damals in den 70ern :-) Daher gibt es immer wieder Probleme mit Berechtigungen und Zugriffen, da das Rechtesystem im Grunde nur „aufgesetzt“ ist.

Auf der AS/400 sind die Sicherheitsfunktionen schon beim Design des Systems eingeplant, jeder elementare Teil des Systems vertraut auf diese Rechte und benutzt sie. Daher ist die Berechtigungsvergabe und die daraus resultierende Sicherheit so durchgängig.

Jeder Objektzugriff wird vom Machine Interface, also dem Teil unterhalb des Betriebssystems auf das nötige Recht geprüft. Somit kann selbst das Betriebssystem nicht alle Rechte verwenden, wenn man dies will. Umgehen kann es diese Sperre auch nicht. Da die Objekte durch Software- und Hardware geschützt sind, sind tägliche Security-Meldungen durch sogenannte Buffer-Overruns wie im Windows- oder Unix-Bereich unbekannt (man möge mir das Gegenteil mitteilen.)

Der erste Schritt zur Sicherung einer AS/400 sind die so genannten

Sicherheitsstufen

Die AS/400 kennt 5 grundlegende Stufen der Systemsicherheit. Diese werden vom QSECOFR durch das Einstellen des Systemwerts[1] QSECURITY bestimmt. Die Stufen sind:

- **Stufe 10: Keine Sicherheit.**
  Unter dieser Sicherheitsstufe ist das System offen wie ein Scheunentor. Man benötigt kein Kennwort beim Anmelden, und jeder Benutzer hat alle Rechte für alle Objekte. Kein Systemobjekt ist gegen Manipulation geschützt. Ein Benutzer benötigt kein Benutzerprofil, er muß beim Anmeldebildschirm nur seinen Namen angeben, es wird automatisch ein Profil erstellt. Nur untereinander können sich Benutzer nicht die Jobs stören. Diese Stufe würde nur Sinn machen, wenn am System ein Bildschirm direkt angeschlossen ist und der Rechner in einem abgeschlossenen Raum steht.
- **Stufe 20: Passwort-Sicherheit.**
  Diese Stufe ähnelt Stufe 10, allerdings muß der Benutzer ein Kennwort beim Anmelden eingeben. Einige Rechte können über das Benutzerprofil dem Anwender vorenthalten sein. Diese Stufe ist auch nicht wirklich sinnvoll, da man recht einfach mehr Schaden anstellen kann, als der QSECOFR will.
- **Stufe 30: Ressourcen-Sicherheit.**
  Jeder Benutzer hat ein eigenes Profil, er benötigt ein gültiges Kennwort und kann detaillierte Rechte für einzelne Objekte haben. Diese Stufe ist Minimalstandard für eine AS/400 und reicht für lokale Anwendungen über dumme Terminals im firmeninteren Netzwerk oft aus. Soll aber über Netzwerk oder das Internet auf das System zugegriffen werden (Netzwerkdienste, Client/Server etc.), so wird auch diese Stufe nicht mehr ausreichen.
- **Stufe 40: Gesichertes Betriebssystem.**
  Hier wurde von IBM bei der Umstellung auf RISC und Client/Server-Computing großes Augenmerk auf die Systemobjekte gelegt. Ab sofort muss auch QSECOFR sein Kennwort regelmäßig ändern. Spezielle Benutzerklassen regeln weit abgesteckte Rechtegruppen. Ab jetzt laufen Programme entweder im Systemstatus oder Anwenderstatus, so daß erkannt werden kann, ob ein Programm überhaupt die Möglichkeit hat, Systemobjekte zu verwenden, auch wenn es ausdrückliches Recht dazu hätte. Ein Anwenderobjekt kann nicht einfach so auf ein Systemobjekt zugreifen, selbst wenn QSECOFR das entsprechende Programm gestartet hat. Wem dies auch nicht reicht, greift zu Stufe 50.

[1]Ein Systemwert ist eine sytemweit gültige Variable zur Festlegung des Verhaltens des Systems. Beispiel: Uhrzeit oder Systemname.

- **Stufe 50: C2-Sicherheit.**
  Die Krönung der Sicherheitsstufen, zertifiziert von US-Regierungsstellen für höchste Sicherheit für kommerzielle Rechnersysteme. Hier kann der Besitzer der Systemressourgen detailliert entscheiden, wer das Recht auf welche Systemressourcen hat. Es kann jederzeit rekonstruiert werden, wer wann welches Objekt in welcher Art und weise verwendet hat. Diese Protokollierung kann vom Betriebssystem vorgenommen werden und produziert auf einem grossen, ausgelasteteten System schon ein sehr großes Journal. Das Betriebssystem wird sehr stark abgekapselt. (Der Objektschutz vor Zerstörung eines Objekts oder Manipulation durch System- oder Speicherfehler ist allerdings schon in Stufe 10 integriert). Allerdings ist es hier mit normalen Programmiermitteln nicht möglich, über APIs auf besondere Systemfunktionen zuzugreifen. Hierzu benötigt es den speziellen MI-Compiler.

Benutzerprofile

Jeder Anwender auf der AS/400 benötigt ein Benutzerprofil, mit dem er sich gegenüber dem System identifiziert und über das die Zugriffsrechte definiert werden. Der Objektschutz geschieht auf unterster Softwareebene und liegt in der Architektur noch unterhalb des Betriebssystems. Folgende Möglichkeiten kann ein Anwender haben:

- Benutzerklasse – Hier wird der Benutzer einer besonderen Klasse wie „Programmierer“ oder „Systemverwalter“ zugeordnet und erbt entsprechend vorgegebene Rechte.
- Anmeldung einschränken – Ein Anwender wird in der Anmeldung auf eine Sitzung beschränkt.
- Berechtigung einschränken – Hierbei kann der Benutzer keine Systembefehle eingeben, sondern nur Auswahlen aus angezeigten Menüs durchführen. Sehr einfache und effektive Methode, „Spielkindern“ den Sand aus der Grube zu nehmen.
- Anfangsmenü und Startprogramm – Hier kann definiert werden, welches Menü der Benutzer nach dem Anmelden angezeigt bekommt, bzw. welches Programm nach dem Anmelden gestartet wird. Sehr nützlich in Verbindung mit eingeschränkter Berechtigung und einem guten Menüsystem.
- Aktuelle Bibliothek – Hier kann definiert werden, in welcher Bibliothek neu erstellte Objekte des Benutzers erstellt werden. Dieser Parameter wird oft und gerne vergessen, und der Administrator wundert sich dann über die Müllhalde in der Bibliothek QGPL.
- Kennwort – Ab Sicherheitsstufe 20 zwingend notwendig und muß nicht näher erklärt werden.
- Sonderberechtigungen und priviligierte Befehlsrechte – Man kann in einem Benutzerprofil durch gewisse Parameter Gruppenrechte und Sonderbefehle angeben. Gibt man einem normalen Benutzer zum Beispiel das Recht `*ALLOBJ`, darf er auf alle Objekte im System zugreifen!
- Objektberechtigungen – Im Benutzerprofil sind Rechte für bestimmte Objekte eingetragen.
- Eigene Objekte – Dem Profil ist eine Liste zugefügt, die alle Objekte enthält, die dem Benutzer gehören.
- Speicherplatzbenutzung – Definiert, wieviel Platz die Objekte des Benutzers maximal verwenden dürfen.
- Ablaufprioritätenlimit – Besonders auf Systemen mit sehr vielen Benutzern ist es oft notwendig, die Priorität wichtiger Benutzer bei der Planung von Aufgaben über die von normalen Benutzern zu stellen.

