Praxisorientierte Beispiele

AREXX MISCHT MIT

von Bob Malzan
und Rainer Zeitler

Daß ARexx mit ARexx-fähigen Programmen gut umgehen kann, hat sich inzwischen herumgesprochen. Weniger spektakulär – aber nicht weniger nützlich – sind kurze Programme, Einzeiler genannt. In diesem Beitrag zeigen wir einige Beispiele, mit denen sich bequem Routineaufgaben durch kleine ARexx-Routinen ohne großen Aufwand bewältigen lassen.

Die Stärke von ARexx ist vor allem die Möglichkeit, Zeichenketten (Strings) einfach zu verarbeiten. Mit ARexx lassen sich ohne großen Aufwand einfache, aber nützliche Programme erstellen. Sie benötigen nicht mehr als ARexx selbst und einen Editor, um eigene Programme zu erstellen.

ARexx verfügt über einen großen Befehlsschatz: logische, arithmetische und Vergleichsoperatoren, Arrays, Eingabe- und Ausgabefunktionen, Kommentare und Schleifen. Ebenso finden wir leistungsfähige String-Operationen, die in ARexx eine besonders wichtige Rolle spielen. Nach Sound- oder Grafikbefehlen jedoch, wie sie von Amiga-BASIC bekannt sind, sucht man bei ARexx vergeblich.

Möchte man ein ARexx-Programm ausführen, muß zunächst ARexx aktiv sein. Dieses geschieht durch Aufruf von »RexxMast« via CLI/Shell oder der Workbench. Ein ARexx-Programm ist eine Folge von im ASCII-Format vorliegenden Anweisungen. Das bedeutet aber, daß es ohne weitere Angaben nicht selbst lauffähig ist. Deshalb müssen wir jedes ARexx-Listing mit dem Zusatz »rx« versehen. Rx ist ein ARexx zugehöriges Programm, das ein Listing einliest und interpretiert, d.h. Zeile für Zeile übersetzt und ausführt. Diese Arbeitsweise kennen wir von Amiga-BASIC und kennzeichnet sog. Interpreter-Sprachen (engl. »to interpret« = übersetzen).

Doch ARexx wäre nicht ARexx, gäbe es nicht dennoch die Möglichkeit, Betriebssystemfunktionen zu nutzen, also Grafiken zu zeichnen, Fenster zu öffnen oder Schalter (Gadgets) zu erzeugen. Die Hintertür, die sich ARexx offengelassen hat, sind sog. RexxLibraries. Da gibt es z.B. die RexxSupport-Library, zuständig für die Datei- und Speicherverwaltung bzw. das Port-Handling, oder die RexxArplib-Library, u.a. Bindeglied zwischen ARexx und der Grafik- bzw. Intuition-Library. Im nächsten AMIGA-Magazin zeigen wir, wie solche Libraries selbst zu erstellen sind.

ARexx-Einzeiler wie die folgenden zeigen die Effektivität der Skript-Sprache. Ein Einzeiler, der uns zu einer vorgegebenen Zeit eine Nachricht mitteilt. Im übrigen sind die Einzeiler wirklich **Einzeiler,** auch wenn diese aus drucktechnischen Gründen nicht in eine Zeile passen.

```
run rx "'wait until 00:30';say
  'Schlafenszeit, ab ins Bett!'"
```

Eine halbe Stunde nach Mitternacht erinnert ARexx daran, daß Sie gerade im Begriff sind, die halbe Nacht vorm Bildschirm zu verbringen. Hier lernen wir das erste Mal die Verwendung von »rx« kennen. Rx ist es völlig gleichgültig, ob das Argument ein Dateiname oder eine Folge von Befehlen ist, eingeschlossen in »"«. Der erste Befehl lautet

```
'wait until 00:30'
```

Hiermit geben wir an, daß dieses Programm bis 0.30 Uhr warten soll, ehe es fortfährt. Der zweite Befehl, vom ersten durch ein Semikolon getrennt, heißt

```
say 'Schlafenszeit, ab ins Bett!'
```

Die hier verwendete ARexx-Funktion »say« ist mit der Print-Funktion von Amiga-BASIC vergleichbar. Ausgegeben wird der Text in die Standardausgabe. Die Say-Funktion ist im übrigen wesentlich komplexer und leistungsfähiger, als es bei diesem kleinen Programm den Anschein hat. ARexx erlaubt hier eine beliebige Verknüpfung von Text und Zahlen.

