

**ARexxBox**

COLLABORATORS

	TITLE : ARexxBox		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		July 23, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>ARexxBox</b>	<b>1</b>
1.1	ARexxBox.guide	1
1.2	ARexxBox.guide/Copyright	2
1.3	ARexxBox.guide/Wichtig	2
1.4	ARexxBox.guide/Adresse des Autors	3
1.5	ARexxBox.guide/Einleitung	4
1.6	ARexxBox.guide/Systemanforderungen	5
1.7	ARexxBox.guide/Installation	6
1.8	ARexxBox.guide/Bedienung	7
1.9	ARexxBox.guide/Eingabe	7
1.10	ARexxBox.guide/MsgPort Basename	8
1.11	ARexxBox.guide/CommandShell	8
1.12	ARexxBox.guide/Merge in	9
1.13	ARexxBox.guide/Print	9
1.14	ARexxBox.guide/Generate Source	9
1.15	ARexxBox.guide/Clipboard	10
1.16	ARexxBox.guide/Konzept	11
1.17	ARexxBox.guide/Initialisierung	11
1.18	ARexxBox.guide/Kommandos von ARexx	11
1.19	ARexxBox.guide/Expandierung von Befehlen	12
1.20	ARexxBox.guide/Kommandos an ARexx	13
1.21	ARexxBox.guide/CloseDown	13
1.22	ARexxBox.guide/Dateien	14
1.23	ARexxBox.guide/Interfacefunktionen	15
1.24	ARexxBox.guide/Argumente&Resultate	18
1.25	ARexxBox.guide/Fehler	21
1.26	ARexxBox.guide/Einbindung	22
1.27	ARexxBox.guide/Oberon-2	24
1.28	ARexxBox.guide/ReadArgs	26
1.29	ARexxBox.guide/Library	28

1.30	ARexxBox.guide/ARexxBox-ARexxDispatch . . . . .	29
1.31	ARexxBox.guide/ARexxBox-CloseDownARexxHost . . . . .	30
1.32	ARexxBox.guide/ARexxBox-CommandShell . . . . .	30
1.33	ARexxBox.guide/ARexxBox-CommandToRexx . . . . .	31
1.34	ARexxBox.guide/ARexxBox-CreateRexxCommand . . . . .	32
1.35	ARexxBox.guide/ARexxBox-DoShellCommand . . . . .	33
1.36	ARexxBox.guide/ARexxBox-ExpandRXCommand . . . . .	34
1.37	ARexxBox.guide/ARexxBox-FindRXCommand . . . . .	34
1.38	ARexxBox.guide/ARexxBox-FreeRexxCommand . . . . .	35
1.39	ARexxBox.guide/ARexxBox-ReplyRexxCommand . . . . .	36
1.40	ARexxBox.guide/ARexxBox-SendRexxCommand . . . . .	37
1.41	ARexxBox.guide/ARexxBox-SetupARexxHost . . . . .	38
1.42	ARexxBox.guide/ARexxBox-StrDup . . . . .	38
1.43	ARexxBox.guide/Dateifformat . . . . .	39
1.44	ARexxBox.guide/Geschichte . . . . .	40
1.45	ARexxBox.guide/Danksagungen . . . . .	47
1.46	ARexxBox.guide/Index . . . . .	47

# Chapter 1

## ARexxBox

### 1.1 ARexxBox.guide

ARexxBox 1.12 Dokumentation

\*\*\*\*\*

Rechtliches:

Copyright	Copyright und andere rechtliche Dinge
Wichtig	Wichtige Bemerkungen
Adresse des Autors	Wohin man Bug reports, Kommentare & Spenden schickt

Die ARexxBox (kurz ARB):

Einleitung	Warum eigentlich ARexxBox?
Systemanforderungen	68040, 16 MB RAM, ... ;-)
Installation	Was kommt wohin?
Bedienung	Was passiert, wenn ich diesen Knopf drücke?

Der erzeugte Code:

Konzept	Wie das Ganze im Endeffekt funktioniert
Dateien	Welche Datei enthält was?
Interfacefunktionen	Schnittstelle zwischen ARexx und dem Programm
Argumente&Resultate	Parameter und Rückgabewerte
Fehler	Erweiterte Fehlerrückgabe
Einbindung	Richtlinien, Schritte
Oberon-2	Hinweise zum Oberon-2--Source

Kapitel zum Nachschlagen:

ReadArgs	Welcher Switch bedeutet welchen Typ
Library	Die Bibliotheksfunktionen der ARB

Anhänge:

Dateiformat	Aufbau der .arb--Files
Geschichte	Die Entwicklung der ARexxBox
Danksagungen	Der Autor möchte danken...
Index	Index für dieses Dokument

## 1.2 ARexxBox.guide/Copyright

Copyright und andere rechtliche Dinge

\*\*\*\*\*

Copyright (C) 1992,1993 Michael Balzer

Oberon-2 source copyright (C) 1993 hartmut Goebel

Diese Dokumentation darf kopiert und weitergegeben werden solange die Copyright-Notiz und diese Erlaubnis unverändert auf allen Kopien enthalten ist.

Es wird keine Garantie gegeben, daß die Programme, die in dieser Dokumentation beschrieben werden, 100%ig zuverlässig sind. Sie benutzen diese Programme auf eigene Gefahr. Die Autoren können auf \*keinen\* Fall für irgendwelche Schäden verantwortlich gemacht werden, die durch die Anwendung dieser Programme entstehen.

Das Paket ist "freely distributable", aber das Copyright liegt weiterhin bei Michael Balzer. Dies bedeutet, daß es von jedem kopiert werden darf solange er nicht mehr als eine angemessene Kopiergebühr dafür verlangt. Diese Gebühr \*darf nicht\* höher sein als US \$5 oder 5 DM.

\*Dieses Limit gilt auch für deutsche Public-Domain Händler!!\*

Dieses Paket darf in Public-Domain Sammlungen aufgenommen werden, insbesondere in Fred Fishs Amiga Disk Library (CD ROM Versionen dieser Sammlung eingeschlossen). Die Distributionsdatei darf in Mailboxsystemen oder auf FTP Servern abgelegt werden. Wenn Sie dieses Paket weitergeben wollen, dann \*müssen\* Sie die originale Distributionsdatei 'ARexxBox111.lha' benutzen.

Dieses Paket darf nach Absprache mit dem Autor auch in eine Sammlung von Entwicklerwerkzeugen (wie z.B. das Native Developer's Upgrade Kit von Commodore) aufgenommen werden.

Die Programme und der Quelltext (oder Teile davon) dürfen \*auf keinen Fall\* auf irgendeiner Maschine benutzt werden, die für die Forschung, Entwicklung, Konstruktion, Tests oder Produktion von Waffen oder anderen militärischen Gütern benutzt wird. Dies gilt natürlich auch für alle Maschinen, die für das Training von Personen in \*irgendeiner\* der obengenannten Tätigkeiten benutzt werden.

## 1.3 ARexxBox.guide/Wichtig

Wichtige Bemerkungen

\*\*\*\*\*

---

Eines vorweg: Die ARexxBox ist ein Werkzeug von Programmierern für Programmierer. Wenn Sie Anwender sind brauchen Sie gar nicht weiter zu lesen, die ARexxBox ist für Sie dann von keinerlei Nutzen.

Eines der primären Designziele bei der ARexxBox, und speziell natürlich dem erzeugten Code, war möglichst nahe an die Vorgaben und Ideen des "User Interface Style Guide" von Commodore heranzukommen.

Wenn Sie dieses Buch (oder die Kapitel zu ARexx) noch nicht gelesen haben, wäre diese Lektüre vor der Benutzung der ARexxBox erst einmal angebracht.

Meiner Meinung nach ist eine Normung der ARexx--Interfaces bezüglich Syntax und Verhalten dringend notwendig; es kann nicht angehen, daß der Anwender für jedes neue Programm erst wieder ein anderes Konzept und andere Befehle lernen muß.

Allerdings sind einige der Ansätze im Style Guide nicht mehr als das -- Ansätze. Speziell bei den Parametern der Standardfunktionen bleiben an vielen Stellen Fragen offen. Ich habe versucht, diese Stellen zumindest teilweise bei der Übernahme in die ARexxBox zu ergänzen, wenn trotzdem Verständnisprobleme auftreten bitte ich um Nachricht, am Besten sofort mit Verbesserungsvorschlag.

Die ARexxBox ist FreeWare, das heißt unter den im Kapitel zu Copyright festgelegten Bedingungen frei verteilbare, aber urheberrechtlich geschützte Software. Weder für die Box selbst, noch für den erzeugten Code müssen irgendwelche Lizenzen an die Autoren gezahlt werden.

Wenn Sie den von der ARexxBox erzeugten Code direkt oder verändert in ihren Applikationen einsetzen, dann muß dies im About--Fenster und in der Dokumentation erwähnt werden. Außerdem würde ich mich freuen, wenn Sie mir eine kurze Nachricht darüber zukommen lassen (bevorzugt per EMail), eine Übersicht über die implementierten ARexx--Befehle wäre auch nicht schlecht, muß aber nicht sein.

## 1.4 ARexxBox.guide/Adresse des Autors

Adresse des Autors

\*\*\*\*\*

Der Autor kann unter folgenden Adressen erreicht werden:

Postadresse:

Michael Balzer  
Wildermuthstraße 18  
W-5828 Ennepetal  
GERMANY

InterNet Electronic Mail:

An der Uni:  
balzer@heike.informatik.uni-dortmund.de

Zu Hause:  
    bilbo@bagsend.aworld.de  
oder im Z-Netz  
    m.balzer@aworld.zer  
oder von Fido aus  
    Michael Balzer of 2:241/5604.19

Die Fido--Adresse wurde noch nie getestet, wenn kein Reply kommt  
bitte andere Adresse benutzen.

Bei Fragen zum Oberon-2--Source wenden Sie sich am besten direkt an  
dessen Autor:

Postadresse:  
    hartmut Goebel  
    Aufseßplatz 5  
    W-8500 Nürnberg 40           neue Postleitzahl ab 1. Juli 1993: 90459  
    GERMANY

InterNet Electronic Mail:  
    Zu Hause:  
        hartmut@oberon.nbg.sub.org  
oder im Z-Netz (eher selten)  
        hartmut@asn.zer  
oder von Fido aus  
        Harmut Goebel of 2:246/81.1

## 1.5 ARexxBox.guide/Einleitung

Einleitung  
\*\*\*\*\*

Die Wichtigkeit und Leistungsfähigkeit von ARexx auf dem Amiga  
dürfte Ihnen schon klar sein, sonst würden Sie sich wahrscheinlich  
nicht dieses Manual durchlesen. Ich muß Sie also nur noch davon  
überzeugen, daß die ARexxBox das richtige Werkzeug zum Erstellen der  
ARexx--Anbindung für Ihr Programm/Projekt ist ;-).

Warum ARexxBox?  
=====

Die ARexxBox (inspiriert von der GadToolsBox von Jan van den Baard)  
ist ein Tool, das das Erstellen eines ARexx--Interfaces für ein  
Programm extrem erleichtert und vereinfacht und dabei ein  
ARexx--Interface erzeugt, das den Anforderungen des "User Interface  
Style Guide" genügt.

Die Fähigkeiten in Stichpunkten:

- \* Die Syntax und Resultaterzeugung der ARexx--Befehle entspricht in  
allen Punkten den vom Style Guide vorgeschlagenen Regeln, z.B.  
werden Argumenttemplates wie bei DOS-Befehlen (ReadArgs) benutzt  
und die Keywords VAR und STEM werden automatisch unterstützt.



- \* Zu jedem Befehl kann eine beliebig große Argumentenliste und eine beliebig große Resultatenliste angelegt werden.
- \* Für Argumente sind alle ReadArgs()--Templates erlaubt und werden \*automatisch\* unterstützt, für Resultate werden die optionalen Templates /N und /M unterstützt.
- \* Intuitionoberfläche (erstellt mit GadToolsBox :-).
- \* Erzeugt wahlweise C--Source oder Oberon-2--Source (1)
- \* Auf Wunsch wird Source für eine CommandShell erzeugt, d.h. eine Shell in der die ARexx--Befehle direkt eingegeben und ihre Ausgaben betrachtet werden können.
- \* Die CommandShell kann auch zum Ausführen von Macrodateien benutzt werden.
- \* Ein Programm kann beliebig viele ARexx--Ports und CommandShells öffnen
- \* Die vom Style Guide vorgeschlagenen Standard--Kommandos habe ich schon für Sie abgetippt. Beispielcode für einige dieser Befehle liegt auch bei.