Benutzerklassen

Auf der AS/400 gibt es ähnlich der Sicherheitsstufe 5 Benutzerklassen, die auf einfache Art die Definition einer Benutzerstufe ermöglichen. Allerdings sollte man hierbei sehr vorsichtig sein und mit Rechten

sparsam umgehen. Nur, damit ein Benutzer einen Drucker steuern kann, muss er nicht zwangsweise das Recht für alle Drucker im System habe. Es gibt folgende Benutzerklassen:

- **Sicherheitsbeauftragter** – Dies ist die höchste Berechtigung im System und sollte üblicherweise dem Benutzer QSECOFR vorbehalten sein. Diese Personen haben Rechte auf alles und jeden und dürfen alle Sicherheitsfunktionen ausführen, auch das Erstellen neuer Benutzerklassen.
- **Sicherheitsadministrator** – Diese Gruppe hat das Recht, Profile zu erstellen und das Betriebssystem auf Band zu sichern. Hier sollte man auch Vorsicht walten lassen.
- **Systemprogrammierer** – nicht zu verwechseln mit dem normalen Programmierer. Der Systemprogrammierer kann Befehle mit Zugriff auf direkte System-APIs verwenden und somit dem System Ressourcen entziehen und die Leistung reduzieren.
- **Systembediener** – Diese Benutzergruppe enthält auch QSYSOPR. Der Benutzer darf Sicherungen von Objekten erstellen, Drucker steuern und Geräte ab- und anmelden.
- **Benutzer** – Die unterste Ebene der Benutzerklassen. Dieser Benutzer hat keine grossartigen Rechte auf Systemobjekte und Ressourcen, und kann am wenigsten Schaden anstellen.

Objekteigner und Objektberechtigungen

Das Eignerprofil beinhaltet zwei Listen, in denen zum Einen alle Objeke stehen, die das Profil besitzt, und zum Anderen alle Objekte stehen, auf die das Profil ein Zugriffsrecht hat. Beim Erstellen eines Objekts wird dieses automatisch in die Eignerliste eingetragen, selbstverständlich kann der Objektbesitz übertragen werden. Der Eigner kann andere Benutzer auf das Objekt berechtigen.

Objektberechtigungen

OS/400 definiert 4 vorgegebene Möglichkeiten, Zugriff auf ein Objekt zu regeln. Diese Möglichkeiten sind Kombinationen aus feiner definierten Rechten, die auch einzeln vergeben werden können.

- `*EXCLUDE` – Verweigert das Recht auf ein Objekt. Es hat Vorrang vor jedem anderen Recht, das ein Benutzer zum Beispiel über eine Gruppe oder die Angabe für „Jeder“ erlangt.
- `*USE` – Der Benutzer darf das Objekt verwenden und den Inhalt lesen.
- `*CHANGE` – Der Benutzer darf das Objekt verwenden, den Inhalt lesen, den Inhalt erweitern (z.B. Datensätze hinzufügen), löschen (Datensätze löschen) oder Bearbeiten (Daten verändern).
- `*ALL` – Der Benutzer hat alle Rechte, auch das Recht, das Objekt zu löschen. Beachte, daß das Recht „Inhalt löschen“ von *CHANGE nicht gleichgesetzt ist mit „Objekt löschen“!

Befehlsprivilegien und Sonderberechtigungen

Abschließend noch die Sonderwerte, die man besonderen Benutzern zuteilen kann:

- *ALLOBJ – Berechtigt das Profil für die Verwendung aller Objekte und Systemressourcen!
- *SECADM – Hiermit ermöglicht man dem Benutzer, Profile zu erstellen, zu ändern und zu löschen.
- *SAVSYS – Damit darf ein Anwender Sicherungs- und Wiederherstellungsbefehle auf alle Objekte im System anwenden, unabhängig davon, ob er selbst Zugriff aud diese Objekte hat. (!)
- *JOBCTL – Jobsteuerung ermöglicht dem Benutzer, Jobs zu Ändern, Anzuzeigen, Abzubrechen und zu löschen.
- *SERVICE – Der Benutzer darf jetzt Wartungsbefehle ausführen und zum Beispiel ein angeschlossenes Modem zur Abholung von Patches verwenden.
- *SPLCTL – Spoolsteuerung ermöglicht dem Anwender, alle Spoolfunktionen auszuführen.
- *AUDIT – Protokollierung bedeutet, daß der Benutzer mit diesem Recht ein Überwachungsprotokoll für Objekte oder Profile anzeigen, erstellen und die Steuerung dessen verändern kann.
- *IOSYSCFG – Damit wird dem Benutzer erlaubt, Geräte am System zu steuern.

So, nun haben wir uns genug mit der Theorie der Berechtigungen beschäftigt, greifen wir nun aktiv ins Geschehen ein.

Ich gehe in diesem Bereich davon aus, dass Du weisst, wie man Befehle im System eingibt, die Parameter promptet und wo die Tasten F1 und F4 auf Deiner Tastatur sind ;-) Weiterhin gehe ich davon aus, dass Du Zugriff auf ein System hast, auf dem Du administrative Tätigkeiten durchführen *darfst!* Ich übernehme keinerlei Verantwortung für Fehler und Ausfähle von Produktivsystemen, die durch Studium dieses Manuals und Ausprobieren entstehen!

Mir ist es auch schon einmal passiert, dass ich einem Auszubildenden den Befehl PWRDWNSYS (System herunterfahren) erklären wollte und dazu diesen eingetippt habe. Ich konnte so schnell nicht schauen, wie der auf Datenfreigabe drückte. In diesem Fall hilft nur eins: Tür von innen verschließen und den Telefonhörer neben den Apparat legen ;-)

Du wirst Dich nun an Deinem System mit QSECOFR und dem passenden Kennwort anmelden. Ich gehe nun davon aus, dass Du ein gebrauchtes System vor Dir hast und es erforschen willst. Wenn Du gerade im Büro sitzt und den Verwalter Deiner AS/400 fragst, ob Du mal eben das QSECOFR-Kennwort haben kannst, halte bitte einen Fotoapparat bereit und sende mir das Ergebnis :-) Ab hier gibt es kein Zurück mehr!

PS: Solltest Du über eine AS/400 mit einen Satz Systembänder verfügen, und weißt nicht, wie man das System installiert, keine Sorge, auch darüber ist ein Kapitel geplant.

Arbeiten mit Benutzerprofilen

Schauen wir zunächst einmal nach, welche Benutzerprofile auf Deiner AS/400 existieren. Dazu verwendest Du den Befehl

`WRKUSRPRF *ALL` (WoRK with USeR PRoFiles).