Ganz nützlich ist ebenfalls ein »Alias«, der alle z.Zt. geöffneten Libraries ausgibt:

```
alias lslibs rx "say showlist(L)"
```

Das AmigaDOS-Kommando »Alias« weist einer Kommandofolge einen Begriff (hier lslibs) zu. So läßt sich Tipparbeit sparen, reicht es doch aus, nur noch »lslibs« einzutippen, anstatt »rx "say showlist(L)"«.

Ein weiterer Einzeiler nutzt die Möglichkeit von ARexx, Zahlen unterschiedlicher Systeme zu konvertieren. In unserem Beispiel geben wir eine dezimale Zahl an und erhalten das Ergebnis in sedezimaler Notation. Verantwortlich hierfür ist die ARexx-Funktion d2x(). Die beiden eckigen Klammern im Funktionskörper sind für den Befehl Alias von Interesse. Hier wird das zweite Argument eingesetzt, das nach dem eigentlichen Alias-Name folgt:

```
alias xcalc rx "say d2x([])"
```

Probieren Sie doch mal »xcalc 255 x 1024«. ARexx rechnet den numerischen Ausdruck um und gibt das Ergebnis in sedezimaler Notation aus – Resultat: 3FC00.

Die elegante Lösung, ARexx-Programme in Alias-Definitionen zu packen, regt zu weiteren Ideen an. Folgender Einzeiler – »Banner« – gibt einen umrahmten Text im Bildschirm aus:

```
alias Banner rx
 "say Copies('*',70);
  say '*'Center('[]',68)'*';
  say Copies('*',70)"
```

Auch in diesem Beispiel wird die Möglichkeit genutzt, mehr als einen Befehl in einer Zeile auszuführen. Dabei trennen wir die Befehle mit Semikola. Und wie schon erwähnt: auch dieses kleine Programm ist ein Einzeiler.

Na, fallen Ihnen jetzt auch einige hilfreiche Einzeiler ein, die den Computeralltag von Nutzen sein können? Doch lesen Sie zunächst weiter. Wir stellen im Anschluß ein wirklich nützliches, zumindest aber aufschlußreiches Programm vor.

Banner ist in jedem Fall zu empfehlen, möchte man z.B. Listings einen Titel voranstellen. Folgendes Kommando wäre in diesem Fall anzugeben:

```
Banner Mein_ARexxPrg >PRT:
 type MeinARexxPrg.rexx >PRT:
```

Einige Drucker allerdings treffen hier Probleme auf, da die dazugehörigen Druckertreiber beim Schließen von PRT: einen automatischen Seitenvorschub erzeugen.

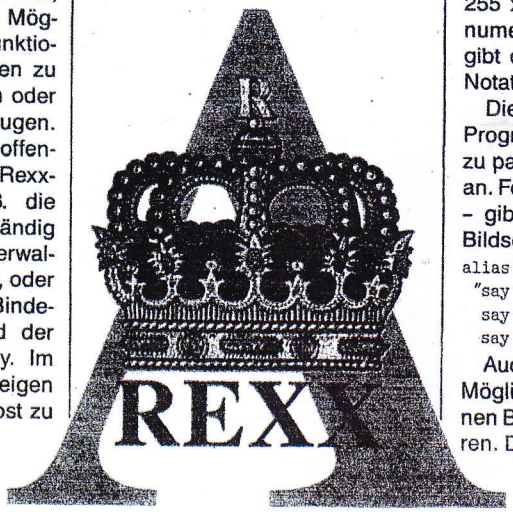
ARexx – der Tip für Tippfaule

Besser wäre also eine Lösung, den Text von Banner mit dem Listings mischt. Hier hilft uns »Pipe« weiter. Selten genutzt, doch halb aber nicht weniger interessant. Mit Hilfe der Pipe lassen sich beide Texte in einem Eingabestrom (engl. »Input Stream«) vereinen. Folgendes Kommando erledigt ebendies:

```
type MeinARexxprg.rexx |
 rx Banner_2 Mein_ARexxPrg >PRT:
```

