ARexx von der Pike auf lernen (Folge 2)

Die Krone aufgesetzt

Nach den Innereien von ARexx wenden wir uns im zweiten Teil der Außenwelt zu: Funktionen für Dateizugriffe, Textein- und -ausgabe sowie die Steuerung des Texteditors »GoldED«. Mit ARexx ist das leicht verdauliche Kost.

von Karsten Wysocki

Um sinnvolle Anwendungen zu entwickeln, ist es in jeder Programmiersprache nötig, Daten aus Dateien zu lesen und Benutzereingaben entgegenzunehmen, um sie anschließend zu verarbeiten. Auch ARexx beherrscht das und hält für diese Zwecke eine Reihe von Funktionen in Bibliotheken bereit.

```
/* Repeat ------------------------------------------------- */

/* Öffnen einer Ein/Ausgabekonsole. */
call open("Console","CON:0/0/400/100/Repeat/WAIT/CLOSE/SMART")

/* Eingabeaufforderung ausgeben.    */
call writeln("Console","Bitte geben Sie etwas ein (Ende = exit)")

input = ""                      /* Eingabevariable initialisieren */
call open("In","T:Input",W)     /* Datei zum Schreiben öffnen     */
do until input = "exit"         /* bis "exit" eingegeben wird     */
   call writech("Console","*>") /* Eingabeprompt ausgeben         */
   input = readln("Console")    /* Benutzereingabe einlesen       */
   call writeln("In",input)     /* Eingabe speichern              */
   call writeln("Console",">>>Gespeichert<<<")      /* melden!   */
end                             /* Zurück zum Schleifenanfang     */
call close("In")                /* Datei schließen                */

call writeln("Console","Sie haben eingegeben:") /* Melden         */

output = ""                     /* Ausgabevariable initialisieren */
call open("Out","T:Input",R)    /* Datei zum Lesen öffnen         */
do while ~eof("Out")            /* Solange Ende nicht erreicht    */
   output = readln("Out")       /* Zeile aus Datei einlesen       */
   call writeln("Console",output) /* Zeile ausgeben               */
end                             /* Zurück zum Schleifenanfang     */
call close("Out")               /* Datei schließen                */

/* Nicht mehr benötigte Datei löschen. */
address command "Delete >NIL: t:Input"

exit                            /* Programmende */
```

Listing 1: Grundsteine für komplexere Anwendungen. Damit sind beliebige Dateizugriffe machbar.

Funktionen bestehen immer aus ihrem Namen und einer Liste von Parametern in Klammern:

```
funktion(parameter,parameter,...)
```

Damit der ARexx-Interpreter den Funktionsaufruf eindeutig erkennt, muß die geöffnete Klammer direkt hinter dem Namen der Funktion stehen. Der Aufruf von Funktionen läßt sich in der »ARexxShell« des »ARexxMangers« ausprobieren.

```
say show(L)
```

Nach Drücken der Return-Taste erhalten Sie als Ausgabe alle zur Zeit geladenen ARexx-Funktionsbibliotheken. Die Funktion »show()« befindet sich in der »rexxsyslib.library«. Der Parameter »L« ist ein Kürzel für »Libraries«. »Say« sorgt dafür, daß der Rückgabewert von »show()« auf der Konsole (im Fenster) erscheint.

Funktionsrückgabewerte lassen sich jedoch in ARexx-Programmen auch in Variablen auffangen. Dazu gibt man statt der Anweisung »say« einfach »variable = « an. Probieren Sie es aus:

```
var = show(P); say var
```

Daraufhin erhalten Sie eine Liste aller vorhandenen ARexx-Portnamen, denn »P« ist das Kürzel für »Ports«. Wenn Sie in Ihrem ARexx-Handbuch unter der Funktion show() nachschauen, werden Sie noch einige weitere Kürzel finden, die Sie als Parameter einsetzen können.

Funktionsrückgabe-Parameter müssen aber nicht verwertet werden – entweder weil das Ergebnis nicht interessiert oder aber die Funktion sowieso keinen Wert zurückliefert. Dann sind sie per »call« aufzurufen:

```
call funktion(par1,par2,...)
```

Sesam öffne dich

Dieses Wissen reicht schon aus, ARexx-Funktionen zu benutzen, um etwa eine Datei zu öffnen. Geben Sie als eine Zeile in der ARexxShell ein:

```
call open("MyDat","T:Test",W);
say show(F)
```

Nach Drücken der Return-Taste erhalten Sie im Ausgabefenster die Zeile »STDOUT STDIN MyDat«. Das wurde durch die Anweisung »say show(F)« bewirkt. Sie gibt alle momentan von ARexx geöffneten Dateien aus.