----- Footnotes -----

(1) Der Oberon-2--Source wurde objektorientiert und "type-save" designet.

## 1.6 ARexxBox.guide/Systemanforderungen

### Systemanforderungen

\*\*\*\*\*

#### Hardware

=====

An die Hardware werden so gut wie keine Anforderungen gestellt, die ARexxBox sollte auf jedem Amiga laufen.

Naja, \*eine\* Anforderung wird gestellt: Da das ARexxBox--Fenster mehr als 200 Zeilen hoch ist, muß eine NTSC--MedRes--Workbench im Overscan betrieben werden damit das Fenster geöffnet werden kann. Alternativ könnte man auch einen 6 oder 7 Punkte großen Systemfont einstellen.

Da Programmierer aber naturgemäß mit höheren Auflösungen arbeiten, stellt sich dieses Problem wohl kaum.

#### Software

=====

Bedingung zur Lauffähigkeit sowohl der ARexxBox als auch des erzeugten Codes ist mindestens AmigaOS 2.04, d.h. mindestens Kickstart 37.175 und Workbench 37.67.

Sie sollten sich außerdem eine Kopie von Nico Francois' `retools.library` besorgen, wenn Sie noch keine haben. Diese wird nur für die Box benötigt, der erzeugte Code benutzt sie nicht.

#### C-Source =====

Um den erzeugten Source zu kompilieren benötigen Sie einen ANSI--C--Compiler, der auch Funktionen aus der `'amiga.lib'` von Commodore importieren und verarbeiten kann.

Sie benötigen mindestens Version 37.32 (12.11.91) der `'amiga.lib'`, um den erzeugten Code linken zu können. Diese Version ist im Native Developer's Upgrade Kit V37.4 (11/91) (erhältlich bei Hirsch&Wolf) enthalten und wird auch schon bei manchen Compilern mitgeliefert.

Sie können auch ohne die `'amiga.lib'` auskommen. Ich habe die benötigten Funktionen (`'GetRexxVar'` und `'SetRexxVar'`) reassembliert und dem Paket beigelegt. Der Aztec C Compiler braucht sehr lange, um Funktionen aus der `'amiga.lib'` zu extrahieren, daher verkürzt das Linken mit diesem Modul den Linkvorgang erheblich.

Andererseits kann ich natürlich für die Kompatibilität der reassemblierten Funktionen nicht garantieren, und von zukünftigen Verbesserungen in der `'amiga.lib'` machen Sie damit auch keinen Gebrauch.

#### Oberon-2--Source =====

Für den Oberon-2--Source benötigen Sie AmigaOberon V3.0 oder höher.

Der Source baut auf einer Klassenbibliothek auf, die im Lieferumfang des Compilers nicht enthalten ist, ebenso einige andere benötigte Module. Diese befinden sich jedoch alle mit Quelltext im ARB-Paket.

Das Modul RVI stellt eine Schnittstelle zu einem Teil der `amiga.lib` her. Diese muß dann beim Linken mit angegeben werden, z. B.

    OLink RxTest OBJ oberon:obj/amiga.lib  
Mehr zur `amiga.lib` finden Sie oben im Abschnitt C--Source.

Sollten Sie das Object-File `"rexxvars.o"` aus einem der ARexx Entwickler-Kits haben, können Sie dieses gleiche mit der Option `$JOIN` an das Object-File des Moduls RVI anhängen. Die gesonderte Angabe der `amiga.lib` beim Linken entfällt dann. Wegen der Größe der `amiga.lib` empfiehlt sich dieses Verfahren mit dieser nicht.

## 1.7 ARexxBox.guide/Installation

---

## Installation

\*\*\*\*\*

Im Prinzip ist bei der ARexxBox keine Installation nötig. Nur im Oberon-Teil sind einige benötigte Module (1), die nicht von der Box selbst erzeugt werden. Diese sollten an eine geeignete Stelle kopiert werden, wo sie der Compiler auch findet.

Wo Sie den Rest unterbringen (z.B. die Dokumentation und die ARexxBox selbst) ist Ihnen überlassen.

----- Footnotes -----

(1) Diese Module dienen nicht nur dem von ARB erzeugten Source, sondern können natürlich auch in anderen Projekten verwendet werden.

## 1.8 ARexxBox.guide/Bedienung

### Bedienung

\*\*\*\*\*

An der Bedienung der ARexxBox ist absolut nichts ungewöhnliches. Sie sollten auf Anhieb damit zurechtkommen, wenn Sie jemals mit einer Amiga--Applikation gearbeitet haben.

Bevor Sie mit der Eingabe von Befehlen beginnen, sollten Sie vielleicht das Kapitel zum Konzept durchlesen. Als Anschauungsbeispiel können Sie z.B. erst mal aus dem Verzeichnis 'arb' die Datei 'Misc.arb' laden. Klicken Sie in der linken Liste (Commands) auf die Zeile 'GETATTR'.

Die beiden rechten Listen füllen sich daraufhin mit den Parametern und Resultaten des Kommandos 'GETATTR'. Jedes Element jeder Liste kann über das Stringgadget unter der Liste verändert werden.

Eingabe	Vorgehensweise für einen neuen Befehl
MsgPort Basename	
CommandShell	
Merge in	
Print	
Generate Source	C oder Oberon, das ist hier die Frage
Clipboard	Cut, Copy und Paste

## 1.9 ARexxBox.guide/Eingabe

### Eingabe

=====

---

Für die Eingabe einer Befehlsdefinition empfiehlt sich folgende Vorgehensweise: Taste N drücken, Namen des ARexxbefehls eingeben und mit RETURN bestätigen. Die ARexxBox wandelt den Namen in Großbuchstaben und unzulässige Zeichen in Underscores ('\_') um und prüft den Namen auf Eindeutigkeit. Falls schon ein solcher Befehl existiert kann der Name korrigiert werden.

Danach legen Sie die Argumente fest. Für jedes Argument drücken Sie die Taste E und geben dann den Namen ein. Auch dieser wird auf Eindeutigkeit geprüft.

Genau so verfahren Sie für die Resultatenliste, nur daß das Shortcut für ein neues bzw. weiteres Resultatfeld dort die Taste W ist.

Argumente und Resultate können entweder direkt nach der Eingabe oder später nach dem Anwählen per Mausklick in der Liste mit den jeweiligen 'up'-- und 'do'--Gadgets nach oben bzw. unten verschoben werden oder mit der jeweiligen 'Remove'--Funktion gelöscht werden.

Die Typen der Argumente und Resultate werden im ReadArgs()--Stil (siehe AutoDocs) an die Namen angehängt, also zum Beispiel ARG1/K/N oder LISTE/M. Bei den Resultatfeldern sind nur die Switches /M (für Liste) und /N (für Numerisch) erlaubt.

## 1.10 ARexxBox.guide/MsgPort Basename

MsgPort Basename  
=====

Der 'MsgPort Basename' ist der Default--Basisname für alle von der Applikation eröffneten ARexx--Ports. Bei Bedarf (schon existentem Port unter dem Namen) wird so lange eine steigende Nummer an den Basisnamen angehängt bis der Port ohne Konflikt geöffnet werden kann.

Auf jeden Fall sollte aber die Applikation (wie im Demo--Programm gezeigt) ein Argument bzw. Tooltype namens "PORTNAME" verstehen, damit der Benutzer Einfluß auf die Namensgebung nehmen kann. Ein solcher per Argument festgelegter Portname wird zum neuen Basisnamen und ggf. auch (automatisch) entsprechend durch eine angehängte Nummer ergänzt.

Der 'MsgPort Basename' wird außerdem als Standard--Dateiextension für von der Applikation gestartete ARexx--Skripten eingesetzt. Dies ist unabhängig von einem eventuell vom Anwender gesetzten Portnamen.

## 1.11 ARexxBox.guide/CommandShell

CommandShell  
=====

Mit diesem Gadget bestimmen Sie, ob bei der Sourcegenerierung Code für eine CommandShell erzeugt werden soll. Diese Option sollte im Normalfall ruhig eingeschaltet sein, da die CommandShell--Funktionen auch zum Ausführen externer Makros nützlich sein können.

Für Oberon wird der CommandShell--Source immer erzeugt.

Die CommandShell ist eine Art Shell, die sämtliche ARexx--Befehle einer Applikation ausführen kann, als wären diese von ARexx gekommen. Dabei werden die Resultate der Befehle in lesbarer Form in dem Shellfenster ausgegeben.

Die CommandShell eignet sich z.B. dafür, mal schnell ein ARexx--Kommando abzusetzen ohne gleich ARexx selbst zu bemühen. Man kann hier auch einige Befehle erlauben, die nur dem geübten Anwender zur Verfügung stehen sollten.

Wenn man die CommandShell mit einer Datei als Eingabekanal versorgt, arbeitet sie die darin enthaltenen Kommandos Zeile für Zeile ab. Auf diese Art und Weise kann sie auch für Makros verwendet werden.

## 1.12 ARexxBox.guide/Merge in

Merge in  
=====

Diese Funktion ergänzt die im Speicher befindliche Kommandoliste um die ARexx--Kommandos aus dem auszuwählenden ARexxBox--File, die noch nicht in der Liste vorhanden sind.

Dadurch ist es möglich, sich Pakete von ARexx--Kommandos für verschiedene Zwecke anzulegen, und diese dann nach Bedarf zu kombinieren.

## 1.13 ARexxBox.guide/Print

Print  
=====

Soll eine Dokumentationshilfe sein. Für jeden Befehl wird der Name, die Syntax und die Resultate ausgegeben.

Vorschlägen für eine andere und bessere Formatierung gegenüber bin ich sehr aufgeschlossen.

## 1.14 ARexxBox.guide/Generate Source

---

## Generate Source

=====

Hiermit wird der Sourcegenerator gestartet. Der im FileRequester eingegebene Name wird um die evtl. vorhandene Extension (`.c`, `.h` oder `\_rxif.c` bzw. für Oberon `.mod`) gekürzt, dann wird nach ein paar Sicherheitschecks (hauptsächlich, ob durch die Generierung ein anderes Modul unabsichtlich überschrieben werden könnte) mit der Sourcegenerierung begonnen.

Jegliche Generierung findet zunächst einmal in `T:` statt, um der Möglichkeit eines Absturzes während des Schreibens zumindest teilweise den Schrecken zu nehmen. Erst nach korrektem Abschluß der Generierung werden die Dateien an ihren tatsächlichen Bestimmungsort kopiert.

Wenn beim Kopieren irgendein Fehler passiert, werden die Originaldateien in `T:` nicht gelöscht, so daß Sie diese zur Not noch von Hand an einen sicheren Ort kopieren können.

Beim Generieren des Sources wird für alle Kommandos der Status auf "Alt" gesetzt. Wenn ein Kommando in der Box verändert wird, wird der Status auf "Neu" gesetzt. Daran erkennt der Sourcegenerator, bei welchen Kommandos sich eventuell etwas geändert hat, und erzeugt vor den Interfacefunktionen im Source einen Kommentar ("ATT: Interface changed!"), der darauf hinweist.

Wo dieser Kommentar steht, sollte der Programmierer vor dem nächsten Compilerlauf die Konsistenz des Interfaces und der RXD--Struktur mit der tatsächlichen (eventuell auf einem älteren Interface basierenden) Routine überprüfen und wo nötig den Source den neuen Gegebenheiten anpassen.

Damit die Änderung des Status' der Kommandos auch gespeichert wird (muß ja über mehrere Sitzungen hinweg erhalten bleiben), wird beim Generieren des Sources auch das gesamte Projekt als 'geändert' gekennzeichnet. Sinnvollerweise speichern Sie das Projekt einfach direkt nach der Sourcegenerierung.

Zur Zeit stehen zwei Sourcegeneratoren zur Verfügung, einer für C (von mir) und einer für Oberon (von hartmut Goebel). Bei Fragen bezüglich Oberon bitte an hartmut wenden.

## 1.15 ARexxBox.guide/Clipboard

### Clipboard

=====

Cut, Copy und Erase beziehen sich entweder auf das aktuelle Kommando oder auf einen vorher festgelegten Bereich.

Um einen Bereich zu markieren, wählen Sie "Mark Range" an, klicken dann zuerst auf den ersten Befehl des gewünschten Bereiches, und dann

auf den letzten. Alle Befehle vom ersten bis zum letzten würden nun bei einem anschließenden Copy kopiert werden.

Die Anwahl eines beliebigen Befehls löscht die Bereichsmarkierung.