Wenn Du `*ALL` auslässt, fragt das System, mit welchen Benutzerprofilen Du arbeiten willst. Du kannst hier eine generische Angabe machen, also den Teil, mit dem die Profilnamen anfangen sollen, gefolgt von einem Stern. Beispiel:

Mit Profilen arbeiten

`WRKUSRPRF Q*` (zeigt Dir alle Benutzerprofile, die mit Q anfangen)
Blättere nun soweit, bis Du das Profil `QSECOFR` siehst und bearbeite dieses mit der Auswahl `2` davor.

```
                        Benutzerprofil ändern (CHGUSRPRF)

Auswahl eingeben und Eingabetaste drücken.

Benutzerprofil . . . . . > QSECOFR       Name
Benutzerkennwort . . . . . .   *SAME_____    Name, *SAME, *NONE
Kennwort auf abgelaufen setzen     *NO__       *SAME, *NO, *YES
Status . . . . . .   *ENABLED_    *SAME, *ENABLED, *DISABLED
Benutzerklasse . . . . . .   *SECOFR       *SAME, *USER, *SYSOPR...
Unterstützungsstufe  . . . . . .   *INTERMED     *SAME, *SYSVAL, *BASIC...
Aktuelle Bibliothek  . . . . . .   *CRTDFT___    Name, *SAME, *CRTDFT
Aufzurufendes Startprogramm  . .   *NONE_____    Name, *SAME, *NONE
  Bibliothek . . . . . .     __________  Name, *LIBL, *CURLIB
Anfangsmenü  . . . . . .   MAIN_______    Name, *SAME, *SIGNOFF
  Bibliothek . . . . . .     *LIBL_____  Name, *LIBL, *CURLIB
Möglichkeiten einschränken . . .   *SAME___      *SAME, *NO, *PARTIAL, *YES
Text 'Beschreibung'  . . . . . .   'Sicherheitsbeauftragter'____________________
______


                                                                          Ende
F3=Verlassen   F4=Bedienerf.   F5=Aktualisieren   F10=Zusätzl. Parameter
F12=Abbrechen   F13=Verwendung der Anzeige   F24=Weitere Tasten
```

Bearbeiten des Benutzerprofils QSECOFR

Du siehst nun den Bildschirm, auf dem Du die Eigenschaften des Benutzerprofils anpassen kannst. Die vorhandenen Eigenschaftswerte sind in die Felder bereits eingetragen.

Viele Parameter müssen hier wohl nicht näher erklärt werden, im Zwiefelsfall hilft die Onlinehilfe hier großarti weiter. Hervorzuheben ist an dieser Stelle zunächst die Taste `F10`, die – wie auch in vielen anderen Bildschirmen weitere Parameter anzeigt. Du musst dann mit BildHoch und BildRunter zwischen mehreren Bildschirmseiten blättern, um alle Parameter zu sehen.
Ebenso existiert in fast jedem Bildschirm die Taste `F11` (siehe `F24` = weitere Tasten). Diese Taste wechselt zwischen der Anzeige möglicher Feldinhalte (rechts im Bildschirm) und den Schlüsselwörtern (dann links vor dem jeweiligen Eingabefeld). Somit kann man sich schnell einen Überblick über die Parameternamen verschaffen, so daß man später nicht immer den Befehl prompten muß, um nur einen Parameter zu ändern.

Parameter für Benutzerprofile

Hier einige Parameter, auf die ich Deine Aufmerksamkeit etwas lenken will:

- **Unterstützungsstufe (ASTLVL):**
  Hiermit kann man für viele Systemanzeigen einstellen, ob der Benutzer mehr oder weniger ausführliche Angaben sowie Funktionstasten und Auswahlen angezeigt bekommt. Diesen Wert kann der Anwender später selbst ändern, es gibt in vielen Bildschirmen dafür eine Funktionstaste. Meistens reicht für alle Zwecke der Wert *INTERMED, der eine gute Mischung aus Information und Benutzerführung bietet.
- **Anfangsmenü (INLMNU):**
  Hier steht üblicherweise MAIN als das Startmenü (kann auch mit dem Befehl GO MAIN aufgerufen werden). Du kannst hier ein eigenes Menü eintragen, wenn Du eins erstellt hast. Benutzerprofile, die für Dienste verwendet werden (z.B. QTMHHTTP für den HTTP-Server) haben hier den Eintrag *SIGNOFF. Dies hat zur Folge, daß sie sich nicht interaktiv anmelden können.
- **Sonderberechtigung (SPCAUT):**
  Hier werden die Werte für Sonderberechtigungen eingetragen, die der Benutzer erhalten soll. Dieses Feld ist ein besonderes Feld. Du siehst auf dem Bildschirm zwei Zeilen, in die Du zwei einzelne Werte eintragen kannst. Da für diesen Parameter aber mehr als zwei Angaben gleichzeitig gemacht werden können, reicht der Platz nicht aus. Daher kannst Du (wie auf dem Bildschirm beschrieben), in eines der beiden Felder ein Pluszeichen + eintragen und DF drücken. Es erscheint ein eigener Bildschirm mit weiteren Eingabefeldern.
- **Maximal zulässiger Speicher (MAXSTG):**
  Gibt an, wieviel Speicher das Profil auf dem System maximal verwenden darf. Wird der zulässige Speicher bei einer Objekterstellung überschritten, wird eine Fehlermeldung an den Job gesendet und das Objekt nicht erstellt. Besonders bei Profilen für Dienste sollte man auf genügend Speicher achten, oder *NOMAX angeben.
- **Gruppenprofil (GRPPRF):**
  Um wie unter Unix Gruppenzugehörigkeiten zu verwenden, geht man wie folgt vor: Zunächst erstellt man ein Benutzerprofil mit bestimmten Rechten. Dieses Profil betrachtet man als Gruppenprofil und trägt dessen Namen unter GRPPRF ein. Das aktuelle Profil erbt dann die Berechtigungen dieses Gruppenprofils. Es gibt weitere Parameter um diese Vererbung zu steuern. Vorsicht! Der hier gemachte Eintrag wird erst aktiv, wenn sich der Benutzer des aktuellen Profils ab- und wieder anmeldet!
- **Eigner (OWNER):**
  Gibt an, wer der Eigner neu erstellter Objekte durch das aktuelle Profil wird. Der Wert *USRPRF definiert das aktuelle Profil als Besitzer eines Objekts. Gehört das aktuelle Profil zu einem Gruppenprofil, kann hier *GRPPRF angegeben werden, um die gesamte Gruppe zum Eigner zu machen. Letzter Parameter macht natürlich nur Sinn, wenn das aktuelle Profil auch ein Gruppenprofil definiert hat :)
- **Gruppenberechtigung (GRPAUT):**
  Auch dieser Parameter macht nur Sinn, wenn ein Gruppenprofil definiert ist. Weiterhin darf der Parameter nur angegeben werden, wenn OWNER dann auf *USRPRF steht. Mit GRPAUT kann definiert werden, welche Berechtigung das Gruppenprofil auf vom aktuellen Profil auf neu erstellte Objekte erhält. Somit kann zum Beispiel Benutzer MAIER, dem das Gruppenprofil VERTRIEB zugeordnet ist, ein Objekt erstellen. Als Ersteller erhält MAIER dann *ALL Berechtigung auf das Obhekt, und die Gruppe VERTRIEB zum Beispiel nur *USE.

Ein Profil erstellen

Nun hast Du gesehen, wie Du mit dem Befehl `WRKUSRPRF` mit Benutzerprofilen arbeiten kannst. Dieser Befehl ist nur zur Anzeige einer Liste von Profilen zuständig. Wenn Du mit Auswahl `2` ein Profil

anwählst, wird der Befehl `CHGUSRPRF` gestartet. Diesen kannst Du natürlich auch direkt eingeben, Du landest im gleichen Bildschirm wie eben.

Ein neues Profil erstellt man mit `CRTUSRPRF`, löschen kannst Du ein Profil mit `DLTUSRPRF`.