Im Verzeichnis »T:« befindet sich nun eine leere Datei »Test« (und nicht »MyDat«). Warum die zwei Namen? Der erste Parameter ist der interne Name der Datei; Dieser kommt immer dann zum Einsatz, wenn eine ARexx-Funktion Daten in die Datei schreiben, aus ihr lesen oder Informationen über die Datei ermitteln soll. Er ist meist recht kurz, damit man nicht so viel zu tippen hat.

Der zweite Parameter ist der vollständige Dateiname inklusive Pfadangabe. Dieser wird nur beim Öffnen der Datei dem Betriebssystem übergeben und später nicht mehr benötigt.

Der dritte Parameter bestimmt die Zugriffsart: »W« steht für »Write« (Schreiben). Wird eine Datei im Schreib-Modus geöffnet, legt sie ARexx immer neu an, auch wenn es schon eine gleichen Namens gab. Die alte ist dann futsch. Als Zugriffsarten sind weiterhin noch »R« für »Read« (Lesen) und »A« für »Append« (Anhängen) zulässig.

Geöffnete Dateien schließt ARexx beim Verlassen eines Programms automatisch. Es gehört jedoch zum guten Programmierstil, geöffnete Dateien immer mit »close(Name)« zu schließen. Die Funktion close() kennt als einzigen

```
/* RepeatProc --------------------------------------------- */

kon = "Console"          /* Internen Namen der Konsole defineren */
                         /* Öffnen einer Ein/Ausgabekonsole */
call open(kon,"CON:0/0/400/100/RepeatProc/WAIT/CLOSE/SMART")
call writeln(kon,"Bitte geben Sie etwas ein (Ende = exit)")

call open("In","T:Input",W)       /* Datei zum Schreiben öffnen */
call storeinput()          /* Unterprogramm storeinput aufrufen */
call close("In")                  /* Eingabedatei schließen */

call writeln(kon,"Sie haben eingegeben:")   /* Meldung ausgeben */
filename = "Out"                 /* Internen Dateinamen festlegen */
call open(filename,"T:Input",R) /* Datei zum Lesen öffnen        */
anz = showinput(kon,filename)   /* Unterprogramm mit Parametern */
                                /* aufrufen. Ergebnis in »anz«  */
                                /* speichern. */
call close(filename)            /* Datei schließen               */
call writeln(kon,"Sie haben" anz "Zeilen eingegeben")

/* Nicht mehr benötigte Datei löschen. */
address command "Delete >NIL: t:Input"
exit                    /* Programmende */

/*
 Unterprogramm storeinput()

 Dieses Unterprogramm hat keinen eigenen Variablenbereich und
 benutzt die im Hauptprogramm definierten Variablen. Alle
 Änderungen sind daher auch nach der Rückkehr ins Hauptprogramm
 wirksam!
*/

storeinput:                     /* Unterprogrammnamen festlegen */

input = ""
do until input = "exit"         /* Bis "exit" erreicht ist      */
   call writech(kon,"*>")       /* Eingabeprompt ausgeben       */
   input = readln(kon)          /* Eingabe einlesen             */
```

Parameter den internen Namen einer geöffneten Datei und kann ohne weiteres mit der Anweisung »call« aufgerufen werden.

Ein paar oft gebrauchte Funktionen können Sie Listing 1 entnehmen. Starten Sie es einfach über den SkriptManager.

Sie erhalten dann auf der Workbench ein Fenster mit einer Eingabeaufforderung und einem Prompt. Wenn Sie nun einige Zeilen eingeben, schreibt das Programm diese in die Datei »T:Input« Nach Eingabe von »exit« schließt es die Datei und öffnet sie erneut zum Lesen. Anschließend liest es alle zuvor in die Datei geschriebenen Zeilen und gibt sie auf die Konsole aus. Zum Schluß löscht das Programm die Datei »T:Input«.

Wahrscheinlich ist Ihnen das Zeichen »~« vor dem Funktionsaufruf »eof()« aufgefallen. Dieses Zeichen bewirkt nichts weiter als eine Verneinung des dahinter stehenden Ausdrucks, also in diesem Fall »**Not** End of File« (**Nicht** Dateiende).

Vorratskeller

Bei der Entwicklung größerer Programme kommt es häufig vor, daß man bestimmte Befehlsfolgen immer wieder benötigt. Damit man diese im Programm nicht ständig wiederholen muß, gibt es die Unterprogrammtechnik. Dabei werden oft benötigte Anweisungen zusammengefaßt, bekommen einen Namen und lassen sich fortan einfach durch Angabe des Namens ausführen.