»Banner_2« muß im Prinzip genauso beschaffen sein, wie der schon vorgestellte Banner-Alias. Allerdings muß Banner_2 nach dem Erzeugen des Titelrahmens den Text, der von der einen Seite die Pipe hineingereicht wird, der anderen Seite wieder herausreichen. Diese Prozedur ist für einen Einzeiler zu lang bzw. zu unübersichtlich. Deshalb ein ersvollständiges ARexx-Listing:

```
/*Banner_2.rexx - erzeugt einen
  *eingerahmten Titel und kopier
  *von der Standardeingabe in di
  *Standardausgabe */
parse arg Titel
say copies('*',70)
say '*'center("'"Titel"'",68)'*
say copies('*',70)
do while ~eof(stdin)
  writeln(stdout,readln(stdin))
end
```



Hier lernen wir ein wichtiges Merkmal von ARexx-Listings kennen: Jedes muß mit einem Kommentar beginnen. Dieser ist identisch mit denen von C-Programmen, also zunächst die Zeichenfolge »/*«, ein Text, anschließend »*/«. Es reicht, am Beginn eines ARexx-Programms folgende Zeile einzufügen:

```
/**/
```

Hiermit ist die Vorgabe eines ARexx-Kommentars erfüllt.

Banner_2 funktioniert schon ganz gut. Besitzer von PostScript-Druckern aber – und die Zahl derer nimmt zu – haben auch hiermit noch ihre Schwierigkeiten, existieren doch (noch?) keine Druckertreiber für PostScript. PostScript ist eine Seitenbeschreibungssprache. Texte und Grafiken sind zuvor in eine definierte Form zu bringen.

Selbst PostScript ist kein Problem

Bei Texten schickt man zunächst einen Vorspann an den PostScript-Drucker, den sog. Header. Anschließend folgt der Text, der zusätzlich noch entsprechend zu verpacken ist. Das Listing »PSList.rexx« erfüllt diese Bedingungen.

Der Vorspann, hier PS_BEGIN, hat eine genau festgelegte Aufgabe. Da wir nicht nur die Standardzeichen ausgeben möchten, sondern auch alle Umlaute, muß dies PostScript mitgeteilt werden. Zum Glück hat sich Commodore beim Amiga an die sog. ISO-Norm gehalten. Die beherrscht auch ein PostScript-Drucker, sofern man sie einstellt. ISOLatin1Encoding nennt man dies in PostScript. Der Vorspann codiert also einen PostScript-Font (Courier) nach ISO-Norm. Wir nennen ihn ISOCou.

Im Prinzip reicht dieser Vorspann aus. Doch bei PostScript handelt es sich um einen kleinen Computer, ein Druckjob also ist nichts anderes als ein Programm. Lassen wir ihn also mehr tun, außer einfach nur einen Text drucken. Im PostScript-Vorspann finden Sie deshalb die Routine SH. Diese gibt nicht nur eine Zeile Text aus, sondern berechnet zudem, an welcher Position sich die nächste Druckzeile befindet, um gegebenenfalls einen Seitenvorschub (showpage) auszulösen.

Doch auch ARexx muß noch etwas tun. PostScript verbietet in Texten bestimmte Steuerzeichen. Das sind zum einen die runden Klammern, sofern sie nicht innerhalb der Zeile auch wieder geschlossen werden, und der Backslash (»\«). Beide, die Klammer und der Backslash, sind durch einen vorangestellten Backslash einzuleiten (»\\« oder »\(«). Die Funktion ProtectLine tut also nichts weiter, als nach dem zu schützenden Zeichen zu suchen und vor diesem einen Backslash einzufügen. Damit ist für ARexx die Arbeit allerdings getan.

Gelesen wird von der Standardeingabe, geschrieben in die Standardausgabe. Hier zwei Beispiele:

```
rx pslist <MeinARexxprg.rexx
 Mein_ARexxPrg | more
```

und

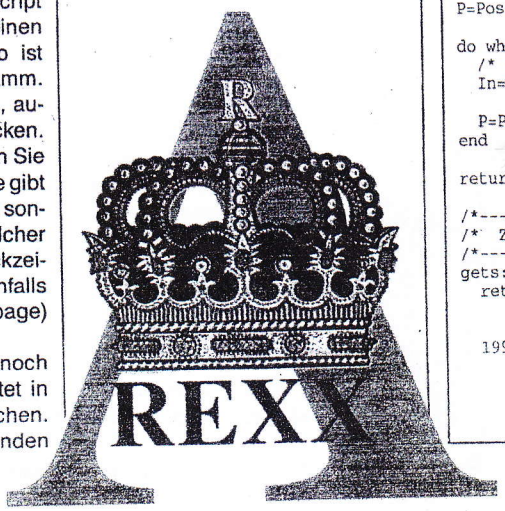
```
rx Banner_2 <MeinARexxprg.rexx
 Mein_ARexxPrg | rx pslist
 >Text.ps
```