Somit erkennst Du auch den Zusammenhang der Befehlsbenennung:

`CRTXXX` - Erstellen
`CHGXXX` - Ändern
`WRKXXX` - Mit einer Liste von xxx arbeiten
`DLTXXX` - Löschen

Diese Kombination gilt für alle Objekte und Befehle. Deiner Experimentierfreude soll kein Hindernis in den Weg gestellt werden :)

Anhang 4 – Wie man einen Ausdruck des Compilers liest

Umwandlungsfehler – und Behebung

Jedesmal, wenn Du mit der Auswahl 14 im PDM oder dem entsprechenden CRT- oder CHG- Befehl dem Compiler etwas zu tun gibst, hinterläßt er Dir eine Menge Informationen über den Erfolg der Arbeit. Mit der Zeit sammeln sich eine Menge sogenannter Spoolausgaben. Spool heissen diese deshalb, da sie eigentlich ganz normale Druckausgaben sind, aber vom System immer erst gesammelt werden, damit man die Druckausgaben auch schön verwalten kann. Ausserdem braucht man keinen Drucker, um eine Druckausgabe erstellen zu können.

Nachdem Dein Compiler die Arbeit (mehr oder weniger erfolgreich) beendet hat, erhälst Du im der Fußzeile des Bildschirms nur eine Meldung, ob die Umwandlung erfolgreich war.

Hinweis: Solltest Du eine Nachricht wie unten gezeigt erhalten, ist Dein Benutzerprofil so eingestellt, dass Umwandlungen im SEU im Hintergrund gestartet werden. Ich empfehle für Einsteiger, im SEU mit F18 den Parameter „Umwandlung im Stapelbetrieb“ auf N zu setzen.

Hier ein Beispiel einer Nachricht, wenn etwas schiefgelaufen ist und Deine Umwandlung im Batch läuft:

```
                              Nachrichten anzeigen
                                                        System:   S4404756
Warteschlange . :   HOLLE                    Programm. . . . :   *DSPMSG
  Bibliothek  . :     QUSRSYS                  Bibliothek  . :
Bewertung . . . :   00                       Zustellung. . . :   *BREAK

Antwort eingeben (falls erforderlich) und Eingabetaste drücken.
  Abnormale Beendigung des Jobs 000811/HOLLE/KONTOBS.
...
```

So sieht es aus, wenn Du im interaktiven Modus umwandelst und ein Fehler auftritt:

```
                    Mit Teildateien arbeiten (mittels PDM)            S4403D4A

Datei . . . . . .   QRPGLESRC
  Bibliothek  . .     HHBUCH               Listenanfang bei . . .

Auswahl eingeben und Eingabetaste drücken.
 2=Editieren    3=Kop.  4=Lösch. 5=Anzeigen      6=Drucken   7=Umbenennen
 8=Beschreibung anz.    9=Sich. 13=Text ändern  14=Umwand.  15=Modul erst. ...

Ausw. Teildatei  Art         Text
14   BUCHEIN     RPGLE       BUCHUNGSEINGABE









                                                                        Ende
Parameter oder Befehl
===>
F3=Verlassen     F4=Bedienerführung   F5=Aktualisieren     F6=Erstellen
F9=Auffinden     F10=Befehlseingabe   F23=Weitere Angaben  F24=Weitere Tasten
Umwandlung gestoppt. Fehler mit Bewertungsstufe 30 im Programm gefunden.     +
```

Du siehst eine entsprechende Meldung am Bildschirmfuss, die Auswahl 14 vor der Teildatei bleibt bestehen, damit Du sofort siehst, dass etwas nicht stimmt.

Das hört sich recht fatal an. In diesem Falle steht in Deiner Spooldatei eine Listenausgabe mit allem, was der Compiler so getrieben hat.

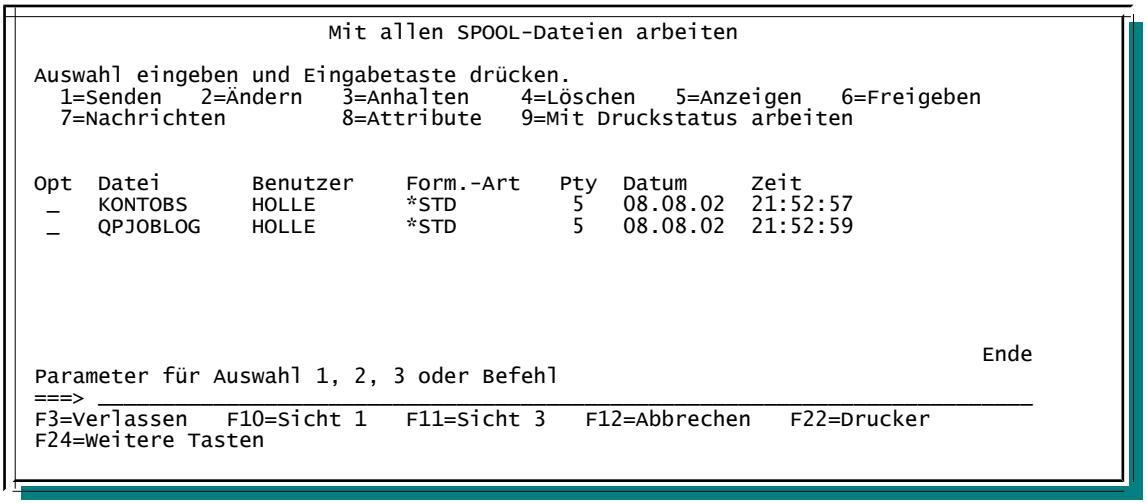
Wo ist mein Druck?

Um diese (oder jede andere) Spooldatei anzusehen, gib einfach in die Befehlszeile ein:

```
WRKSPLF
```

Es erscheint ein Bildschirm mit einigen Einträgen (nach längerem Arbeiten können das schon eine ganze Menge sein, so dass Du blättern musst. Die einzelnen Ausdrucke sind sortiert nach Verarbeitungsstatus und Datum, Deine eben erzeugte Ausgabe sollte also ganz unten in der Liste stehen. Sollte die Liste über mehrere Bildschirme gehen, blättere bitte, oder verwende die Taste F18 (siehe: F24) um an das Ende der Liste zu gelangen.

Du siehst in Spalte „Datei“ den Namen des Objekts, das Du umwandeln wolltest, Deinen Benutzernamen sowie ein paar weitere Angaben, die uns an dieser Stelle nicht unbedingt verwirren sollen. Drückst Du auf die Taste F11, siehst Du Datum und Uhrzeit, zu der die Ausgabe erstellt wurde.



Hier siehst Du meinen Bildschirm mit meinen (zugegeben recht wenigen) Spoolfiles. Wir haben hier zwei Einträge, die erstellt wurden, als ich erfolglos das Objekt „KONTOBS“ erstellen wollte. Warum zwei? Ganz einfach: Ich habe die Umwandlung im Stapelbetrieb (Batch) laufen lassen. OS/400 hat die Angewohnheit, nach der Beendigung eines Jobs ein Protokoll über dessen Ablauf zu erstellen. Da ein Batch-Job ein eigener Job ist, haben wir somit einmal die Ausgabe des Systems über den Verlauf des Stapeljobs, und einen über den Verlauf der Umwandlung an sich (in einem Stapeljob kann man ja auch viel mehr als nur einen einzigen Befehl unterbringen). Letztere Ausgabe ist die einzige, die Du erhälst, wenn Du interaktiv umwandelst. (Wo man das einstellt? Drücke im PDM mal auf F18, da kannst Du viele Parameter zu Deiner PDM-Umgebung einstellen.)