Paste setzt nur die Befehle in die Datei ein, die vorher noch nicht darin waren (wie Merge).

## 1.16 ARexxBox.guide/Konzept

Konzept

\*\*\*\*\*

Neben der Style--Guide--Konformität war ein anderes primäres Designziel der ARexxBox, dem Programmierer die Handhabung des ARexxPorts so leicht wie möglich zu machen.

Initialisierung	Port öffnen
Kommandos von ARexx	Dispatcher und Interfacefunktionen
Expandierung von Befehlen	ExpandRXCommand() und Externe Befehle
Kommandos an ARexx	SendRexxCommand()
CloseDown	Port schließen

## 1.17 ARexxBox.guide/Initialisierung

Initialisierung

=====

Zum Öffnen des ARexxPorts dient die Funktion SetupARexxHost, die einen Zeiger auf ein Objekt vom Typ 'struct RexxHost' zurückgibt. Dieser Zeiger ist als eine Art FileHandle für den ARexxPort zu verstehen, alle weiteren ARexx--Funktionen der ARexxBox benötigen diesen Zeiger.

In Oberon-2 liefert SetupARexxHost ebenfalls einen Pointer, der als Empfänger (engl. "receiver") der Methoden dient.

Somit beziehen sich auch alle Operationen immer auf einen bestimmten Port (eine Applikation kann ja durchaus auch mehrere Ports öffnen, z.B. könnte ein Texteditor für jedes offene File einen Port öffnen).

## 1.18 ARexxBox.guide/Kommandos von ARexx

Kommandos von ARexx

=====

Während des Betriebs muß die Applikation im einfachsten Fall nichts weiter tun, als bei einer am ARexxPort eingegangenen Message die Funktion ARexxDispatch aufzurufen. Diese holt sich selbständig alle Messages des angegebenen REXXHosts und führt alle nötigen Schritte durch, um die Requests zu befriedigen.

Zu jedem Kommando gehört eine Interface--Funktion, deren Adresse in der Kommandoliste vermerkt ist. Diese Interface--Funktion enthält den eigentlichen Code zum Befehl, üblicherweise ruft dieser nach einer Parameterüberprüfung die entsprechenden Kernfunktionen auf. Alles was Sie als Programmierer tun müssen, ist das Schreiben dieser Interfacefunktion -- wobei die Box bei neuen Funktionen natürlich ein Leermuster erzeugt.

Die Interfacefunktion bekommt die Parameter des Kommandos über eine speziell angelegte Datenstruktur, führt die Operation durch und gibt über dieselbe Datenstruktur eventuelle Resultate an die ARexxBox--Routinen zurück. Die ARB--Routinen sorgen dann dafür, daß auch das Resultat Style--Guide--konform an ARexx zurückgeschickt wird.

## 1.19 ARexxBox.guide/Expandierung von Befehlen

### Expandierung von Befehlen

=====

Um den ALIAS--Befehl (Standard) und andere Möglichkeiten wie Makros zu unterstützen gibt es einen Callback, der vom Parser aufgerufen wird, bevor ein als unbekannt eingestuftes Kommando einen Fehler erzeugt.

Der Parser ist der Teil der ARexxBox, der versucht, zu erkennen um welches Kommando es sich bei einer Message handelt. Wenn diese Erkennung fehlschlägt, ruft er die Funktion ExpandRXCommand() auf, die z.B. eine Liste von Aliases durchgehen kann und eine geeignete Substitution vornimmt.

Anschließend versucht der Parser noch einmal, das Kommando zu ermitteln.

Wenn dieser zweite Anlauf ebenfalls fehlschlägt, wird das Kommando an ARexx weitergereicht (das ursprüngliche, nicht das expandierte). Falls ein externes Skript unter dem Namen existiert wird dieses also völlig transparent aufgerufen. Auch das Resultat eines solchen externen Befehls wird korrekt an den ursprünglichen Caller zurückgegeben.

Demonstration: Wechseln Sie in das Verzeichnis 'test' und starten Sie 'test'. Beenden Sie die CommandShell durch EOF (C-\) und geben Sie in einer anderen Shell folgendes Kommando ein:

```
'rx "options results; address 'arbttest'; test 'FooBar'; say result'"
```

Da das Testprogramm keinen internen Befehl "test" kennt, wird das externe Skript 'test.arbttest' gestartet, welches im Ausgabefenster des



Testprogramms eine Meldung ausgibt und dann den String "Testtext!" zurückgibt (ja, ich weiß, sehr einfallsreich).

Erst wenn auch kein Skript unter dem gegebenen Namen existiert wird der Befehl endgültig als "unbekannt" eingestuft und ein Fehler erzeugt.

Hier noch mal der Ablauf für ein Kommando:

1. Kommando intern bekannt?
2. Falls nein: ExpandRXCommand() -> neues Kommando bekannt?
3. Falls nein: Existiert ein Skript unter dem Namen?
4. Falls nein: Error - Not Implemented

## 1.20 ARexxBox.guide/Kommandos an ARexx

Kommandos an ARexx  
=====

Die Applikation kann natürlich auch beliebig eigene Kommandos an ARexx senden, dazu stehen einige Hilfsfunktionen zur Verfügung (siehe SendRexxCommand).

Da solche Kommandos prinzipiell (als Messages) asynchron ablaufen, bietet die ARexxBox die Möglichkeit, beim Eintreffen eines Replies zu solch einem Kommando eine spezielle Funktion aufzurufen, die z.B. das Resultat des Kommandos auswerten kann. Dabei muß natürlich der Programmierer Sorge tragen, daß er im nachhinein einen Reply dem richtigen Kommando zuordnen kann (siehe Einbindung).

Um späte Rückmeldungen von gesendeten Kommandos nicht ins Leere laufen zu lassen zählen die Boxroutinen sämtliche ausgehenden Kommandos, der Port bleibt so lange offen, wie noch Rückmeldungen erwartet werden (siehe CloseDown).

## 1.21 ARexxBox.guide/CloseDown

CloseDown  
=====

Das Gegenstück zu SetupARexxHost ist dann folgerichtig CloseDownARexxHost. Diese Funktion schließt den Port und gibt alle angeforderten Ressourcen frei.

Wenn Kommandos an ARexx geschickt wurden, kann sich das endgültige Schließen des Ports verzögern, da die Box dann zunächst auf alle noch ausstehenden Replies warten muß. Wenn während dieser Wartephase neue Kommandos am Port eintreffen, werden diese sofort mit dem Fehler "Host

---

closing down" an ARexx zurückgeschickt.

## 1.22 ARexxBox.guide/Dateien

Dateien

\*\*\*\*\*

Die ARexxBox erzeugt Source, der im Prinzip ohne jede Änderung sofort compilierbar sein sollte. Sämtliche Interfacefunktionen müssen theoretisch im Code zumindest als Leerfunktionen generiert worden sein.

C

=

Für C werden fünf Dateien erzeugt:

- 'name.c' enthält die grundlegenden Funktionen wie SendRexxCommand(), SetupARexxHost() und ARexxDispatch(). Dieses Modul muß im Normalfall nur ein Mal compiliert werden, da sich (außer beim Verändern des Gadgets 'CommandShell') nichts ändert.
- 'name.h' enthält die grundlegenden Konstanten und Strukturen zum ARexxPort. Hier befinden sich auch die Prototypen und Strukturen aller im jeweiligen Projekt verwendeten Interfacefunktionen.
- 'name\_rxcl.c' enthält die globalen Variablen, allen voran die Liste der Kommandos, die das Interface verstehen soll.
- 'name\_rxif.c' enthält die Interfacefunktionen. Damit ist dieses Modul das einzige, das von Ihnen verändert werden muß. Alte Funktionen werden intelligent in den neu erzeugten Source übernommen, nicht mehr benötigte Funktionen wandern dabei in den
- 'name\_rxifstore', wo sie bei Bedarf wieder herausgeholt werden können.

Oberon

=====

Für Oberon werden vier Dateien erzeugt:

- 'RxName.mod' enthält die Kommandoliste und einige grundlegenden Funktionen wie SetupARexxHost().
  - 'NameARB.mod' enthält die grundlegenden Konstanten.
  - 'NameRXIF.mod' enthält die Interfacefunktionen und die Datenstrukturen dazu. Damit ist dieses Modul das einzige, das von Ihnen verändert werden muß. Funktionen werden intelligent in den neu erzeugten Source übernommen, nicht mehr benötigte Funktionen wandern dabei in den
  - 'Name\_rxifstore', wo sie bei Bedarf wieder herausgeholt werden können.
-

ARB--Kommentare  
=====

Die ARexxBox generiert im REXX--Interface--Modul einige spezielle Kommentarzeilen der Form `'/* $ARB: ... */'` (bei Oberon `'(* !ARB: ... *)'`), die Sie nicht verändern dürfen.

Am Anfang des Interface--Moduls steht eine Kommentarzeile, durch die die ARexxBox prüfen kann, ob dieses Sourcemodul zur aktuellen Interfacedefinition gehört.

Zu jeder Interfacefunktion gehören zwei Kommentarzeilen, alles zwischen diesen beiden wird als zu der Interfacefunktion gehörend betrachtet und komplett nach rxifstore ausgelagert, wenn die Funktion nicht mehr benötigt wird. Wenn Sie also spezielle Variablen und Funktionen zu einer Interfacefunktion benötigen, denken Sie daran, diese zwischen den beiden Kommentarzeilen zu definieren, so bleibt alles beisammen.

Wenn die Definition eines Kommandos verändert wurde, erzeugt die Box vor der entsprechenden Interfacefunktion eine Kommentarzeile der Form `"ATT: Interface changed!"`. Dieser Kommentar darf gelöscht werden, allerdings sollte man wirklich erst prüfen, ob der Code an die neuen Argumente bzw. Resultate angepaßt werden muß.

## 1.23 ARexxBox.guide/Interfacefunktionen

Interfacefunktionen  
\*\*\*\*\*

Allgemein  
=====

Jede Interfacefunktion besteht aus drei Teilen,

- \* Initialisierung,
- \* Eigentliche Funktion und
- \* Freigabe,

die in dieser Reihenfolge von den ARexxBox--Routinen aufgerufen werden.

Die erste Phase dient dabei der Allokierung und Initialisierung des Speichers für die Datenstruktur, in der Parameter und Resultate zwischen Interfacefunktion und ARB--Routinen hin und her transportiert werden.

Man kann diese Phase auch für andere Initialisierungen benutzen, z.B. Allokierung von lokalem Pufferspeicher o.ä..

Die zweite Phase führt die eigentliche Operation durch und benutzt

---

dabei die Parameter aus der in der ersten Phase allokierten Transferstruktur und setzt die eventuell anfallenden Resultate ebenfalls dort ein.

Es gibt zwei Felder in jeder Transferstruktur, rc und rc2. Diese beiden entsprechen den normalen Returncodes, die jeder ARexxbefehl erzeugen kann. Die ARexxBox nimmt dabei allerdings noch eine kleine Erweiterung vor (siehe Fehler).

Nach der Ausführung von Phase zwei übernimmt die ARexxBox--Routine die Resultate und schickt diese an ARexx zurück. Dann wird noch Phase drei aufgerufen, die der Interfacefunktion Gelegenheit gibt, angeforderte Ressourcen wieder freizugeben.

Einige Beispiele, wie Interfacefunktionen aufgebaut sein sollten und funktionieren, finden Sie im Verzeichnis 'rxif', speziell die HELP--Funktion dürfte ein gutes einführendes Beispiel abgeben.

Hier mal die konkrete Struktur einer Interfacefunktion:

```
void rx_help( struct REXXHost *host, struct rxd_help **rxd,
              long action, struct REXXMsg *rexxmsg )
{
    struct rxd_help *rd = *rxd;    /* zur Vereinfachung und für lokale
                                    Erweiterungen */

    struct rxs_command *rxc;        /* lokale Variablen */
    int cnt = 1;

    switch( action )
    {
        case RXIF_INIT:
            /* Erste Phase! */
            /* Hier wird nun die Transferstruktur allokiert */
            *rxd = calloc( sizeof *rd, 1 );
            /* Defaultwerte setzen wir hier mal nicht, also */
            break;

        case RXIF_ACTION:
            /* Zweite Phase! */
            /* Hier wird gearbeitet */

            if( rd->arg.prompt )
            {
                rd->rc = -10;
                rd->rc2 = (long) "Prompt option not yet implemented";
                return;
            }

            usw. usf., unter anderem werden hier die Allokationen
            rd->res.commanddesc = malloc( ... )
            und
            rd->res.commandlist = [calloc];
            gemacht, die in Phase drei wieder freigegeben werden müssen.

            break;
```

```

    case RXIF_FREE:
        /* Dritte und letzte Phase! */
        /* Angeforderten Speicher zurückgeben */

        if( rd->res.commanddesc )
            free( rd->res.commanddesc );
        if( rd->res.commandlist )
            free( rd->res.commandlist );

        /* Transferstruktur freigeben */
        free( rd );
        break;
    }

    /* Zurück zum Dispatcher */
    return;
}

```

#### Lokaler Speicher =====

Wenn Sie in einer Interfacefunktion lokalen Speicher benötigen, sollten Sie *\*keinesfalls\** auf statische oder globale Variablen zurückgreifen -- denken Sie immer daran, daß bei mehreren Ports ein und dieselbe Interfacefunktion quasi gleichzeitig mehrfach in Benutzung sein kann. Außerdem widersprechen lokale statische Variablen jedem objektorientierten Design (und würden z.B. bei Oberon schon Probleme bereiten).