ARexx verfügt über zwei Arten von Unterprogrammen. Die eine ist die prozedurale Art. Sie heißt so, weil am Ende des Hauptprogramms Prozeduren definiert werden. Sie lassen sich so angeben, daß sie eigene Variablen besitzen und somit die Variablen des Hauptprogramms nicht beeinflussen oder auch so, daß sie zusammen mit dem Hauptprogramm denselben Satz von Variablen verwenden. Dies wird durch das Schlüsselwort »PROCEDURE« bestimmt.

Die Definition einer Prozedur beginnt mit dem Namen und dem darauf folgenden Doppelpunkt. Sie können vom Hauptprogramm oder einer anderen Prozedur wie Funktionen aufgerufen werden. Das zweite Listing »RepeatProc« zeigt, wie es geht. Das Programm »RepeatProc« ähnelt dem ersten Listing, gibt am Ende jedoch noch die Anzahl der eingegebenen Zeilen aus.

Die andere Möglichkeit, unter ARexx Unterprogramme zu definieren, ist recht trickreich: Dabei füllt man zuerst Programmanweisungen in eine Variable und läßt sie dann mit dem Befehl »Interpret« ausführen (wie in Listing 3 zu sehen). Wenn Sie sich die Definition der Variablen »storeinput« ansehen, werden Sie feststellen, daß die Definition über mehrere Zeilen geht. Auch die Kommentare werden hier in die Variable eingefüllt.

Wenn Sie das Listing abtippen und die Kommentare weglassen wollen, müssen Sie die Semikola, die sich jeweils vor den Kommentaren befinden, stehen lassen. Sie benötigt der ARexx-Interpreter, um zu erkennen, daß die Definition des Variableninhalts noch weitergeht.

Abgeschlossen ist die Definition nach dem zweiten Anführungszeichen oben; deswegen sind in dieser Definition auch alle anderen Anführungszeichen oben durch Hochkommata ersetzt worden. Das Hochkomma dient ARexx wie die Anführungszeichen als Zeichenketten-Begrenzung. Der Interpreter behandelt sie aber wie verschiedene Zeichen.

Ferngesteuert

Das Steuern von Anwendungsprogrammen mit ARexx funktioniert im Prinzip genauso wie mit dem im ersten Workshop-Teil vorgestellten Programm »TestMsg«. Das Anwendungsprogramm stellt eine ARexx-Schnittstelle zur Verfügung, an die man Befehle und Daten senden kann. Im Prinzip kann man an diese Schnittstelle alles Mögliche senden, die Programme reagieren jedoch nur sinnvoll auf Befehle, auf die sie vorbereitet sind. Anwendungsprogramme verarbeiten eben nur Befehle, die sie verstehen.

Der implementierte Befehlssatz ist je nach Anwendung recht unterschiedlich. Auch die Befehls-Syntax und die Übergabe von Daten können verschieden sein.

In diesem Kurs wird als zu steuerndes Anwendungsprogramm der Texteditor »GoldED« verwendet. Sie können ihn auf den PD-Disketten der Ausgabe 7/95 finden. Wenn Sie beabsichtigen, Anwendungsprogrammme mit ARexx zu steuern, sollten Sie in den jeweiligen Handbüchern nachlesen, auf welche Befehle und Daten Ihr Anwendungsprogramm reagiert.

Zuerst gilt es, den ARexx-Port von GoldED auszuprobieren. Starten Sie dazu den Editor und den ARexxManager. Aktivieren Sie das Gadget mit der Beschriftung »Ports« im ResourceManager. Daraufhin erhalten Sie eine Liste aller zur Zeit existierenden ARexx-Ports. In dieser Liste sollte sich auch ein Eintrag »GOLDED.1« befinden. Ist dies der Fall, wechseln Sie in die ARexxShell und geben Sie folgendes ein:

```
address "GOLDED.1" xxx
```

Nach Drücken der Return-Taste schaltet sich GoldED in den Vordergrund und präsentiert ei-

ARexx-Datei-Funktionen

Um dem Programmierer die Möglichkeit zu geben, mit Daten aus Dateien zu arbeiten, besitzt ARexx zahlreiche Dateizugriffsfunktionen.