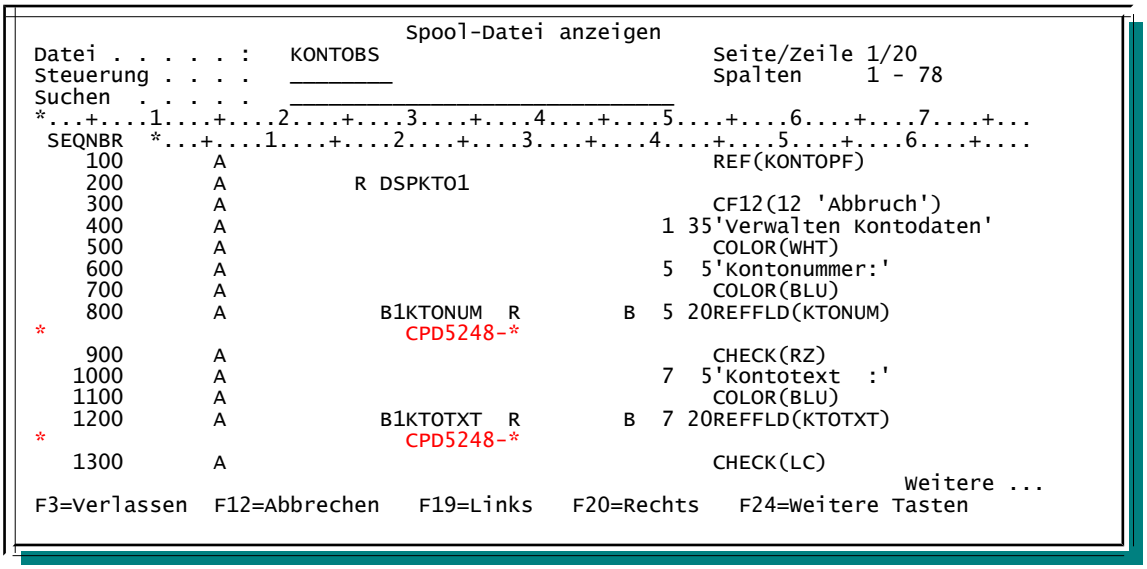
Beachten wir nur die für uns derzeit interessante Ausgabe in der Datei „KONTOBS“. Den Inhalt eines Spools kann man mit der Auswahl 5 anzeigen lassen. Der Bildschirm zeigt ein neues Bild.

Abhängig davon, ob Dein Terminal 80 oder 132 Spalten anzeigen kann, wirst Du evtl. nur einen Teil der Ausgabe sehen. Compilerausgaben sind fast immer in 132 Spalten erstellt, damit mehr Informationen angegeben werden können. Wenn Du nur 80 Spalten gleichzeitig sehen kannst, mußt Du mit den Tasten F19 und F20 die Ausgabe horizontal verschieben.

Als Beispiel, und weil die Twinaxkarte in meinem alten PC nur 80 Zeichen darstellen kann, zeige ich Dir nun die Druckausgabe mit einem 80-Spalten-Bildschirm.

Zunächst gibt uns der Compiler einen Überblick über die Aufgabe, die Du ihm gestellt hast, also was er wann wie umwandeln sollte.
Blätterst Du nach unten, wird der gesamte Quellcode aufgelistet. Oben am Bildschirm siehst Du rechts von den beiden Eingabefeldern die Position der Ausgabe, die gerade gezeigt wird.

In meinem Beispiel habe ich einmal geblättert, so dass die Ausgabe auf Seite 1 ab Zeile 20 beginnt.



Hier beginnt der Quelltext in der Spoolausgabe. Der Compiler numeriert die Zeilen ähnlich wie im Editor, nur wird hier der Punkt weggelassen.
Du siehst unterhalb der Zeile 800 in der ersten Spalte ein Sternchen (*). Damit will der Compiler Dir mitteilen, dass er an dieser Stelle im Quellcode ein Problem hat. In der Zeilenmitte steht eine Fehlernummer (hier: CPD5248), gefolgt von einem weiteren Stern. Die Position des zweiten Sterns verrät Dir, wo der Fehler in der Zeile darüber genau liegt, in meinem Beispiel hakt der Compiler am „R“, das für „Referenziertes Feld“ steht.

Unterhalb der Zeile 1200 moniert der Compiler genau den gleichen Fehler an, wieder ein Problem mit der Referenzierung. Geübte Benutzer wissen schon, was das Problem ist. Da wir aber genau nachforschen wollen, müssen wir weiter runterblättern. Der Compiler sammelt die Fehler und gibt sie im Block am Ende der Druckausgabe an. Blättere also nun an das Ende der Ausgabedatei. Wenn Du faul bist, gibst Du in das erste Eingabefeld oben ein E ein. Warum, das verrät Dir die Taste F1. (Ich erspare mir die Erklärung der Steuerungsbefehle, der Hilfstext sagt alles. Ebenso schau Dir mal die Hilfe zum Suchen-Feld an).

Wenn Du am Ende der Datei angelangt bist, siehst Du unter dem Wort „Nachrichten“ eine Liste aller Fehlernummern mit kurzem Text. Unter `CPD5248` steht dort (die anderen Fehler sind Folgefehler, daher ignorieren wir sie):

```
 * CPD5248      30        3      Nachricht . . . :   Im Schlüsselwort REF oder REFFLD
angegebene Datei nicht gefunden.
```

Nun müsste es bei Dir klingeln. Ich referenziere auf eine externe Datei, und die kann der Compiler nicht finden. Warum? Nun, bestimmt liegt die Datei in einer Bibliothek, die nicht in meiner Bibliotheksliste steht.
In meinem Falle reicht ein einfaches

```
ADDLIBLE LIBMANTEST
```

und der Compiler wird Erfolg haben.

Anhang 5 – Die Beispielbibliothek

Damit Du nicht so viel tippen musst, habe ich die hier im Manual beschriebenen Beispielprogramme auf der öffentlichen IBM i („ipublic.online“) in eine eigene Bibliothek gestellt. Von dort aus kannst Du Dir die Quelldateien in Deine eigene Bibliothek kopieren und damit experimentieren. Diese Bibliothek heisst `LIBMANUAL`. Um Dir von dort Beispiele zu kopieren, schaust Du zunächst, was sich darin so befindet.

Einerseits findest Du in dieser Bibliothek einige Codefragmente, die generell interessant sein können, sowie die Quellcodes der in diesem Buch beschriebenen Beispielprogramme. Um die Übersicht zu wahren, habe ich für jedes Projekt oder jeden Abschnitt eine eigene Quelldatei erstellt. Aus dieser solltest Du die jeweiligen Quellcodes in die entsprechenden Dateien in Deiner Bibliothek kopieren. Du kannst auch alle Quellen in einer einzigen Quellendatei sammeln, allerdingst musst Du dann mit der Unordnung leben.

Weiterhin werden für jedes Projekt so genannte Sicherungsdateien angelegt, in der die Quelldatei ebenso liegt, falls Du die Quellen auf Dein System übernehmen willst.