Ein einfacher Trick verschafft Ihnen beliebige lokale Speicher: Erweitern Sie einfach die Transferstruktur am Ende um die gewünschten Felder. Beispiel:

```

void rx_rx( struct REXXHost *host, struct rxd_rx **rxd,
            long action, struct REXXMsg *rexxmsg )
{
    struct {
        /* Einbettung der Originalstruktur */
        struct rxd_rx rd;
        /* Es folgen die Erweiterungen */
        long tempval;
        char *tempbuffer;
        ...
    } *rd = (void *) *rxd;
    ...
}

```

Oder in Oberon:

```

PROCEDURE Rx * (host: rxh.REXXHost; VAR rxd: rxh.RXDPtr; action: INTEGER);
VAR
    rd: POINTER TO RECORD (rxdRx) (* Erweiterung der Originalstruktur *)
        tempval: LONGINT;
        tempbuffer: BT.DynString;
        ...
    END;
...

```

Wie man sieht wurde hier eine lokale Erweiterung der Struktur `rxd_rx` vorgenommen, die 'lokale' Variablen (`tempval` und `tempbuffer`) zur Verfügung stellt. Da die Transferstruktur inklusive der Erweiterung in der ersten Phase allokiert und erst in der letzten freigegeben wird, steht das zusätzliche Feld dazwischen (während der zweiten Phase) konfliktfrei zur Verfügung.

#### Beispiele

=====

Im Verzeichnis `'rxif'` finden Sie einige Beispielbefehle bzw. die Interfacefunktionen dazu. Einige davon sind quasi gebrauchsfertig, z.B. braucht der `HELP--`Befehl nur eine anwendungsspezifische graphische Oberfläche.

Wenn Sie einen allgemein brauchbaren (sprich applikationsunabhängigen) Befehl geschrieben haben und diesen auch den anderen ARexxBox--Benutzern zur Verfügung stellen wollen, dann schicken Sie mir bitte die Befehlssyntax und den Source.

Mir schwebt eine Sammlung von Standardbefehlen vor, die ohne großen Aufwand in ein beliebiges Programm eingebunden werden können, so eine Bibliothek von ARexx--Befehlen würde die ARexx--Interfaceentwicklung noch weiter vereinfachen.

## 1.24 ARexxBox.guide/Argumente&Resultate

### Argumente & Resultate

\*\*\*\*\*

#### Argumente

=====

Das Konzept der ARexxBox orientiert sich sehr eng an den Ideen und Vorschlägen aus dem Style Guide.

Eine wichtige Grundlage ist hierbei, daß die Argumente der ARexx--Befehle genau so funktionieren und aussehen sollen, wie die Argumente von Shellbefehlen. Das setzt die ARexxBox dann auch folgerichtig um, indem zum Parsen der Parameter eines konkreten Befehls die DOS--Funktion `ReadArgs()` benutzt wird.

Zunächst einmal sollten Sie nun das Kapitel zu `ReadArgs` durchlesen.

Sobald mindestens ein Resultat vorgesehen ist, erzeugt die ARexxBox automatisch zwei weitere Argumente (erst bei der Sourcegenerierung, in der Box werden diese nicht angezeigt): `VAR` und `STEM`.

Der Anwender kann durch `VAR` und `STEM` steuern, in welcher Form die Resultate eines Befehls in ARexx--Variablen abgelegt werden. Es gibt zwei solche Formen, die eine gibt sämtliche Resultate in einer Variable zurück, die andere benutzt eine Stem--Variable um die einzelnen Resultatfelder einzelnen Stem--Feldern zuzuweisen.

Wenn weder VAR noch STEM spezifiziert wurde, wird die Form der Rückgabe in einer Variable benutzt, und zwar in der Standardvariable 'RESULT'. Dabei besteht der Variableninhalt einfach aus der Verkettung aller Ergebnisfelder, wobei die einzelnen Felder durch Leerzeichen getrennt sind.

Sobald strukturierte Rückgaben erforderlich sind, sollte die STEM--Rückgabe verwendet werden.

VAR und STEM schließen sich nicht gegenseitig aus, beide Rückgabemethoden können kombiniert werden. Es wird nur kein 'RESULT' mehr erzeugt, sobald eine dieser beiden Methoden explizit vom Anwender gewählt wird.

Resultate  
=====

Aus Gründen der Konsistenz benutzt die ARexxBox für Resultate von Befehlen \*exakt die gleichen\* Konventionen wie für Argumente, mit der Einschränkung, daß nicht typgebende Switches (wie /A oder /K) ignoriert werden. Außerdem wird der Typ Bool (/S und /T) nicht unterstützt, benutzen Sie stattdessen Integer (flexibler da ein Bool immer in der Ausgabe auftauchen würde, ein Integer nur wenn er gesetzt wurde).

Konkret gilt folgende Typäquivalenz für die Resultate:

- \* 'FOOBAR' ist ein String
- \* 'FOOBAR/M' ist ein Array von Strings
- \* 'FOOBAR/N' ist ein Integer (oder Bool)
- \* 'FOOBAR/N/M' ist ein Array von Integeren

Das bedeutet, daß auch bei den Resultaten (die ja von \*Ihrer\* Interfacefunktion erzeugt werden!) exakt die gleichen Typen erwartet werden, wie bei den Argumenten. Sämtliche Resultatfelder sind damit natürlich auch optional.

Nehmen wir mal ein konkretes Beispiel: Öffnen Sie in der Box die Datei 'arb/Other.arb' und wählen Sie den Befehl 'HELP' aus. Sie sehen, daß der Help--Befehl offensichtlich zwei Resultate zuläßt, einen einfachen String ('COMMANDESC') und ein Array von Strings ('COMMANDLIST').

Das bedeutet, daß die ARexxBox für diesen Befehl folgende Felder als Resultatfelder in der Struktur 'rxd\_help' (Transferstruktur) erzeugen wird:

In C:

```
char *commanddesc;  
char **commandlist;
```

In Oberon (die Typbezeichner wurden für dieses Beispiel aufgelöst):

---

```
commanddesc: BasicTypes.DynString;
commandlist: POINTER TO ARRAY OF BasicTypes.DynString;
```

Ihr Help--Befehl kann nun beliebige dieser beiden Felder setzen, indem er ihnen korrekte Werte zuweist, also z.B.

```
static char *myarray[4] = { "foo", "bar", "dubidu", NULL };
rxd->res.commanddesc = "Blafasel";
rxd->res.commandlist = myarray;
```

In Oberon muß dies etwas anders gemacht werden, da die Längeninformation ebenfalls benötigt wird. Dafür benötigen die Listen kein abschließendes NIL:

```
rxd.res.commanddesc := MoreStrings.CopyString("Blafasel");
NEW(rxd.res.commandlist,3); (* drei Einträge *)
rxd.res.commandlist[1] := MoreStrings.CopyString("foo");
rxd.res.commandlist[2] := MoreStrings.CopyString("bar");
rxd.res.commandlist[3] := MoreStrings.CopyString("dubidu");
```

(Anm.: Der normale HELP--Befehl erzeugt in Abhängigkeit von den Parametern immer nur eins der beiden Resultate, aber es wäre durchaus möglich, beide Felder zu besetzen.)

Die ARexxBox kümmert sich nun darum, die Resultate Style--Guide--konform an ARexx zurückzugeben: Integers werden in Strings umgewandelt, bei Arrays werden die Einträge gezählt und die Anzahl gefolgt von den einzelnen Einträgen des Arrays zurückgegeben.

Sie können das Verhalten leicht ausprobieren: Starten Sie das Programm 'test/test' und geben Sie in der CommandShell 'help' ein. Ohne Parameter gibt der Help--Befehl die Liste der Kommandos aus, benutzt also nur das Feld 'rxd->res.commandlist'.

Die Ausgabe enthält die Anzahl der Listenelemente gefolgt von den Elementen selbst. Sie sehen nun das Format, in dem das Resultat bei einer ARexx--Anforderung in der REXX--Variablen RESULT gelandet wäre. Dieses Format können Sie mit dem Schlüsselwort 'VAR' in eine beliebige Variable umlenken.

Geben Sie nun 'help stem kl.' ein. Bei einer ARexx--Anforderung wäre die Stem--Variable kl genau so belegt worden wie angezeigt, d.h. bei Arrays wird vorab ein Element 'Resultatname.COUNT' angelegt, dem die N Elemente der Liste mit jeweils Benennung 'Resultatname.N' folgen.

Im obigen Beispiel würde das Resultat also lauten:

```
In VAR--Form:
  "BlaFasel 3 foo bar dubidu"
```

In STEM--Form mit der Wurzel hr (sinnvoller wegen der strukturierten Rückgabewerte):

```
hr.COMMANDDESC = "BlaFasel"
hr.COMMANDLIST.COUNT = 3
hr.COMMANDLIST.0 = "foo"
hr.COMMANDLIST.1 = "bar"
hr.COMMANDLIST.2 = "dubidu"
```



Das ARexx--Programm kann diese Liste mit einer einfachen Schleife abfragen:

```
help stem a.  
say 'Es gibt' a.commandlist.count 'Befehle im Testprogramm, und zwar:'  
do i=0 to a.commandlist.count-1  
    say a.commandlist.i  
end i
```

Um die Benennung anhand der Resultatnamen zu verdeutlichen geben Sie nun noch 'help stem einzel. help' ein. Der Help--Befehl erzeugt nun nur das Resultat 'COMMANDDESC' (Feld 'rxd->res.commanddesc').

## 1.25 ARexxBox.guide/Fehler

Fehler

\*\*\*\*\*

Wenn während der Ausführung eines Kommandos Fehler auftreten, müssen diese an das aufrufende Programm zurückgegeben werden, damit dieses entsprechend darauf reagieren kann.

ARexx bietet dazu normalerweise nur die Variable 'RC', die einen ganzzahligen Fehlercode enthalten kann, per Konvention im Bereich 0 bis 20.

Die ARexxBox bietet hier eine etwas erweiterte Möglichkeit über eine automatisch generierte weitere Variable namens RC2. In dieser Variable können sowohl Fehlercodes als auch Fehlerstrings (Beschreibungen) zurückgegeben werden, die dann vom ARexx--Programm abgefragt werden können.

Diese beiden Variablen entsprechen den Elementen rc und rc2, die in jeder Transferstruktur existieren. Ob der Inhalt von rc2 ein Integerwert oder die Adresse eines Strings ist, wird durch das Vorzeichen von rc bestimmt: Wenn rc negativ ist, ist rc2 ein String.

Beispiel für Integerrückgabe (am Besten wo möglich DOS--Fehlercodes verwenden):

```
rd->rc = 10;  
rd->rc2 = ERROR_NO_FREE_STORE;
```

Beispiel für Stringrückgabe:

```
rd->rc = -10;  
rd->rc2 = (long) "Prompt option not yet implemented";
```

In Oberon wird hier (wie in 'C') nur die Adresse des Strings übergeben, da er normalerweise konstant ist. Also:

```
rd.rc = -10;  
rd.rc2 = SYSTEM.ADR("Prompt option not yet implemented");
```

---

Ein Integer wird als String zurückgegeben, Strings werden ohne Veränderung an ARexx übergeben. Wenn rc negativ ist, wird das Vorzeichen vor der Rückgabe umgedreht.

Diese erweiterte Fehlerrückgabe steht zwar nicht im Style Guide, ist meines Erachtens aber praktischer als der einfache Fehlercode in RC, da man so detaillierte Fehlerursachen mit zurückgeben kann.