Funktion	Beschreibung
open(name,dateiname,modus)	Datei öffnen
close(name)	Datei schließen
writeln(name,zeichen)	Write Line = Zeile schreiben
readln(name)	Read Line = Zeile lesen
writech(name,zeichen)	Write Char = Zeichen schreiben
readch(name,länge)	Read Char = Zeichen lesen
exists(name)	prüfen, ob Datei existiert
eof(name)	prüfen, ob Dateiende erreicht
seek(name,offset,pos.)	Dateizeiger positionieren
statef(dateiname)	Überprüfen des Dateistatus

```
   call writeln("In",input)    /* Eingabe speichern            */
   call writeln(kon,">>>Gespeichert<<<")          /* melden! */
   end                         /* Zurück zum Schleifenanfang   */

return                         /* Rückkehr zum Hauptprogramm   */

/*
Unterprogramm showinput(fenster,datei)

Dieses Unterprogramm ist mit dem Schlüsselwort PROCEDURE versehen
und benutzt einen eigenen Variablenbereich, verändert also nicht
die im Hauptprogramm benutzten Variablen, selbt wenn sie gleiche
Namen haben würden. Die zur Verarbeitung vom Unterprogramm
benötigten Werte werden als Parameter beim Aufruf übergeben. Im
Unterprogramm werden sie mit der Anweisung »parse arg«
übernommen. Dabei spielt die Namensgebung keine Rolle, lediglich
die Reihenfolge der Parameter ist entscheidend. Das Tennzeichen
zwischen den Parametern bestimmt die Aufrufsystax. Hier wurde ein
Komma ohne zwischenstehende Leerzeichen vereinbart, so daß beim
Aufruf die Parameter auch mit einem Komma ohne zusätzliche
Leerzeichen getrennt werden müssen.
*/

showinput: PROCEDURE
parse arg fenster,datei

x = 0                          /* Zähler initialisieren        */
do while ~eof(datei)           /* Solange nicht Dateiende      */
   output = readln(datei)      /* Zeile aus Datei lesen        */
   call writeln(fenster,output) /* Zeile auf Konsole ausgeben  */
   x = x + 1                   /* Zähler erhöhen               */
end                            /* Zurück zum Schleifenanfang   */
x = x - 1                      /* Zähler verringern, da eof    */
                               /* mitgezählt wurde             */
return x                       /* Rückkehr zum Programm mit    */
                               /* Rückgabe eines Ergebnisses   */
```

Listing 2: So lassen sich mehrfach wiederkehrende Programmteile in eigene Funktionen umwandeln

Kursübersicht

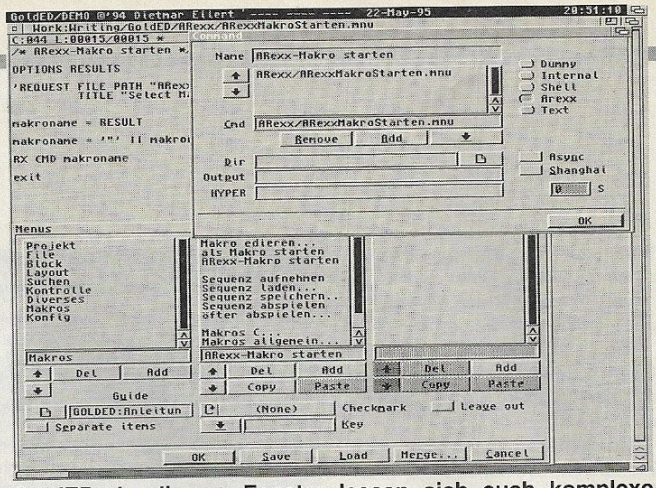
ARexx in Perfektion, für Leute, die sich bisher nie aufraffen konnten, diese einfache aber mächtige Sprache zu lernen, die zu jedem Amiga gehört: Das ist das Ziel dieses Kurses.

Folge 1: Einführung in ARexx, Variablen-Handling, Schleifen, Abfragen, Blöcke, Nachrichten; »TestMsg« verschickt Nachrichten.

Folge 2: Funktionsaufrufe, Dateizugriffe und die Konsole, Unterprogramme, Steuern anderer Programme, Anbindung von GoldED mit Skripten.

Folge 3: Die »apig.library«: grafische Oberflächen mit Schaltern, Fenstern und Requestern; eine Oberfläche für »LhA«.

Folge 4: Tabellenverarbeitung, ein Listview-Gadgets, Fenster-Refresh, TAC- (TransAktionsCode-)Programmiertechnik, weitere ARexx-Funktions-Bibliotheken.