Überblick der Bibliothek LIBMANUAL

Mit dem Befehl:

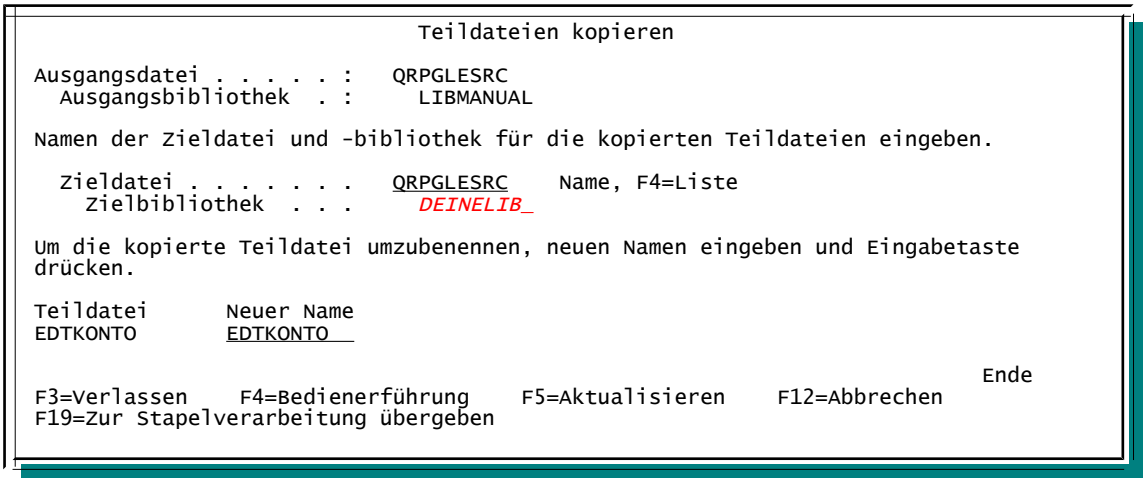
```
WRKLIBPDM LIBMANUAL
```

startest Du den PDM direkt in der Bibliotheksauswahl. Vor den Eintrag der Bibliothek die Auswahl `12`, und Du siehst die einzelnen Quelldateien (`QDDSSRC`, `QRPGLESRC` etc). Davon wählst Du Dir die gewünschte (bzw. die von mir angegebene) Datei aus, und schaust mit erneuter Auswahl `12` die dort enthaltenen Teildateien an.

```
                    Mit Teildateien arbeiten (mittels PDM)           MY150

Datei . . . . . .   QRPGLESRC
  Bibliothek  . .     LIBMANUAL            Listenanfang bei . . .

Auswahl eingeben und Eingabetaste drücken.
 2=Editieren    3=Kop.  4=Lösch. 5=Anzeigen      6=Drucken   7=Umbenennen
 8=Beschreibung anz.    9=Sich. 13=Text ändern  14=Umwand.  15=Modul erst. ...

Ausw. Teildatei  Art         Text
__   EDTKONTO    RPGLE       Mein erstes interaktives Programm

                                                                       Ende
Parameter oder Befehl
===>
F3=Verlassen     F4=Bedienerführung   F5=Aktualisieren     F6=Erstellen
F9=Auffinden     F10=Befehlseingabe   F23=Weitere Angaben  F24=Weitere Tasten
```

Nun kannst Du die gewünschte Teildatei mit Auswahl 3 („Kop.“ ist die Abkürzung für Kopieren :-) kopieren.

Das System fragt nach Zieldatei und Bibliothek, sowie einem neuen Namen für die Teildatei.



Hier reicht es, wenn Du den Namen der Bibliothek änderst (in den Namen Deiner Bibliothek). Die Datei QRPGLESRC sollte hier dann bereits existieren.
Schon hast Du die Kopie in Deiner Bibliothek und kannst damit arbeiten.

Inhalt

QEDTKTO

In dieser Datei sind die Quellen für das EDTKTO-Beispiel enthalten:

Teildatei	*Typ*	*Beschreibung*
EDTKTO	RPGLE	RPG-Programmquelle
KONTOBS	DSPF	Bildschirmdatei
KONTOPF	PF	Datendatei Konto
PRTKONTO	RPGLE	Druckprogramm Kontoinformationen

QHHBUCH

In dieser Datei sind die Quellen für das HHBUCH-Projekt enthalten:

Teildatei	*Typ*	*Beschreibung*
#REF	PF	Referenzdatei
BUCHDSP	DSPF	Bildschirmdefinitionen
BUCHEIN	RPGLE	Programm Buchungseingabe
BUCHPF	PF	Buchungsdatei physisch
KATPF	PF	Kategoriendatei physisch
KONTPF	PF	Kontendatei physisch
PERPF	PF	Periodendatei physisch

Anhang 6 – RPG im Überblick

Hier findest Du einen Überblick über die Programmiersprache RPG an sich. Es ist eine Beschreibung dessen, was mit RPG machbar ist.

Überblick über die Programmiersprache RPG IV

Dieser Abschnitt enthält einen groben Überblick über die Eigenarten der Programmiersprache RPG, die diese von anderen Sprachen unterscheidet. Diese Grundlagen sind besonders wichtig für das Verständnis des Programmierens in RPG, besonders wenn man schon sehr tief in anderen Sprachen steckt und sich deren Eigenarten verinnerlicht hat. Die folgenden Punkte sind besondere Merkmale von RPG:

- Spezifikationsarten
- Der Programmzyklus
- Anzeigevariablen / Indikatoren
- Befehlscodes

Spezifikationsarten

RPG ist eine sehr alte Sprache, sie stammt aus einer Zeit, zu der nicht jeder Computer einen Bildschrim und eine Tastatur hatte, oder eine Benutzeroberfläche, über die man direkt mit dem System kommunizierte. Damals, als Lochkarten ganz modern waren, hat der Programmierer seine Befehle auf verschiedene Weisen auf einer Lochkarte, die automatisch oder manuell in der angegebenen (spezifizierten) Reihenfolge in den Computer eingegeben wurden. Die Ausgabe der Programme wurde meist auf einem Drucker ausgegeben. Daher bezeichnet man bei RPG die einzelnen Befehls- oder Definitionszeilen auch als „Spezifikationen“.

Auch heute müssen diese Spezifikationen (von denen es verschiedene Arten gibt) in einer bestimmten Reihenfolge dem System übergeben, also in den Quelltext eingegeben werden. Aufgrund des Ursprungs in der Lochkartenzeit muss man bei der Eingabe dieser Spezifikationen auch ein bestimmtes Format berücksichtigen. Diese Formatierung besagt, dass bestimmte Angaben einer Zeile in bestimmten Spalten eingegeben werden müssen! Eine Angabe, die um eine Spalte versetzt ist, kann eine falsche Interpretation des Systems oder eine Fehlermeldung hervorrufen. Das Eingabeprogramm SEU hilft Dir aber dabei, die richtige Spalte zu treffen :-)

Ab Betriebssystemversion V5 gibt es auch einen neuen RPG-Compiler, der zumindest bei den C-Spezifikationen (Commands, Befehle) eine formatierungslose (genannt „free form“) Eingabe ermöglicht. Dies erleichtert dem Programmierer die Eingabe komplexerer Befehle.

Als zusätzliche Schleuderfalle ist zu beachten, dass es innerhalb einer Spezifikationsart mehrere Formatierungsmöglichkeiten, bzw. unterschiedliche Spaltenaufbauten gibt. Zum einen ist dies darin begründet, dass bei der Erweiterung von RPG um neue Sprach- oder Funktionselemente die Spalten einer Zeile nicht mehr ausreichten. Weiterhin waren früher nur 6stellige Feld- und Dateinamen erlaubt, inzwischen darf man 10 Stellen angeben. Viele Spezifikationszeilen erstrecken sich über mehrere Zeilen auf dem Bildschirm, die unterschiedlichen Aufbaus sind.
Bevor man vor Verwirrung aufgibt, sollte man sich mit dem F-Befehl von SEU vertraut machen, dieser kann Dir für jede Art eines Spezifikationsaufbaus eine Formatierungshilfe und einen Prompt bieten.