## 1.26 ARexxBox.guide/Einbindung

Einbindung

\*\*\*\*\*

Designrichtlinien

=====

Es empfiehlt sich, Applikationen objektorientiert von innen nach außen zu entwickeln. Im Idealfall besteht eine Applikation aus einem bestimmten Kern der eigentlichen Funktionen und einer Schicht von darauf aufsetzenden Funktionen, die zwischen dem Kern und der Außenwelt vermitteln.

Bei so einem Design reicht es, eine Schicht für die graphische Benutzeroberfläche zu implementieren -- die Schicht für ARexx besteht dann aus den von der ARexxBox generierten und von Ihnen ausgefüllten REXX--Interfacefunktionen. Weitere Interface--Schichten sind denkbar, z.B. für Terminals an der seriellen Schnittstelle, ...

Wenn ein Kern von Basisfunktionen existiert, vereinfacht das die Implementation der ARexx--Schicht stark.

Ein einfaches Beispiel: Wenn eine Applikation Dateien öffnet, kann z.B. eine Kernfunktion vorhanden sein, die nur den kompletten Dateinamen bekommt und das gesamte Einlesen der Datei vornimmt, eventuelle Fehler durch definierte Rückgabewerte an den Aufrufer zurückgibt.

Die GUI--Schicht enthält dann eine Funktion, die vom Window--Dispatcher aufgerufen wird, wenn der Benutzer aus dem Projekt--Menü den Punkt "Open" anwählt. Diese Funktion öffnet dann z.B. einen FileRequester um den Namen zu ermitteln und ruft dann die Kernfunktion auf, gibt eventuelle Fehler wiederum über einen geeigneten Requester an den Benutzer weiter.

Die entsprechende Routine in der ARexx--Schicht kommt mit weniger aus, da sie in der Regel den Namen der zu öffnenden Datei als Parameter bekommt. Sie muß also nur einen Parameter--Check vornehmen und ruft dann ihrerseits genauso die Kernfunktion auf. Eventuelle Fehler werden dann durch den Rückgabecode an ARexx weitergeleitet.

Beispiel

=====

---

Dieses Beispiel soll nicht das perfekte Schichtendesign wie oben vorgeschlagen demonstrieren, sondern nur die Anwendung der ARexxBox--Routinen.

Am Besten holen Sie sich nun das File 'test/test.c' in den Editor und sehen sich dort die Realisierung der folgenden Schritte an.

#### Initialisierung -----

Die einzige Library, die für die ARexxBox geöffnet werden muß, ist die 'rexsyslib.library'. Es ist natürlich sinnvoll, einen Parameter zu erlauben, mit dem der Benutzer den Namen des ARexxPorts bestimmen kann.

Wenn eine Routine angemeldet werden soll, die bei Resultaten von mit SendRexxCommand() abgesetzten Kommandos aufgerufen werden soll, sollte das zu Anfang durch die Zuweisung der Adresse der Routine an die Variable 'ARexxResultHook' geschehen. In Oberon überschreiben Sie hierfür einfach die abstrakte Methode HandleResult in einem geeigneten Modul des Projekts.

Um einen ARexx--Port zu öffnen und alle notwendigen Initialisierungen vorzunehmen wird nun SetupARexxHost() aufgerufen.

#### CommandShell -----

Die CommandShell benötigt einen Ein-- und einen Ausgabekanal. Im Beispiel wird eine Konsole (Fenster) dafür geöffnet, es wären auch ganz normale Dateien dafür denkbar (z.B. um Skripte auszuführen).

Die CommandShell selbst läuft synchron, der Aufruf kehrt also erst zurück, wenn in der Eingabe der CommandShell EOF aufgetreten ist.

Für das Schließen der Ein-- und Ausgabekanäle ist natürlich der Aufrufer zuständig.

#### SendRexxCommand -----

Wie man sieht, ist das Absetzen von Kommandos an ARexx sehr einfach.

Wenn Sie mehrere asynchrone Kommandos auch beim ResultHook unterscheiden wollen, dann müssen Sie selbst dafür sorgen, daß Sie die in beliebiger Reihenfolge eintreffenden Resultate auch den entsprechenden vorher abgesetzten Kommandos zuordnen können.

Eine Möglichkeit ist, die Adresse der ARexxMessage zu speichern und nachher im ResultHook zu vergleichen. Eine andere ist es, die ungenutzten ARG--Felder der RexxMessage zu verwenden um dort eine Kennung anzubringen. In diesem Fall müssen Sie die Funktion von SendRexxCommand selbst durch die Benutzung der beiden elementaren Funktionen CreateRexxCommand und CommandToRexx nachbilden.

#### ARexxDispatch -----

---

In einer Endlosschleife wird nun auf das Eintreffen von Signalen gewartet. Wenn eines vom ARexx--Port kommt, wird die Funktion ARexxDispatch() aufgerufen, die alles weitere erledigt.

Der gleiche Aufbau (Signal -> Dispatcher) eignet sich natürlich auch hervorragend für die Signale von Fenstern, also für die graphische Benutzeroberfläche (siehe oben: GUI--Schicht).

CloseDown  
-----

Der Abschluß wird im Beispiel indirekt (per atexit wurde closedown als Endroutine angemeldet) vorgenommen. Achtung, die Rückkehr von CloseDownARexxHost() kann sich verzögern, wenn noch Replies ausstehen. In dem Fall kann außerdem die in ARexxResultHook eingesetzte Routine noch aufgerufen werden!

## 1.27 ARexxBox.guide/Oberon-2

Hinweise zum Oberon-2--Source  
\*\*\*\*\*

Im folgenden gibt der Autor des Oberon-2--Source (hartmut Goebel) einige Hinweise zu dessen Konzept, Implementation und zu einigen Design-Entscheidungen.

Konzept  
=====

Ziel bei der Erstellung des Oberon-2--Sources war es, dem Programmierer, der ARB benutzen will, volle Typsicherheit und den Komfort von Oberon (z.B. GarbageCollector) zu erhalten. Dies war für mich teilweise eine Herausforderung, da die Parameter der Result-Struktur ja vom Resultat-Template abhängt, zur Compile-Time also nicht allgemein geprüft werden können (1).

Ein weiteres wichtiges Ziel war es, auf die Klassen-Bibliothek PortHandle aufzusetzen, die sich gerade in der Entwicklung befand, als ich den Oberon-2--Source erstellte -- einige Ideen aus ARB sind dort auch eingefloßen.

Aus diesem Ziel folgt zwangsläufig -- aber zwanglos :-) --, daß der Source auch objektorientiert gestaltet wurde. Dies bot sich auch an, da fast alle Prozeduren, die ARB erzeugt, den aktuellen RexxHost als Parameter mitbekommen. Lediglich die Interface-Funktionen wurden nicht als Methoden realisiert, da sie sonst nicht generisch ermittelt und aufgerufen werden könnten.

Resultate  
=====

Während die Parameter von Dos.ReadArgs() geparsed und in der entsprechenden Struktur zur Verfügung gestellt werden (siehe

---

'ReadArgs'), erledigt das für die Resultate ARBRexxHost. Dies bot die Möglichkeit, wenigstens für letztere die volle Oberon-Typsicherheit und denn vollen Oberon-Komfort zu erhalten.

Damit sie dies auch nutzen könne, hier einige nähere Informationen:

Es gilt folgende Typäquivalenz für die Resultate (Typbezeichner sind wieder aufgelöst):

- \* 'FOOBAR' -> 'BasicTypes.DynString;'
- \* 'FOOBAR/M' -> 'POINTER TO ARRAY OF BasicTypes.DynString;'
- \* 'FOOBAR/N' -> 'POINTER TO ARRAY 1 OF LONGINT;'
- \* 'FOOBAR/N/M' -> 'POINTER TO ARRAY OF LONGINT;'

Wie zu sehen ist, werden /M-Resultate als dynamische Arrays spezifiziert. Sie brauchen dieses lediglich mit der benötigten Anzahl Elemente zu allozieren und die Daten dort eintragen. Also z.B.

```
NEW(rx.res.foobar,10);  
für 10 Elemente.
```

Da in Oberon-2 die Anzahl der Elemente leicht zu ermitteln ist, entfällt das abschließende NIL wie in 'C'. Dies ist auch der Grund, weshalb bei /M/N-Resultaten direkt eine 'ARRAY OF LONGINT' benutzt wird und kein 'ARRAY OF POINTER TO LONGINT'. Letzteres macht die Schnittstelle zwar etwas inkonsistent, dafür aber wesentlich komfortabler.

Parameter und einfache numerische Resultate wurden als 'POINTER TO ARRAY 1 OF CHAR' deklariert, das Oberon keine Zeiger auf unstrukturierte Typen zulässt (obwohl AmigaOberon dies tut).

Dokumentation  
=====

Durch das Design und schon durch die Sprache ist die 'C'-Dokumentation für den Oberon-Source nur bedingt geeignet. Dies betrifft insb. die Prozeduren, die in der Klassen-Bibliothek PortHandle und deren Erweiterung ARBRexxHost definiert wurden.

Ich verwies daher auf die Dokumentation zu dieser Bibliothek. Dort sollten alle benötigten Informationen zu finden sein. Falls nicht, bitte ich um eine kurze Mitteilung, damit ich sie ergänzen kann.

hartmut Goebel

----- Footnotes -----

(1) Die generischen Prozeduren in ARBRexxHost (speziell ARBRexxHost.CreatStem) lösen diesen Problem -- wie ich meine -- recht elegant. Ich empfehle durchaus, sie sich einmal näher anzusehen.

## 1.28 ARexxBox.guide/ReadArgs

ReadArgs

\*\*\*\*\*

ReadArgs() benutzt Templates in Textform und schreibt die erkannten Argumente in ein Array von Langwörtern, die vom Programm je nach Typ des korrespondierenden Arguments interpretiert werden müssen.

Beispiel: Ein Template "DATEI,ARG1,FORCE" bedeutet, daß drei Argumente möglich sind. Demzufolge muß man als Zielarray für ReadArgs() ein Array von mindestens drei Langwörtern zur Verfügung stellen.

Template--Syntax

=====

ReadArgs() erlaubt fünf grundlegende Datentypen, die durch entsprechende Switches im Template gekennzeichnet werden:

- String: Default, kein Switch notwendig.
- Integer: '/N'
- Boolean: '/S' für Switch oder '/T' für Toggle
- Liste: zusätzlich '/M' - Listen von Strings und Integers werden unterstützt

Wichtig ist hierbei, daß jedes Argument optional ist, es sei denn, der Switch '/A' (für Always) kommt im Template vor.

Beispiel: 'FILE/A,LINES/N,STDIN/S' bedeutet, daß hier drei Argumente möglich, aber nur das erste benötigt werden. Dabei ist das erste Argument ein String, das zweite ein Integer und das dritte ein Bool.

Weiterhin ist es möglich, für beliebige Argumente die Angabe des Namens als Schlüssel zu erzwingen. Das geschieht mit dem zusätzlichen Switch '/K'. Außerdem sind Aliases möglich, d.h. daß mehrere Namen dasselbe Argument bezeichnen.

Beispiel: 'FILES/M/A,AS=TO/K/A' ist das Template für den Join--Befehl. Hier wird verlangt, daß die Zielfeile vom Anwender konkret gekennzeichnet wird. Für dieses Schlüsselwort kann der Anwender genausogut 'AS' wie 'TO' verwenden. Gültige Eingaben für dieses Template wären z.B.:

- \* 'Dateil Datei2 as ram:Ziel'
- \* 'foo bar test to blafasel'
- \* 'as=ram:test source1 source2 source3 source4'

Hierbei sehen Sie auch, daß ReadArgs() bei Angabe des Schlüsselwortes keine bestimmte Reihenfolge der Argumente mehr benötigt. Außerdem ist die Schreibweise 'KEYWORD DATA' äquivalent zu

```
'KEYWORD=DATA'.
```

Die Forderung nach der Angabe des Schlüsselwortes für bestimmte Parameter ist in vielen Situationen, die Multiple Argumente erlauben, zwingend notwendig, da ReadArgs() sonst diese Argumente nicht von der Liste der Parameter für ein Multi--Argument unterscheiden könnte.

Ergebnisse  
=====

Da die einzelnen Parameter optional sein können werden nicht direkt Objekte des jeweiligen Typs, sondern Zeiger darauf zurückgegeben.

Strings werden als Zeiger auf Character zurückgegeben. Numerische Werte werden als Zeiger auf Langworte zurückgegeben. Nur Bool'sche Werte werden direkt als Langworte zurückgegeben.