In der ersten Version erzeugt pslist.rexx einen Listingkopf in doppelter Schriftgröße, der mit dem Programm »more« (geht nicht mit jedem more-Programm; es muß Pipe-fähig sein) in Ruhe durchgeblättert werden kann. Beim zweiten Aufruf mischt sich Banner_2.rexx ein, um einen durch Sternchen gerahmten Titel in den Ausgabestrom einzuschleusen. PsList macht daraus ein PostScript-Programm, das in die Datei »Text.ps« umgelenkt wird. Diese Datei kann mit einem Editor nachbearbeitet oder direkt an den Druckerport (z.B. SER:) gelenkt werden:

```
type Test.ps >SER:
```

Sind Sie auf den Geschmack gekommen? ARexx, richtig eingesetzt, nimmt uns viel Arbeit ab. Vielleicht fallen Ihnen weitere nützliche ARexx-Einzeiler ein. Schikken Sie diese an die Redaktion. Wir werden sie sammeln und in einer der nächsten Ausgaben veröffentlichen. *rz*

Literatur:
[1] Bob Malzan: ARexx-Programmierung auf dem Amiga, Markt & Technik Verlag AG, ISBN 3-87791-035-1, 59 Mark



Programm:	PsList.rexx
Programmiersprache:	ARexx
Aufruf:	s. Artikel

```
Programmautor: Bob Malzan
-------------------------

/*
 * Filtert Textdateien von stdin nach stdout
 */

parse arg Title

CR='0A'x

PS_BEGIN="/Courier findfont"CR,
         "dup length dict begin"CR,
         "  {1 index /FID ne {def} {pop pop} ifelse} forall"CR,
         "  /Encoding ISOLatin1Encoding def currentdict"CR,
         "end /ISOCou exch definefont pop"CR,
         "/LineNum 0 def"CR,
         "/SH { show"CR,
         "  /LineNum LineNum 1 add def"CR,
" currentpoint 50 lt {pop showpage 30 800 moveto /LineNum 0 def}"CR,
         "  {pop 30 800 LineNum 11 mul sub moveto} ifelse"CR,
         "} def"CR,
         "30 800 moveto"

if Title ~="" then PS_BEGIN=PS_BEGIN||CR,
         "/ISOCou findfont 20 scalefont setfont"CR,
         "("Title") SH"

PS_BEGIN=PS_BEGIN||CR,
         "/ISOCou findfont 10 scalefont setfont"

/*-------------------------------------------------------------*/
/*                    Drucke eine Datei                        */
/*-------------------------------------------------------------*/

/* PostScript-Vorspann */
call writeln(STDOUT,PS_BEGIN)

do while ~EOF(stdin)
    Line=gets()

    /* '\'-Zeichen schützen */
    Line=ProtectLine(Line,'\')

    /* '('-Zeichen schützen */
    Line=ProtectLine(Line,'(')

    /* ')'-Zeichen schützen */
    Line=ProtectLine(Line,')')

    call writeln(STDOUT,"("Line") SH")
end

/* letzte Seite drucken */
call writeln(STDOUT,"showpage")

/* EOF-Zeichen für PostScript */
call writech(STDOUT,'04'x)
exit

/*-------------------------------------------------------------*/
/* Verbotene Zeichen für PostScript durch '\' schützen         */
/*-------------------------------------------------------------*/
ProtectLine: PROCEDURE
parse arg In,Ch

/* Position des 'verbotenen' Zeichens */
P=Pos(Ch,In,1)

do while P>0
  /* schützen */
  In=Insert('\',In,P-1,1)

  P=Pos(Ch,In,P+2)
end

return In

/*-------------------------------------------------------------*/
/*  Zeile (ohne Carriage-Return am Ende) von STDIN holen       */
/*-------------------------------------------------------------*/
gets: PROCEDURE EXPOSE CR
  return strip(readln(STDIN),T,CR)

  1992 M&T
```

Listing Auch ohne Postscript-Druckertreiber lassen sich Texte auf Postscript-Druckern ausgeben