Insgesamt gibt es sieben verschiedene Spezifikationsarten. In einem RPG-Programm müssen nicht alle diese Spezifikationsarten verwendet werden, sie sind alle optional. Hier eine Übersicht über die einzelnen Spefizikationsarten, in der Reihenfolge, in der Du sie eingeben musst:

Code	*Beschreibung der Spezifikationsart / Angabe*
H	**Kontrollspezifikationen** Diese Zeilen dienen dazu, den Compiler zu steuern. Hiermit kann man Angaben über die Compilierung und den Programmablauf festlegen, beispielsweise das im Programm gültige Datums- und Zeitformat, den Programmnamen, Umwandlungsparameter etc.
F	**Dateispezifikationen** In diesen Spezifikationszeilen gibst Du an, welche Dateien das Programm verwenden soll. Die Reihenfolge der F-Spezifikationen (oder F-Angaben) ist beliebig.
D	**Definitionsarten** Hiermit kannst Du Daten beschreiben, die in Deinem Programm verwendet werden. Dazu gehören beispielsweise Variablen, zusätzliche Felder, Datenstrukturen etc.
C	**Befehlsangaben (Commands)** In diesen Zeilen werden die Befehle (oder Commands, Calculations) angegeben, die Dein Programm durchführen soll. Ursprünglich stand das C für „Calculations“, also auf IBM-Deutsch „Rechenbestimmungen“. Inzwischen ist es üblich, das C für „Commands“ zu verwenden, da man in diesen Zeilen nicht nur rechnet, sondern auch Ein- und Ausgabeoperationen durchführt und den strukturellen Ablauf des Programms regelt.
O	**Ausgabebestimmungen** Hiermit kann man (neben der Angabe in einer F-Bestimmung) Ausgabeformate als Satzformate und Felder bestimmen. Hier kann man intern (also im Programm) bestimmen, wie eine Ausgabe (auf den Drucker z.B.) aussehen soll.

Schliesslich gibt es noch spezielle Bestimmungszeilen, die für Unterroutinen verwendet werden können. Zur Verwirrung des Programmierers kann man zum Einen innerhalb der normalen C-Bestimmungen eine Subroutine spezifizieren, zum Anderen kann man auch nach den O-Bestimmungen Subroutinen erstellen.

Letztere Variante hat den Vorteil, dass man hier für die Subroutine lokale Variablen verwenden kann, die als Übergabestruktur definiert sein können. Solltest Du also nach den O-Bestimmungen eine oder mehrere Subroutinen definieren, musst Du folgende Reihenfolge einhalten:

Code	*Bestimmungsarten für Subroutinen*
P	**Prozedurdefinition Anfang** Hier wird der Anfang der Definition einer Subroutine angegeben. Weiterhin kannst Du hier angeben, ob diese Routine exportiert werden kann (für modulare ILE-Programmierung)
D	**Definition lokaler Variablen** Hier beschreibst Du lokale Datenfelder, die nur in dieser Subroutine gültig sind.
C	**Befehle in der Prozedur** enthalten Befehle, die mit lokalen *und* globalen Daten arbeiten können
P	**Prozedurdefinition Ende**

Zyklusprogrammierung

Was ist denn nun bitte ein Zyklus? So wie man bei literarischen Werken, die eine grosse Zeitspanne umfassen, von einem Zyklus spricht[1], geht es auch in einem RPG-Programm immer um den Lauf der Dinge.

Ein Zyklus besagt, in welcher Reihenfolge was passiert, und wer diesen Zyklus steuert. Es gibt zwei Arten der Zyklussteuerung:

- Programmgesteuerter Ablauf
- Dateigesteuerter Ablauf (compilergesteuert)

Erstere Variante wirst Du als Programmierer von jeder anderen Programmiersprache kennen: Dein Programm wird gestartet, es führt Befehle aus, bearbeitet Daten, Du kannst den Ablauf der Dinge steuern und irgendwann, wenn Dir oder Deinem Programm danach ist, kann das Programm beendet werden.

Die zweite Variante, die man auch (als Unterscheidung zur „normalen“ Programmierung) schlicht „Zyklusprogrammierung“ nennt, ist genau anders: Hier steuerst Du oder Dein Programm nicht den Lauf der Dinge, sondern der Compiler übernimmt bei der Erstellung des Programms die Initiative und baut eine Logik, die eine Datei steuert. Diese Zyklusprogrammierung kann (muss aber nicht) auf einer so genannten Primärdatei beruhen. Das Programm übernimmt für diese Primärdatei die Logik des Satzlesens. Für jeden Satz können Befehle ausgeführt werden. Wurde der letzte Satz gelesen, ist Dein Programm beendet.
Man kann diese Zyklusprogrammierung auch ohne Dateien nutzen, dann musst Du dem Compiler aber unbedingt sagen, dass das Programm an einem gewissen Punkt enden muss.

Diese Zyklusprogrammierung stammt aus der klassischen Berichtsprogrammierung. Ein Bericht basiert fast immer auf einer Folge von Sätzen in einer primären Datei. Zuerst wird eine Überschrift produziert, dann jeder Satz gelesen, bearbeitet und gedruckt, und am Ende der Schluss des Berichts.

Sehen wir uns mal diese Zyklusprogrammierung etwas genauer an:

Folgende Schritte werden ausgeführt:

1) Kopf- und Detailzeilen bearbeiten, die besonders gekennzeichnet sind.
2) RPG liest den nächsten Satz aus der primären Datei
3) Aufgrund von speziellen Indikatoren werden Summenberechnungen durchgeführt.
4) Ausgabezeilen (speziell gekennzeichnet) werden ausgegeben
5) Wenn der Indikator LR (last record, letzter Satz verarbeitet) gesetzt ist, also kein weiterer Satz zur Verarbeitung vorhanden ist, wird das Programm beendet
6) In einem speziellen Verarbeitungsbereich kann man mit den gelesenen Daten arbeiten
7) Dateilberechnungen, die nicht speziell gekennzeichnet sind, werden verarbeitet.
8) Springe zu Schritt 1.)

[1]Man möge mir diese augenzwinkernde Formulierung verzeihen, ich lese gerade einen grösseren Stapel Bücher aus dem Science-Fiction-Bereich, und irgendwie gefiel mir dieser Vergleich.

Der erste und letzte Zyklus wird jeweils etwas anders behandelt. Im ersten Zyklus werden – vor dem Lesen des ersten Satzes der Primärdatei – folgende Schritte ausgeführt:
- Eingabeparameter bearbeiten
- Dateien öffnen
- Programmdaten initialisieren
- Alle Sätze ausgeben, die mit dem Indikator 1P (first page, erste Seite) gekennzeichnet sind
- Kopf- und Detailüberschriftszeilen der Ausgabebestimmungen bearbeiten. Beispielsweise sei hier an konstante Daten oder Überschriften wie Datum und Seitennummer gedacht.
- Weiterhin werden im ersten Zyklus Summenberechnungen und Summenausgaben nicht ausgeführt.

Im letzten Zyklus (wenn also keine Sätze mehr aus der Primärdatei gelesen werden können) werden die Indikatoren LR (last record, letzter Satz) sovie L1 bis L9 (Steuerebene, siehe später) auf *ON gesetzt. Das Programm führt alle Summenberechnungen und Summenausgaben aus, alle Dateien werden geschlossen und das Programm beendet.

Sollten Unterroutinen definiert sein, gilt für diese der Zyklus nicht.

Indikatoren / Anzeigevariablen

Indikatoren (oder Anzeigevariablen) kann man sich als Schalter oder Flaggen vorstellen. Es handelt sich hierbei um Felder vom Typ „ein Zeichen“, die entweder den Wert *ON („1“) oder *OFF („0“) enthalten können. Man benutzt sie um das Ergebnis einer Operation zu erfahren, oder den Ablauf von Programmen, Befehlen, Ausgaben oder Bildschirmen zu steuern. Mit Indikatoren kann man elegant (aber gelegentlich auch unübersichtlich) den Ablauf eines Programms steuern. Man sollte es nur nicht übertreiben, und zwingend dokumentieren, sonst verliert ein anderer Programmierer, der das Programm lesen muss, innerhalb weniger Zeilen den Überblick.