Listen werden als Zeiger auf ein Array von Zeigern zurückgegeben. Das Ende einer Liste wird durch einen NULL--Zeiger gekennzeichnet.

Konkret bedeutet das (für C):

```
* 'String' => 'char *'  
* 'Stringarray' => 'char **'  
* 'Integer' => 'long *'  
* 'Integerarray' => 'long **'  
* 'Boolean' => 'long'
```

Für Oberon wurden im Module ARBRexxHost entsprechende Typen vordefiniert. Da sie denen des C-Beispiele entsprechen, wird hier nicht näher darauf eingegangen.

Zur Verdeutlichung ein Beispiel: Dem Template 'FILES/M/A,AS=TO/K/A,FORCE/S,MAX/N,SOUND/T,LINES/N/M' würde folgende Datenstruktur als Rückgabestruktur entsprechen:

```
struct readargs_result  
{  
    char **files;  
    char *to;  
    long force;  
    long *max;  
    long sound;  
    long **lines;  
} rda_result;
```

(Anmerkung: Dieses Template ist nicht einwandfrei in Ordnung und dient nur dazu, die Typgebung zu verdeutlichen - tatsächlich sollte in einem Template nur \*ein\* /M--Switch vorkommen.)

Die einzelnen Felder kann man vor dem ReadArgs--Aufruf mit Defaultwerten füllen, ReadArgs verändert nur die Felder, für die tatsächlich Argumente angegeben wurden. Für Boole'sche Felder gilt

eine andere Regelung: /S--Felder werden immer entsprechend dem Vorhandensein des Keywords gesetzt, Toggle--Felder (/T) drehen den vorher eingesetzten Default--Wert bei Vorhandensein des Keywords um --theoretisch. Praktisch habe ich das noch nie ausprobiert, kann also nicht dafür garantieren 8-).

Bei Strings und Integeren kann man also anhand des primären Zeigers in der obigen Struktur erkennen, ob der Parameter überhaupt spezifiziert wurde, ansonsten ist der Zeiger NULL.

Arrays können beliebig groß werden, daher wird das Ende durch den ersten Arrayeintrag, der NULL ist, gekennzeichnet.

Boole'sche Werte werden immer gesetzt, wenn im Template /S verwendet wurde, und zwar bedeutet die Anwesenheit des Schlüsselwortes im Kommando, daß der Bool--Wert auf 1 gesetzt wird, ansonsten 0. Toggle--Bools (mit /T) drehen den voreingestellten Wert um.

#### Weitere Features

=====

Es gibt noch einen weiteren Switch, /F, der benutzt wird, um einem Argument den gesamten Rest der Kommandozeile ab der Stelle zuzuweisen, selbst wenn andere Schlüsselwörter darin vorkommen. Dieser Switch ist in der Regel unnötig und mit Vorsicht zu genießen.

Parameter, die nicht durch Anführungszeichen explizit vom Anwender als solche gekennzeichnet wurden, werden bei Übereinstimmung mit einem Argument zunächst als Schlüsselwörter betrachtet. So wird die Eingabe "foo bar all qwe" zu einem Template "Dir/M,All/S" den Switch ALL setzen und in DIR das Array "foo", "bar", "qwe" zurückgeben. Wäre in der Eingabe "all" in Anführungszeichen, dann würde der String "all" mit in das Array aufgenommen und der Switch ungesetzt bleiben.

Wie schon oben angedeutet sollte man nicht mehr als ein Array (/M) pro Template verwenden, da ReadArgs sonst Probleme mit der Verteilung der freien Parameter auf die Argumentfelder bekommt.

Wenn nach dem Parsen noch unbesetzte /A--Argumente verbleiben, werden von einem /M--Argument dafür vom Ende des Arrays Strings abgezogen. Ein Beispiel hierfür ist der Copy--Befehl: Das Template "From/A/M,To/A" weist bei der Eingabe "copy file1 file2 file3 destination" das letzte Wort dem TO--Feld zu.

Zu ReadArgs siehe auch 'AutoDocs:dos.doc'.

## 1.29 ARexxBox.guide/Library

Library  
\*\*\*\*\*

TABLE OF CONTENTS

---



```

ARexxBox-ARexxDispatch
ARexxBox-CloseDownARexxHost
ARexxBox-CommandShell
ARexxBox-CommandToRexx
ARexxBox-CreateRexxCommand
ARexxBox-DoShellCommand
ARexxBox-ExpandRXCommand
ARexxBox-FindRXCommand
ARexxBox-FreeRexxCommand
ARexxBox-ReplyRexxCommand
ARexxBox-SendRexxCommand
ARexxBox-SetupARexxHost
ARexxBox-StrDup

```

## 1.30 ARexxBox.guide/ARexxBox-ARexxDispatch

ARexxBox/ARexxDispatch  
 =====

ARexxBox/ARexxDispatch

ARexxBox/ARexxDispatch

### NAME

ARexxDispatch -- get ARexx command from MsgPort and execute it

### SYNOPSIS

```
ARexxDispatch( rexxhost );
```

```
void ARexxDispatch( struct RexxHost * );
```

### FUNCTION

ARexxDispatch fetches and executes all queued commands from the given RexxHost's message port.

If a reply for some previously (with SendRexxCommand()) sent command comes in, the counter variable for still outstanding replies will be decreased by one and the RexxMsg and it's associated memory will be freed by FreeRexxCommand().

In the main program, you should just check for the signal of the host's message port and call ARexxDispatch() without actually getting the message. All work will be done by the dispatcher.

### INPUTS

rexxhost - pointer to an active RexxHost structure with a valid MsgPort

### RESULTS

### SEE ALSO

SendRexxCommand(), SetupARexxHost(), DoShellCommand()

## 1.31 ARexxBox.guide/ARexxBox-CloseDownARexxHost

ARexxBox/CloseDownARexxHost

=====

ARexxBox/CloseDownARexxHost

ARexxBox/CloseDownARexxHost

### NAME

CloseDownARexxHost -- close & free ARexx host

### SYNOPSIS

CloseDownARexxHost ( rexxhost );

void CloseDownARexxHost ( struct RexxHost \* );

### FUNCTION

CloseDownARexxHost() waits until replies for all pending ARexx commands have been received and then closes the ARexx port and frees all memory associated with that RexxHost structure.

All messages sent to a closing host will be replied immediately with an error "Host closing down".

### INPUTS

rexxhost - the RexxHost to close down

### RESULTS

### SEE ALSO

SetupARexxHost(), SendRexxCommand()

## 1.32 ARexxBox.guide/ARexxBox-CommandShell

ARexxBox/CommandShell

=====

ARexxBox/CommandShell

ARexxBox/CommandShell

### NAME

CommandShell -- process Commands from a file

### SYNOPSIS

CommandShell( rexxhost, fhin, fhout, prompt );

void CommandShell( struct RexxHost \*, BPTR, BPTR, char \* );

### FUNCTION

CommandShell() sets the Flag ARB\_HF\_CMDSHELL in the RexxHost's flag field and then processes input from fhin until EOF or the CmdShell flag in the RexxHost being cleared (e.g. by the standard Rexx command "CMDSHELL CLOSE").

The input is read line-wise, with newline as EOL. Each line will be parsed and executed just like a built-in custom ARexx command, exactly like it was called via an ARexx host messageport.

The parsing and execution of each line is done by the function `DoShellCommand()`.

If `fhout` is not `NULL`, the output of the commands will be printed to `fhout`. The output of the commands will NOT be assigned to any variables, as there is no underlying ARexx script program that could hold these variables. Instead, the output will be formatted to be human-readable.

The prompt string (if not `NULL`) will be printed to `fhout` as an input request before reading an input line.

New (ARB 0.99d): The `rexhost` parameter has to point to a valid `RexxHost` structure. This is for identifying which command shell belongs to which window/instance of the main process.

New (ARB 0.99e): To support the "RX" command sending asynchronous messages to ARexx, this function now catches the replies of those messages and frees them using `FreeRexxCommand()`. Messages sent `_to_` this host will be replied immediately with an error "CommandShell Port".

#### INPUTS

`rexhost` - an initialized `RexxHost` structure  
`fhin` - the input `FileHandle` (see `dos.library/Open()`)  
`fhout` - the output `FileHandle` (or `NULL`)  
`prompt` - the prompt string (or `NULL`)

#### RESULTS

#### SEE ALSO

`DoShellCommand()`, `ARexxDispatch()`, `dos.library/Open()`

## 1.33 ARexxBox.guide/ARexxBox-CommandToRexx

ARexxBox/CommandToRexx

=====

ARexxBox/CommandToRexx

ARexxBox/CommandToRexx

#### NAME

`CommandToRexx` -- send a prepared `RexxMsg` to the ARexx server

#### SYNOPSIS

```
msg = CommandToRexx( rexhost, rexxmessage );
```

```
struct RexxMsg *CommandToRexx( struct RexxHost *, struct RexxMsg * );
```

## FUNCTION

CommandToRexx just sends the given RexxMsg to the ARexx server process without changing any fields of the Msg. It will also increment the counter for outstanding replies in the RexxHost structure.

You can use this function together with CreateRexxCommand() to easily create customizable command messages for Rexx.

## INPUTS

rexhost - an initialized RexxHost structure  
 rexxmessage - an initialized ARexx message

## RESULTS

msg - the same as rexxmessage, just for easy further processing

## SEE ALSO

CreateRexxCommand(), SendRexxCommand()

## 1.34 ARexxBox.guide/ARexxBox-CreateRexxCommand

ARexxBox/CreateRexxCommand

=====

ARexxBox/CreateRexxCommand

ARexxBox/CreateRexxCommand

## NAME

CreateRexxCommand -- allocate & initialize rexxmsg for a command

## SYNOPSIS

```
rexxmsg = CreateRexxCommand( rexhost, command, fh );
```

```
struct RexxMsg *CreateRexxCommand( struct RexxHost *, char *, BPTR );
```

## FUNCTION

This function will create a RexxMsg structure for the given RexxHost, create an Argstring from the command string and use that string to initialize the message as a RXCOMM type with RXFF\_RESULT requested.

The file handle will be used for both input and output.

You can use this function to create a standard ARexx command with the additional possibility to set some extra parameters before sending the command to ARexx using CommandToRexx().

## INPUTS

rexhost - an initialized RexxHost structure  
 command - the command string  
 fh - the input/output FileHandle (see dos.library/Open())

## RESULTS

rexxmsg - a pointer to the new RexxMsg structure

SEE ALSO

`CommandToRexx()`, `SendRexxCommand()`, `dos.library/Open()`

## 1.35 ARexxBox.guide/ARexxBox-DoShellCommand

ARexxBox/DoShellCommand

=====

ARexxBox/DoShellCommand

ARexxBox/DoShellCommand

### NAME

`DoShellCommand` -- parse & execute a command line

### SYNOPSIS

`DoShellCommand( rexxhost, commandline, fhout );`

`void DoShellCommand( struct RexxHost *, char *, BPTR );`

### FUNCTION

`DoShellCommand` parses the given string assuming it contains an ARexx-style command line.

New (ARB 0.99e): If normal parsing fails, the external function `ExpandRXCommand()` will be called to expand any macros. If the expansion fails or the expanded command can't be recognized either, an error will be returned.

If no errors occur during parsing, it tries to execute the command with the given arguments. The results of the command's execution will be printed in a human-readable format to `fhout` if `fhout` is not `NULL`.

If errors occur, `DoShellCommand` prints a string describing the error to `fhout` (if not `NULL`).

New (ARB 0.99d): The `rexxhost` parameter has to point to a valid `RexxHost` structure. This is for identifying which command shell belongs to which window/instance of the main process.

### INPUTS

`rexxhost` - an initialized `RexxHost` structure  
`commandline` - the string to be parsed & executed  
`fhout` - the output `FileHandle` (or `NULL`)

### RESULTS

none

SEE ALSO

`CommandShell()`, `ExpandRXCommand()`, `<dos/dos.h>`

## 1.36 ARexxBox.guide/ARexxBox-ExpandRXCommand

ARexxBox/ExpandRXCommand

=====

ARexxBox/ExpandRXCommand

ARexxBox/ExpandRXCommand

### NAME

ExpandRXCommand -- expand macros and/or aliases (V0.99e)

### SYNOPSIS

```
newcommand = ExpandRXCommand( rexxhost, oldcommand )
```

```
char *ExpandRXCommand( struct REXXHost *, char * );
```

### FUNCTION

This is an 'external' function you should provide if you want to have command aliases or the like. The minimal version of this function is just a return(NULL) as generated in the rxif module.