GoldED: In diesem Fenster lassen sich auch komplexe ARexx-Skripts einfach in die Menüs des Editors einbauen

nen Requester mit der Meldung »Unknown cmd/Skript ?!«.

Das liegt daran, daß Sie GoldED etwas gesendet haben, was er nicht versteht: die Anweisung »xxx« ist kein gültiger Befehl. Schließen Sie das Fenster mit der Fehlermeldung. Jetzt wollen wir mal einen Befehl senden, den GoldED auch versteht:

```
address "GOLDED.1";
'REQUEST BODY "Bitte OK drücken"'
```

Achten Sie dabei auf die »'«, sie sind auf der deutschen Tastatur mit der Tastenkombination <Alt ä> zu erreichen (Commodore sei Dank!).

Daraufhin schiebt sich GoldED in den Vordergrund und präsentiert Ihnen einen Requester mit »Bitte OK drücken«. Drücken Sie auf das OK-Gadget und bringen Sie den Workbench-Bildschirm nach vorne. Im Ausgabefenster der ARexxShell sehen Sie, daß GoldED nach dem Senden des ersten Befehls die Fehlernummer 5 (RC:5) als Return-Code zurückgeliefert hat, da die Nachricht keinen gültigen Befehl enthielt. Den zweiten Befehl hat der Editor verstanden, bearbeitet und 0 als Fehler zurückgeliefert – das ist gleichbedeutend mit »kein Fehler«.

Vorsicht Falle

Vielleicht ist Ihnen aufgefallen, daß beim Senden des zweiten Befehls »address« ohne Befehl angegeben wurde, dieser folgt in der nächsten Zeile. Dies funktioniert, weil ARexx die zuletzt eingestellte Adresse solange einsetzt, bis eine andere mit »address« angegeben wird. Der ARexx-Interpreter sendet außerdem alles, was er nicht selbst verarbeiten kann, an diese Adresse. Beim Start eines ARexx-Programms ist die Adresse »REXX« eingestellt, so daß alles an den Interpreter geht.

Beim Testen von ARexx-Programmen muß man daher verstärkt auf Tippfehler achten, da nicht jeder Tippfehler eine sinnvolle Fehlermeldung erzeugt. Denn jede Anweisung, Funktion oder Variable, die der Interpreter nicht erkennt, schickt er an das voreingestellte Anwendungsprogramm. Außerdem geben nicht alle steuerbaren Anwendungsprogramme Fehlermeldungen aus, was eine Fehlersuche erschwert.

```
/* RepeatInt ------------------------------------------------ */
call open("Console","CON:0/0/400/100/RepeatInt/WAIT/CLOSE/SMART")
call writeln("Console","Bitte geben Sie etwas ein (Ende = exit)")

input = ""                    /* Eingabevariable initialisieren */
storeinput = "nop;           /* Variable mit Programmcode füllen */;
   do until input = 'exit';
      call writech('Console','*>');      /* sonst wie gehabt   */;
      input = readln('Console');         /* aber mit ' statt " */;
      call writeln('In',input);
      call writeln('Console','>>>Gespeichert<<<');
      end;
   nop"                       /* Ende der Variablendefinition  */

call open("In","T:Input",W)   /* Datei zum Schreiben öffnen    */
interpret storeinput          /* Definierte Variable ausführen */
call close("In")              /* Eingabedatei schließen        */

/* Rest wie in Listing 1 oder 2 für die Ausgabe */
```

Listing 3: So geht's auch: Oft benötigte Befehlsfolgen einfach per »interpret« zusammenfassen

Abhilfe schafft bei solchen Problemen das im ersten Workshop-Teil erarbeitete Programm »TestMsg«. Damit lassen sich solche Fehler leicht finden; statt der Port-Adresse des Anwendungsprogramms muß man nur den Port »TestMsg« angeben. Anschließend landet alles, was ARexx nicht selbst verdauen kann, bei »TestMsg«, wobei dann auch die durch Tippfehler verursachten, ungewollten Nachrichten auftauchen.

Tiefgang

Um einen tieferen Einblick in die Arbeitsweise des GoldED-ARexx-Ports zu bekommen, gilt es, einen etwas komplexeren Requester zu bauen, der ein verarbeitbares Ergebnis zurückliefert. Dies steht anschließend in der ARexx-System-Variablen »RESULT«. Damit Ergebnisse von Nachrichten auch in dieser Variablen gespeichert werden, muß im ARexx-Programm die Anweisung »OPTIONS RESULTS« stehen. In der ARexxShell können wir uns das jedoch sparen, denn Befehlen, die die ARexxShell versendet, fügt sie diese Anweisung immer hinzu.