Indikatoren können als Variablen innerhalb der Definitionsspezifikationen erstellt werden. Weiterhin kann man Indikatoren benutzen, die von RPG selbst zur Verfügung gestellt, und entweder in einer Spezifikation oder vom RPG-Compiler selbst belegt werden. Jeder Indikator hat einen zweistelligen Namen, beispielsweise `LR`, `05` oder `H2`. Abhängig vom Namen ist auch die Verwendung. Für die Indikatoren gibt es in den Spezifikationen spezielle Stellen, in denen sie verwendet werden können. In einigen Spezifikationen kann man auch den speziellen Namen `*IN`xx verwenden, wobei xx für den zweistelligen Namen steht, also beispielsweise `*INLR`, `*IN05` oder `*INH2`.

Bestimmte Indikatoren werden vom RPG-Programm während des Programmablaufs selbsttätig verändert. Weiterhin kannst Du als Programmierer selbstverständlich jeden Indikator in C-Bestimmungen bearbeiten.

Der Gebrauch von Indikatoren sollte im normalen Programm zur Steuerung auf ein Minimum beschränkt werden, da sonst die Übersicht leidet. Weiterhin kann man Indikatoren auch schön zur Steuerung von Bildschirmdateien verwenden, wobei man es auch hier nicht übertreiben sollte. Generell sei gesagt, dass bei Verwendung von Indikatoren es nicht schadet, wenn man sich ein gewisses Regelwerk setzt. Beispielsweise könnte man festlegen, dass der Indikator 99 immer zur kurzfristigen Abfrage einer Dateioperation verwendet wird (deren Ergebnis nicht später noch einmal benötigt wird), während 90 bis 98 für längere Operationen verwendet werden. 01 bis 24 kann man als Alternative zu Kx verwenden, da es sprechender ist.

Das ILE-Konzept auf einen Blick

ILE bedeutet „Integrated Language environment“ und ist ein Framework von IBM zur sprachübergreifenden, modularisierten Programmierung. ILE bedeutet einen grossen Schritt in der Programmentwicklung, wenn man die statische Programmierung aus der „guten alten Zeit“ gewohnt ist. Bei vielen Programmierern ist noch die Denkweise aus der S/3x Zeit vorhanden, oder viele kennen die Möglichkeiten von ILE gar nicht.

Das Konzept und die Architektur von ILE wurde eingeführt, als die Systemumstellung von CISC auf RISC ins Haus stand, also schon einige Jahre her. Folgende Compiler gehören zur ILE Familie: ILE COBOL, ILE CL, ILE C, ILE RPG sowie VisualAge for C++.

Gegenüber dem OPM (old program model) wurden viele Erweiterungen in das Framework gebracht, die hier im Einzelnen vorgestellt werden sollen:

Erstellung von Programmen

Der Anwender erstellt mit ILE ein Programm mit den folgenden beiden Schritten:

- Mit dem Compiler aus dem Sourcecode ein Modul erstellen
- Kombinieren (binden) eines oder mehrerer Module in ein Programmobjekt

Wer schon früher mit RPG im OPM programmiert hat, kann ein Programm weiterhin in nur einem Schritt erstellen, in dem er den Befehl CRTBNDRPG (create bound RPG program, erstelle gebundenes RPG Programm) verwendet. Die Auswahl 14 im PDM vor Deinem Quelltext erledigt dies für Dich. Dieser Befehl erstellt ein temporäres Modul aus Deinem Quelltext und bindet dieses in ein Programmobjekt ein. Mit Hilfe eines so genannten Binderverzeichnisses (binder directory) kannst Du so auch mehrere Objekte in ein Programmobjekt zusammenfassen.

Als Alternative könntest Du auch die Schritte einzeln ausführen (oder von einem kleinen CL-Programm ausführen lassen). In dieser Variante kannst Du ein Modul erstellen, welches in mehreren Programmen verwendet werden kann, ein vorhandenes Modul in ein neues Programm einbinden, oder ein Modul aktualisieren, welches bereits von anderen Programmen verwendet wird, ohne diese Programme neu umzuwandeln. Da jedes Modul in einer eigenen Sprache verwendet werden kann, ist es so möglich, ein Projekt aus mehreren Sprachen zusammen zu setzen.

Die beiden nötigen Schritte sind:
- CRTRPGMOD (create RPG module, RPG-Modul erstellen) verwenden, um ein Modulobjekt zu erstellen. Dieses Modulobjekt kann nicht direkt ausgeführt werden. Daher musst Du danach:
- CRTPGM (create program, programm erstellen) verwenden, welches aus einem oder mehreren Modulen ein Programmobjekt erstellt.

Daneben gibt es noch so genannte Serviceprogramme. Diese bestehen aus einer Sammlung von Prozeduren in einem oder mehreren Modulen. Diese Serviceprogramme können von jedem ILE Programm verwendet werden. IBM und andere Firmen stellen eine Menge Serviceprogramme zur Verfügung, genauso kannst auch Du Serviceprogramme für andere schreiben. Zum Erstellen gibt es den Befehl CRTSRVPGM (create service program, erstelle Serviceprogramm).

Programm-Management und Aktivierungsgruppen

Mit ILE hat der Programmierer interessante neue Möglichkeiten, sein Programm zu steuern, Ressourcen zu verwalten und zu teilen, Fehlerbedingungen im Programm abzufangen und viele interessante APIs (Programmierschnittstellen) zu verwenden.

Gerade durch tiefgehende Änderungen im Betriebssystem OS/400 macht die Verwendung von ILE Sinn. Denn ILE-Programme und Serviceprogramme werden nun in so genannten „activation groups“ (Aktivierungsgruppen) zusammengefasst. Ein Programm wird nicht mehr einfach nur gestartet, es wird aktiviert. Eine Aktivierungsgruppe ist eine Sammlung von Ressourcen und Jobsteuer-Parametern (Commit-Steuerung, Dateiüberschreibungen, geminsame Nutzung von Datenpfaden), die für alle in dieser Gruppe laufenden Programme gültig ist. Durch die Vorbereitung dieser Jobs in einer Aktivierungsgruppe kann eine Menge Zeit und Ressourcen gespart werden, zum Beispiel bei häufiger Mehrfachverwendung von Programmen und Modulen.

Weiterhin kann ILE das Aufrufen von Programmen beschleunigen. Mit dem OPM wurde bei einem Aufruf eines anderen Programms über den Befehl `CALL` während der Laufzeit die Adresse für das gerufene Objekt ermittelt, das Objekt initialisiert und dann aufgerufen. Wenn man nun modulare Prozeduren verwendet, wird die Adresse während des Bindens der einzelnen Module ermittelt und in das Programm eingebaut und die Prozedur schneller aufgerufen.

Fehlersuche mit dem Source Debugger

Mit dem ILE Source debugger kannst Du ILE und OPM Programme (alle Sprachen) bearbeiten. Du kannst den Ablauf des Programms mit interessanten Befehlen steuern, per Tastendruck einzelne Befehle ausführen, sowie Variablen anzeigen und ändern. Im Falle eines Fehlers oder von Dir gesetzten Breakpoints siehst Du automatisch den Quellcode und kannst Befehle eingeben. Eine genauere Beschreibung des Debuggers folgt.

Danke fürs Lesen bisher!