ExpandRXCommand() will be called by the parser if it doesn't know how to interpret a command string. Expansion could now for example be a look up in the host's macro table.

Any strings returned by this function have to be allocated explicitly using the standard C memory functions. The calling parser will free() them.

### INPUTS

rexxhost - the REXXHost we are working on

oldcommand - the commandline the parser doesn't know

### RESULTS

newcommand - an explicitly allocated memory area containing the expanded command (or NULL)

### SEE ALSO

DoShellCommand(), ARexxDispatch()

## 1.37 ARexxBox.guide/ARexxBox-FindRXCommand

ARexxBox/FindRXCommand

=====

ARexxBox/FindRXCommand

ARexxBox/FindRXCommand

### NAME

FindRXCommand -- search the ARexxBox command table (V0.99e)

### SYNOPSIS

```
rxscmd = FindRXCommand( command )
```

```
struct rxs_command *FindRXCommand( char * );
```

#### FUNCTION

This function returns a pointer to the given command's entry in the ARexxBox-generated command table. It exists to support those functions working on/with commands, like HELP or ENABLE/DISABLE.

This function does no macro expansion. The comparisons are case independant so you don't have to convert your input to upper case beforehand. As this is exactly the routine used by the parser to find a command, it will handle abbreviations.

#### INPUTS

command - the command name to search for

#### RESULTS

rxscmd - the rxs\_command structure of that command  
(or NULL if command not found)

#### SEE ALSO

## 1.38 ARexxBox.guide/ARexxBox-FreeRexxCommand

ARexxBox/FreeRexxCommand

=====

ARexxBox/FreeRexxCommand

ARexxBox/FreeRexxCommand

#### NAME

FreeRexxCommand -- free the associated memory of a RexxMsg

#### SYNOPSIS

```
FreeRexxCommand( rexxmessage );
```

```
void FreeRexxCommand( struct RexxMsg * );
```

#### FUNCTION

This is basically a PD ARexx routine provided by William S. Hawes.

It frees all memory associated with a particular (previously sent) ARexx message structure. It will also close any stdin/stdout channels associated to that Rexx message.

You normally shouldn't have to bother with this one because the dispatcher will call it for you.

#### INPUTS

rexxmsg - the rexx message to free

#### RESULTS

#### SEE ALSO

SendRexxCommand()

## 1.39 ARexxBox.guide/ARexxBox-ReplyRexxCommand

ARexxBox/ReplyRexxCommand

=====

ARexxBox/ReplyRexxCommand

ARexxBox/ReplyRexxCommand

### NAME

ReplyRexxCommand -- reply a rexx message from rexxmast

### SYNOPSIS

ReplyRexxCommand( rexxmsg, primary, secondary, result );

void ReplyRexxCommand( struct RexxMsg \*, long, long, char \* );

### FUNCTION

This is a PD ARexx routine provided by William S. Hawes.

It replies a given rexx message to the rexx master process, filling in a primary and a secondary return code plus optionally a supplied result string.

The result string will only be converted to an ARexx string, if the primary return code equals 0, and will then destroy the contents of the secondary return code. So you provide either primary and secondary return codes or a result string.

You normally shouldn't have to call this function! It is only mentioned here, because it is not part of the ARexxBox routines, but part of the original ARexx distribution by William S. Hawes.

New (ARB V0.99d): Now creates an ARexx variable "RC2" for the secondary return code. If primary is positive, secondary is interpreted as a long, if primary is negative, secondary is interpreted as a char \*. RC will become positive in any case.

RC2 will only be assigned if the ARexx RESULT flag is set.

### INPUTS

rexxmsg - the message structure to reply

primary - the primary return code (rc) ( >0 <0 )

secondary - the secondary return code (rc2) (long or char \*)

result - the result string

### RESULTS

### SEE ALSO

SendRexxCommand(), FreeRexxCommand()



## 1.40 ARexxBox.guide/ARexxBox-SendRexxCommand

ARexxBox/SendRexxCommand

=====

ARexxBox/SendRexxCommand

ARexxBox/SendRexxCommand

### NAME

SendRexxCommand -- invoke rexx command script

### SYNOPSIS

```
rexmsg = SendRexxCommand( rexxhost, command, filehandle )
```

```
struct RexxMsg *SendRexxCommand( struct RexxHost *, char *, BPTR );
```

### FUNCTION

This is basically a PD ARexx routine provided by William S. Hawes.

This function sends the given command string to the ARexx master process for execution as an ARexx command. The command string contains the file name of the ARexx script to be started. If the filehandle is not NULL, it will be used as stdin and stdout for the Rexx script. If it is NULL, the Rexx program will use stdin/stdout of the calling process.

If necessary, the default extension (defined in the generated header file under the name REXX\_EXTENSION) will be added to the file name.

Messages sent using this function will be replied to by the ARexx master process as soon as the execution of the command script stops. The application MUST NOT close it's messageport before all replies have been received! To simplify things, ARexxBox does this book-keeping for you. CloseDownARexxHost() will wait for all missing replies to arrive before closing down the messageport.

The dispatcher will automatically detect any replies, count them and do a FreeRexxCommand() for each reply, so you don't have to bother with this either.

Internally, this function is implemented using the two more atomic functions CreateRexxCommand() and CommandToRexx().

### INPUTS

rexxhost - the RexxHost to be used to send the command  
 command - the file name of the ARexx script  
 filehandle - Filehandle for stdin/stdout or NULL

### RESULTS

rexmsg - the sent rexx message structure (for comparisons)

### SEE ALSO

```
FreeRexxCommand(), CloseDownARexxHost(), ARexxDispatch(),
CreateRexxCommand(), CommandToRexx()
```

## 1.41 ARexxBox.guide/ARexxBox-SetupARexxHost

ARexxBox/SetupARexxHost  
=====

ARexxBox/SetupARexxHost

ARexxBox/SetupARexxHost

### NAME

SetupARexxHost -- initialize and open an ARexx host

### SYNOPSIS

```
rexhost = SetupARexxHost( basename );

struct RexxHost *SetupARexxHost( char * );
```

### FUNCTION

This function allocates and initializes a RexxHost structure. It opens a public message port under the given basename. If no basename (NULL) was specified, the default basename as entered in the ARexxBox window will be used instead.

Anyway, if a public port of that name already exists, SetupARexxHost() will start adding numbers to the name until a unique name is found. So if for example the basename is "myhost" and there is already a port of that name in the system, the name will be changed to "myhost.1" (then to "myhost.2" and so on).

The actual name will be copied to the portname field of the RexxHost structure. It is a good idea to tell the user about the actual port name of the new host.

### INPUTS

basename - the messageport basename or NULL

### RESULTS

rexhost - the initialized RexxHost structure, ready to go. Pass this pointer to CloseDownARexxHost(), ARexxDispatch() and SendRexxCommand().

### SEE ALSO

CloseDownARexxHost(), ARexxDispatch(), SendRexxCommand()

## 1.42 ARexxBox.guide/ARexxBox-StrDup

ARexxBox/StrDup

=====

ARexxBox/StrDup

ARexxBox/StrDup

## NAME

StrDup -- duplicate a string

## SYNOPSIS

copy = StrDup( origin );

char \*StrDup( char \* );

## FUNCTION

This routine will do exactly the same as its standard C lib counterpart strdup(), but use the exec function AllocVec() for allocating the needed memory.

(So YOU are responsible for freeing the copy!)

## INPUTS

origin - the original string

## RESULTS

copy - the copy of the original string

## SEE ALSO

strdup(), exec.library/AllocVec(), exec.library/FreeVec()

## 1.43 ARexxBox.guide/Dateifformat

Dateifformat

\*\*\*\*\*

In diesem Kapitel wird das Dateifformat der ARexxBox erklärt, und was man dabei beachten sollte, um eine externe Nutzung der ARB--Dateien zu ermöglichen. Sie können somit z.B. selbst weitere Sourcegeneratoren schreiben.

Das ARB--Datenformat ist ein erweiterbares ASCII--Format (Zeilentrenner ist LF). Neben den offensichtlichen Daten (Syntax der Befehle, Argumente und Resultate) enthält eine ARB--Datei zusätzlich allgemeine und befehlsspezifische Tags.

ARB--Dateien enthalten zwecks Identifizierung als erste Zeile den String 'ARB'. Ab Zeile zwei steht der erste, globale Tag--Block. Jede Tagzeile fängt mit einem Dollarzeichen an, dem ohne Leerraum die Tag--ID (Typ long) folgt. Der Tag--ID folgen dann -- durch Spaces getrennt -- entsprechend der Tag--ID noch weitere alphanumerische Parameter.

Bislang existiert nur eine globale Tag--ID:

- \* ID 1, zwei Parameter:
  1. File--ID (unsigned long): Diese ID wird von der ARB verwendet, um zu Prüfen ob ein bestehender Source zur Datei paßt.
  2. Next-Cmd-ID (unsigned long): Die ID für den nächsten neuen Befehl (\*nicht\* gleich der Anzahl der Befehle!).

Die erste Zeile nach diesem Tagblock enthält den MsgPort--Basisnamen. Die nächste Zeile enthält den Status des CommandShell--Gadgets (1 für angewählt). Dann folgen die Befehle. Jeder Befehl hat die folgende Struktur:

- \* Befehlsname
- \* Tagblock (erstes Zeichen = '\$', gefolgt von der Tag--ID wie oben)
- \* Argumente (eines pro Zeile)
- \* Trennzeile (enthält nur ein Minuszeichen)
- \* Resultate (eines pro Zeile)
- \* Trennzeile

Bei den Befehlen existieren bislang zwei Tag--IDs:

- \* ID 1, zwei Parameter:
  1. Befehls--ID (unsigned long): Wird zur Identifizierung des Befehls in einem vorhandenen Source verwendet.
  2. Status (char): Kann die Werte 'N' für 'New', 'C' für 'Changed' oder 'O' für 'Old' annehmen.
- \* ID 2, ein Parameter:
  1. ExtStatus (char): Werte wie Status (Dokumentationsstatus).

Bitte beim Einlesen darauf achten, daß in zukünftigen Versionen der ARexxBox die Tagblöcke erweitert werden können. Die Leseroutine für die Tags sollte mit einer beliebigen Anzahl von Tagzeilen zurechtkommen und unbekannte Tags zumindest ignorieren.

## 1.44 ARexxBox.guide/Geschichte

### Geschichte

\*\*\*\*\*

History of ARexxBox Releases:

V0.99  
erster Beta-Release

V0.99a  
FIXED: Argument- und Resultat-Liste konnte mit

ungültigem Kommando arbeiten -> Enforcer-Hit/Absturz.  
(Report: RALF\_KAISER@AWORLD)

## V0.99b

ENHANCED: GadToolsBox-Source fixed um den System Default Font statt des Screenfonts für Layout und Gadgets zu benutzen.  
(Report: SYSOP@INSIDER [Garry Glendown])

## V0.99c

FIXED: Für Argumente wurde das Gleichheitszeichen nicht zugelassen.

FIXED: Nach Änderung eines Argumentes oder Resultates verbleibt die Anzeige auf diesem Element statt (wie bisher) das letzte Listenelement anzuzeigen.

FIXED: In den Font-Routinen war noch ein Fehler.  
(Report: F.J.Reichert [F\_J\_REICHERT@SAARAG])

## V0.99d

FIXED: Alle Pointer-Conversions bereinigt. Der Code wird jetzt selbst bei Compiler 'superscharf' völlig ohne Warnings durchgezogen.  
Pragma-#includes und \_toupper() nun auch für SAS/C drin.  
(Report: W\_KUETING@HSP)

--- V0.99d wurde nicht verteilt! ---

## V0.99e

FIXED: Wenn \_kein\_ Resultatfeld belegt war, wurde ein "out of memory"-Fehler erzeugt.

ENHANCED: ReplyRexxCmd() erzeugt nun die Variable RC2, sie enthält den "Secondary Returncode", der in jeder rxif-Struktur unter dem Namen rc2 steht. Diese Variable kann z.B. benutzt werden, um detaillierte Fehlercodes an das Rexxprogramm zurückzugeben.

Dieses Feature ist eine Erweiterung der Style-Guide-Richtlinien, m.E. eine sinnvolle. Wenn jemand anderer Ansicht ist, bitte Mail an mich!

RC2 wird nur erzeugt, wenn a) Resultate erwünscht sind und b) RC != 0 ist. Das Feld rc2 kann sowohl einen Fehlercode (long, default) als auch eine Fehlerbeschreibung (char \*) enthalten. Unterschieden wird anhand des Vorzeichens von rc, d.h. rc < 0 <=> rc2 long bzw. rc > 0 <=> rc2 char \*. Negatives rc wird vor der Rückgabe nach positiv gewandelt.