Nun aber zum »komplexen« Requester:

```
'REQUEST FILE TITLE "My File Request"'
```

Daraufhin präsentiert GoldED einen Datei-Requester mit dem von Ihnen angegebenen Titel. Wählen Sie eine Datei aus und klicken Sie auf »OK«. Als Folge dieser Aktion sehen Sie wieder die Workbench und dort im Ausgabefenster der ARexxShell eine Erfolgsmeldung: in »RESULT« steht der komplette Name der ausgewählten Datei inklusive Pfad. In einem ARexx-Programm können Sie den Dateinamen in eine Variable übertragen und weiterverarbeiten.

Bevor wir als nächstes ein kleines Steuerprogramm für GoldED schreiben, sollten Sie noch ein wenig mit dem Datei-Requester experimentieren und herausfinden, was er für ein Return-Code (RC) und RESULT zurückliefert, wenn Sie im Requester nicht auf OK drücken oder gar keine Datei auswählen.

Starthilfe

Als erstes brauchen wir ein ARexx-Skript, das einen Dateirequester öffnet, darin alle GoldED-ARexx-Skripts anzeigt und anschließend das ausgewählte Skript ausführt. Dieses Skript wird in GoldED als Menüpunkt integriert.

Die direkte Ausführung von ARexx-Makros über GoldED bringt – durch eine Besonderheit dieses Texteditors – einen Vorteil mit sich: GoldED stellt bei einem gestarteten ARexx-Makro die Host-Adresse automatisch auf sich ein. Dadurch fällt das Angeben einer Port-Adresse im ARexx-Programm weg.

Das kurze Listing 4 erledigt die gestellte Aufgabe. Tippen Sie es ab und speichern Sie es unter »ARexxMakroStarten.mnu« im Verzeichnis »GoldED:ARexx/«.

Nach dem Abtippen und Speichern des ARexx-Skripts wählen Sie bitte aus dem Menü »Konfig« den Menüpunkt »Menus« aus. Daraufhin erscheint ein Fenster mit drei Listen. In der linken sind die Namen der Menüs zu sehen. Wählen Sie aus dieser Liste »Makros« aus. Danach erscheinen in der mittleren Liste alle Punkte dieses Menüs.

Drücken Sie nun mit der Maus auf das Gadget mit der Aufschrift »Add« unter der Liste. Im Eingabefeld erscheint nun ein Fragezeichen, das Sie am besten durch

```
ARexx-Makro starten
```

ersetzen. Nach dem Drücken von <Return> erscheint der eingegebene Text in der Liste. Damit ist der Menüpunkt definiert, es fehlt aber noch die Verbindung zum zuvor abgetippten ARexx-Skript. Klicken Sie dafür doppelt auf den neuen Menüpunkt; daraufhin erscheint ein neues Fenster, in dem wiederum über das Gadget »Add« eine neue Zeile der Liste hinzuzufügen ist. Diese Zeile sollte

```
ARexx/ARexxMakroStarten.mnu
```

```
/* ARexx-Makro starten ------------------------------------ */

OPTIONS RESULTS           /* Rückgabewerte in RESULT speichern */

/* GoldED-Filerequester mit Pfad, Pattern und Titel aufrufen */
'REQUEST FILE PATH "ARexx/" MASK "#?.ged"
         TITLE "Select Makro to start:"'

makroname = RESULT     /* Rückgabewert in Variable übertragen */
makroname = '"' || makroname || '"' /* Name mit »"« klammern */

RX CMD makroname  /* ARexx-Makro von GoldED ausführen lassen */

exit
```

Listing 4: Arbeiter im Untergrund: ARexx-Makro als Steuerung für neuen GoldED-Menüpunkt

lauten. Nach Drücken von <Return> und aktivieren von »ARexx« rechts der Liste, ist die Arbeit getan. Klicken Sie zweimal auf »OK«, um das Fenster zu schließen und den neuen Menüpunkt auszuprobieren.

Nach der Anwahl erhalten Sie einen Datei-Requester, der alle GoldED-ARexx-Makros mit der Endung ».ged« des Verzeichnisses »GoldED:ARexx/« zeigt. Das von Ihnen eingetippte Makro ist unsichtbar, denn es hat die Endung ».mnu«. Wenn Sie z.B. das Makro mit dem Namen »number.ged« wählen, numeriert GoldED den Text im aktiven Fenster zeilenweise durch.

```
/* ARexx-Stripper  © Karten Wysocki 1995 ---------------------- */

OPTIONS RESULTS           /* Rückgabewerte in RESULT speichern */
'REQUEST FILE PATH "ARexx/" MASK "#?" TITLE "Makro to Strip:"'

filename = RESULT         /* Resultat in Variable übertragen   */
'OPEN NAME "' || filename || '"'        /* Textfenster öffnen  */
'LOCK CURRENT'            /* Textfenster für Benutzer sperren  */

/* Konstanten für Zeichen-Vergleiche festlegen: */
leer  = hash(" ")         /* Die ARexx-Funktion hash()         */
slash = hash("/")         /* liefert den ASCII-Code            */
stern = hash("*")         /* eines Zeichens zurück             */

'QUERY LINES VAR=LINES'   /* Anzahl Zeilen Im Text erfragen    */

/* Äußere Schleife zum Abarbeiten aller Textzeilen einleiten   */
 do count = 1 to lines
  'QUERY ANYCHAR'              /* GoldED fragen ob Zeile leer.  */
  if RESULT=TRUE               /* Wenn Zeile nicht leer         */
   then do                     /* dann Beginne Funktionblock.   */
    char = ""                  /* Zeichenvariable initialisieren */
    exit = 0                   /* Abbruchvariable auf unwahr.   */
    del  = 0                   /* Delete-Merker setzen.         */
    do forever                 /* Innere Schleife einleiten.    */
     'QUERY CODE VAR=CHAR'     /* Zeichen unter Cursor lesen.   */
     call companddel()         /* Unterprogramm Aufrufen.       */
     if exit = 1 then leave    /* Wenn exit gesetzt raus        */
    end                        /* Zum Anfang der inneren Schl.  */
    if del = 0 then 'DOWN'     /* Wenn nicht gelöscht, Zeile run.*/
   end                         /* Ende des Funktionsblocks.     */
   else do                     /* sonst Beginne Funktionsblock. */
    'DELETE LINE'              /* Lösche Zeile.                 */
    count = count + 1          /* Inkrementiere Zähler.         */
   end                         /* Ende des Funktionsblocks.     */
  'FIRST'                      /* Cursor zum Zeilenanfang.      */
 end                           /* Zum Anfang der äußeren Schleife*/

/* Dateinamen ohne Pfad holen und in Variable speichern:       */
'QUERY FILE VAR=FILENAME'

'GOTO TOP'                /* Cursor auf erste Textzeile setzen  */
'INSERT LINE'             /* Zeile vor erster Textzeile einfügen */
'GOTO TOP'                /* Cusor auf eingefügte Zeile setzen  */

/* Kommentar in eingefügte erste Textzeile schreiben: */
'TEXT STAY T="/** Stripped' filename '**/"'

'UNLOCK'                  /* Textfenster für Benutzer freigeben */
exit                      /* Programm beenden                   */

/*
 Unterprogramm companddel() = (Compare and Delete)

 Dieses Unterprogramm benutzt die Variablen des Hauptprogramms
 und liefert daher nichts zurück. Im Unterprogramm werden die im
 Hauptprogramm von GoldED eingelesenen Zeichen mit den zu
 löschenden Zeichen verglichen und evtl. gelöscht.
*/

companddel:
 select                       /* Mehrfachauswahl einleiten      */
  when char = leer            /* Wenn Leerzeichen               */
   then 'DEL'                 /*  dann Löschen.                 */
  when char = slash           /* Wenn Schrägstrich              */
   then do                    /*  dann Funktionsblock           */
    'RIGHT'                   /* Cursor nach rechts             */
    'QUERY CODE VAR=CHAR'     /* Nächstes Zeichen lesen         */
    if char = stern           /* Wenn Sternchen gelesen         */
     then do                  /* dann Funktionsblock            */
      'DELETE LINE'           /* Lösche Zeile                   */
      del = 1                 /* Löschmerker setzen             */
     end                      /* Ende des Funktionsblocks       */
   end                        /* Ende des Funktionsblocks       */
  otherwise exit = 1          /* Andernfalls beende Schleife    */
 end                          /* Ende der Mehrfachauswahl       */
return                        /* Zurück zum Hauptprgramm.       */
```

Listing 5: Der ARexx-Stripper kürzt komplexe ARexx-Programme direkt auf dem Bildschirm zusammen

Striptease

Mit dem vorhergehenden Schritt haben wir uns nun die Möglichkeit geschaffen, beliebige ARexx-Makros mit GoldED zu starten, ohne dafür jeweils einen neuen Menüpunkt zu definieren. Im folgenden Schritt erstellen wir ein ARexx-Makro, das ARexx-Skripts auf ein Minimum komprimiert. Das Makro löscht führende Leerzeichen, die nur der Programmierer zur besseren Übersichtlichkeit benötigt, sowie ganzzeilige Kommentare aus ARexx-Skripts. Das nennt man »Strippen«, daher heißt dieses Makro »ARexx-Stripper«. Der Clou bei diesem Makro ist, daß man die Aktion direkt am Bildschirm verfolgen kann.

Dieses Strippen von ARexx-Programmen hat den Vorteil, daß das Laden und Prüfen vor der Abarbeitung weniger Zeit kostet und ein wenig Schutz vor Manipulation bietet, wenn man es an andere weitergibt. Eigene ARexx-Programme sollte man für spätere Weiterentwicklung allerdings immer im übersichtlich formatierten und mit Anmerkungen versehenen Original-Listing aufheben.

Wenn Sie den »ARexx-Stripper« (Listing 5) abtippen, sollten Sie darauf achten, daß Sie keins der Hochkommata vergessen, mit denen die an GoldED zu sendende Befehle eingeschlossen sind.

Nach dem Abtippen speichern Sie das Listing unter »ARexx-Stripper.ged« im ARexx-Verzeichnis von GoldED. Wenn Sie anschließend den Menüpunkt »ARexx-Makro starten« und dann »ARexxStripper.ged« auswählen, legt GoldED vom Skript gesteuert los: Sie erhalten einen weiteren Datei-Requester, der Sie auffordert, ein ARexx-Skript auszuwählen.

Dieses Skript wird geladen und gekürzt. Diesen Vorgang können Sie am Bildschirm mitverfolgen. Während des Durchlaufs, ist das Textfenster für Benutzereingaben mit dem Befehl »LOCK CURRENT« gesperrt. Falls beim Ablauf Ihres ARexx-Strippers – beispielsweise durch einen Tippfehler – etwas schiefläuft, wird der ARexx-Stripper abgebrochen und der Editor bleibt erst einmal blockiert. Diese Sperre können Sie durch Eingabe von

```
ADDRESS "GOLDED.1" "UNLOCK"
```

in der ARexxShell des ARexxManagers aufheben.

Nach einem erfolgreichen Lauf des ARexx-Strippers wird das Skript nicht automatisch gespeichert, damit Sie noch Änderungen anbringen können. Wenn Sie das gestrippte Programm für einen Testlauf starten möchten, müssen Sie es vorher mit der GoldED-Funktion »Speichern als...« sichern. Dabei empfiehlt es sich, dem gestrippten Programm einen anderen Namen zu geben als dem formatierten und kommentierten Original. Das besser lesbare Original sollte man für spätere Änderungen aufheben.

Versuchen Sie einmal, den ARexx-Stripper an sich selbst auszuprobieren, sie werden feststellen, daß alle Kommentare stehenbleiben, die sich am Ende einer Anweisungszeile befinden. Auch Kommentare, die über mehrere Programmzeilen gehen, kann er nicht erkennen. Das zu beheben ist schwerer, als es auf den ersten Blick aussieht. Wenn Sie möchten, können Sie den ARexx-Stripper bei Bedarf leicht selbst erweitern. Die Erweiterungen sollten im »select«-Block des Unterprogramms »companddel()« eingebaut werden. *dg*

ARexx-Funktionsbibliotheken

ARexx-Funktionsbibliotheken enthalten Zusatzfunktionen. Diese sind bereits kompiliert und können daher sehr schnell abgearbeitet werden, was der Geschwindigkeit zugute kommt.

Von diesen Bibliotheken sind schon zwei im Betriebssystem des Amigas vorhanden: Die »rexxsyslib.library« und die »rexxsupport.library«. Sie befinden sich im Verzeichnis »LIBS:« des Betriebssystems.

Die »rexxsyslib.library« wird beim Start des ARexx-Systems von RexxMast automatisch geladen und die darin enthaltenen Funktionen können von jedem ARexx-Programm benutzt werden. Die »rexxsupport.library« enthält weitere Funktionen, vor allem zur Speicherverwaltung. Sie muß vor der ersten Benutzung mit »addlib()« geladen werden.

Literatur:
[1] Amiga-OS 3.1 ARexx, Commodore Electronics Limited. Erhältlich bei Village Tronic Marketing GmbH, Sarstedt
[2] Michael Metz: ARexx, Eine Einführung und mehr ..., CompuStore Handelsgesellschaft, ISBN 3-93073300-5