Beispiel für FehlerCODE:

```
rd->rc = 10;  
rd->rc2 = ERROR_OBJECT_NOT_FOUND;
```

Beispiel für FehlerSTRING:

```
rd->rc = -10;
```

---

```
rd->rc2 = (long) "Objekt nicht gefunden!";
```

Die Standard-Fehlerstrings sind `_nicht_` lokalisiert. Wenn sich C= mal bequemt, mir Infos und Werkzeuge dafür zu schicken, werde ich das natürlich nachholen.

ENHANCED: `SendRexxCommand()` nimmt jetzt zur besseren Unterstützung des Standardbefehls "RX" (zum Starten von ARexx-Programmen) einen zusätzlichen Parameter, ein `FileHandle` (BPTR!), der als `Stdin/Stdout` für das Script eingesetzt wird. `FreeRexxCommand()` übernimmt das Schließen des Kanals.

Außerdem liefert `SendRexxCmd()` nun die Adresse der abgeschickten `RexxMsg` zurück (oder `NULL` wenn Fehler).

ENHANCED: Die `RexxHost`-Struktur hat jetzt ein zusätzliches Feld namens "APTR userdata". Es kann z.B. benutzt werden, um in einen `RexxHost` einen Zeiger auf eine eigene Projekt-Verwaltungs-Struktur einzuhängen o.ä., um den Host eindeutig einem offenen Projekt zuzuordnen.

ENHANCED: Jeder Befehl hat jetzt ein globales "enabled"-Flag, wenn dies gelöscht ist, verweigert der Dispatcher die Ausführung des Befehls. Dieses Flag sollte mit den Standard-ARexx-Befehlen (Style Guide) `ENABLE` und `DISABLE` beeinflussbar sein (siehe `rxif/*.c`).

CHANGED: `CommandShells` benötigen jetzt einen eigenen `RexxHost` auf dem sie alleine arbeiten können. D.h. der dafür bereitgestellte Host DARF NICHT gleichzeitig als normaler ARexx-Port benutzt werden. Eine abwechselnde Benutzung als ARexx- und Cmd-Port ist denkbar, aber nicht empfehlenswert...

ENHANCED: Damit einhergehend bekommt nun jede `rxif`-Funktion den `RexxHost` des ARexx-Ports bzw. der `CmdShell` als (ersten) Parameter (`host`) übergeben. Dadurch kann die Funktion z.B. entscheiden, zu welchem Projekt der Befehl ausgeführt werden soll.

ENHANCED: Ebenfalls damit einhergehend existiert jetzt zur Unterstützung des Standardbefehls `CMDSHELL` in jedem `RexxHost` ein Flag (`ARB_HF_CMDSHELL`), das anzeigt, ob der Host gerade eine `CommandShell` fährt. Wenn dieses Flag (durch `CMDSHELL CLOSE`) gelöscht wird, terminiert die ARexx-Box die `CommandShell`.

FIXED: Aus Spaces oder nur aus Optionen bestehende Argument- bzw. Resultatfeldnamen werden nicht mehr akzeptiert.

CHANGED: Da einige `rxif`-Funktionen darauf zugreifen müssen ist "struct `rxs_command`" und "`rxs_commandlist`" nun im Headerfile definiert. Für den Zugriff auf die Liste (finden eines Kommandos) ist eine neue Funktion da:

```
struct rxs_command *FindRXCommand( char *name );
```

FIXED: Argumente mit '=' wurden nicht korrekt in C-Variablennamen umgesetzt. Nun wird der letzte Alias als Variablenname benutzt.

ENHANCED: Es gibt jetzt eine eigene Handlerfunktion für den Parser unbekannte Kommandos:

```
char *ExpandRXCommand( struct RexxHost *host,  
                      char *commandline );
```

Diese wird aufgerufen, bevor der Parser "Not implemented" als Fehler ausgibt. Wenn die Funktion NULL zurückgibt, wird der Fehler ausgegeben, ansonsten versucht der Parser, den zurückgegebenen String zu analysieren. Diese Funktion kann z.B. die Expansion von Aliases oder die Bearbeitung von Non-Standard-Kommandos übernehmen.

Achtung! Es wird erwartet, daß Expand() eigens Speicher für den Resultatstring allokiert, dieser wird von den ARB-Routinen dann freigegeben!

ENHANCED: Die Checks beim Generieren der Source-Module sind jetzt sicherer. Es wird nun auch auf Existenz des C- und H-Moduls geprüft.

NOTE: Als ich die vom Style Guide vorgeschlagenen Standardkommandos eingab, sah ich, daß die Argumente "VAR" und "STEM" an einer Stelle auch dazu benutzt werden, die Eingabe der Funktion von den damit benannten Variablen zu holen. Ich weiß nicht, ob das so eine glückliche Wahl ist, da man auf diese Art&Weise keine Resultate mehr haben kann. Besser wäre eine andere Benennung, z.B. "FROMVAR" und "FROMSTEM". Vorschläge?

Die ARB kümmert sich wenig um die bereits eingegebenen Argumente, wenn Resultatfelder existieren, werden VAR und STEM immer in das Template aufgenommen.

Allgemein sind einige Definitionen an der Stelle im Style Guide sehr ungenau oder unzureichend. Ich habe die Definition (sinnvoll) erweitert. Wer mit meinen Änderungen nicht einverstanden ist, soll sich bitte melden.

NOTE: Beim übernehmen alter Interface-Routinen daran denken, daß ein neuer Parameter dazu gekommen ist!

FIXED: Numerische Resultate funktionieren jetzt.

V0.99f

ENHANCED: Es werden zwei FileRequester verwendet, einer für Binaries und einer für Sourcen. Die Patterns bleiben nun erhalten und sind standardmässig mit #? statt \*.

(Report: Stefan Zeiger)

---

FIXED: Window-Topedge ist nun Fontsensitiv.  
(Report: Stefan Zeiger)

#### V0.99g

FIXED: Beim Parsen der Argumente wird nun der von  
ReadArgs() erzeugte Fehlercode als rc2 benutzt.

CHANGED: Den Templates wird nun vor dem Parsen kein  
\n mehr angehängt (unnötig).

CHANGED: Die ARexxBox benutzt nun den ASL-Filerequester.  
(Wunsch: Garry Glendown [Sysop@Insider])

#### V1.00

ENHANCED: Source wird nun auch vom GCC fehlerfrei  
und warnungsfrei übersetzt - naja, fast warnungsfrei.  
Die verbleibenden Warnungen können aber getrost  
ignoriert werden.

CHANGED: FindRXCommand() akzeptiert nur noch echte  
Abkürzungen der vorhandenen Befehle.

--- ERSTER ÖFFENTLICHER RELEASE ---

#### V1.01

ENHANCED: FindRXCommand() benutzt jetzt binäre Suche.  
(also Komplexität  $O(\log n)$  statt  $O(n)$ )

#### V1.02

ADDED: Clipboard-Unterstützung

FIXED: CreateVAR() prüfte eine Allokation nicht  
(Report: Rüdiger Dreier)

CHANGED: SendRexxCommand() ist nun aufgeteilt in zwei Routinen,  
CreateRexxCommand() und CommandToRexx(). Somit kann man die  
erstellte Msg bei Bedarf vor dem Versand noch verändern.

ENHANCED: Unbekannte Kommandos werden nun zusätzlich noch an ARexx  
geschickt, bevor sie als "unbekannt" eingestuft werden. So können  
externe ARexx-Programme völlig transparent aufgerufen werden.  
(Vorschlag: Rüdiger Dreier)

ENHANCED: DoRXCommand() und DoShellCommand() bei der  
Rückgabe ein wenig aufpoliert.  
(Report: Hartmut Goebel)

FIXED: Der Dispatcher prüft nun, ob es sich bei den ankommenden Msgs ↔  
wirklich  
um ARexx-Kommandos handelt.  
(Report: Hartmut Goebel)

FIXED: In free\_stemlist() war ein potentieller FreeMem-Bug.

CHANGED: Statt (m|c)alloc und free werden jetzt AllocVec und FreeVec  
benutzt. Damit einhergehend ist eine neue Funktion, ein Replacement für



strdup(), StrDup() (wie auch sonst) vorhanden.  
char \*StrDup( char \*string )  
(Vorschlag: Rüdiger Dreier)

FIXED: Der CloseDown konnte laufende Skripte, die auf dem Port arbeiteten, hängen lassen.  
(Report: Rüdiger Dreier)

#### V1.03

ENHANCED: Neuer Parameter für die ARB: FONT=<name>/<size>, spezifiziert den Font, den die ARB benutzen soll. Default ist jetzt wieder (Style-Guide-konform) der ScreenFont.  
(Vorschlag: Timothy J. Aston)

CHANGED: Wenn ein leerer Portname angegeben wird, wird der Defaultname benutzt.  
(Vorschlag: Hartmut "Essich" Goebel)

CHANGED: VAR- und STEM-Parameter werden nach Großschreibung gewandelt, da SetRexxVar() dies anscheinend nicht selbst durchführt.  
(Rexx-Konvention)

#### V1.04

FIXED: Pfad des Projektfiles wird nun beibehalten wenn man zwischendurch ein File per "Merge" hereinholt.  
(Report: Stefan Reisner)

FIXED: ARG0 wird nun überall auf (char \*) gecastet.  
(Report: Marc Schröer)

ENHANCED: Das Headerfile hat jetzt einen eigenen #define.  
(Marc Schröer)

#### V1.05

ENHANCED: Sourceerzeugung nun intelligenter, alte Interfacerroutinen werden anhand spezieller Kommentare im Source erkannt und entweder übernommen oder in ein Lagerfile kopiert.  
(Vorschlag: Alle; ID-Idee: Christoph Teuber)

CHANGED: Dateiformat geändert um die neue Sourceerzeugung zu unterstützen. Die Box kann das alte Format weiterhin lesen, das erspart mir den Konverter (gell, Olaf? ;-).

ENHANCED: SetupARexxHost() bekommt nun als zweiten Parameter einen optional vorher vom User allokierten MsgPort. So kann man sich vorher einen MsgPort für speziellen Bedarf (z.B. spezielles Signal) schneiden.  
(Vorschlag: Stefan Reisner)

#### V1.10

ENHANCED: Die ARexxBox kann nun auch Oberon-Code erzeugen. Der Oberon-Code stammt komplett aus der außerordentlich fähigen Feder von Hartmut "Essich" Goebel. Fragen und Vorschläge zum Oberon-Teil bitte an ihn, da ich mit Oberon wenig anfangen kann (und will 8-).

ENHANCED: Result-Hook eingebaut, wenn man der Variablen

ARexxResultHook die Adresse einer Funktion zuweist, dann wird diese für alle Replies zu selbst abgeschickten Msgs aufgerufen.

FIXED: FreeRexxCommand() schließt stdin und stdout nicht mehr.  
Vielleicht trägt ja jemand aus Versehen mal diese ein (nicht NULL).

V1.11

FIXED: Letzte Oberon-Änderungen eingebaut.

----- Release 2 -----

V1.12

FIXED: HELP-Befehl erzeugte Enforcer-Hits (strlen(0)).  
(Report: Klaas Hermanns)

ENHANCED: Tag #2 für Kommandos eingeführt: Externes Status-Flag, "Neu"-Status dieses Flags wird nur von externen Programmen (spez. Dokumentationshilfen wie ARB2TeXinfo von Albert Weinert) zurückgesetzt.  
(Vorschlag: Albert Weinert)

FIXED: Die Kommandoliste wurde in der Reihenfolge der IDs anstatt in alphabetischer Reihenfolge generiert.  
(Report: Klaas Hermanns)

ENHANCED: Statt binärer Suche (für die Kommandos) erzeugt die Box nun doch einen endlichen Suchautomaten. Die binäre Suche hatte den grundlegenden Designfehler, daß die Sortierung nach Länge mit Abkürzungen kollidierte.  
(Report des Fehlers: Klaas Hermanns)

FIXED: Für SAS wird toupper() #undefined, außerdem in der CommandList die richtigen Casts für die Funktionen.  
(Report: Klaas Hermanns)

FIXED: Die MIN- und MAX-Resultate von Misc.arb/GETATTR müssen natürlich Numerisch sein.  
(Report: Klaas Hermanns)

FIXED: "Merge" baute Mist bei bestehenden Kommandos.

CHANGED: Die externen Libs werden nun mit ihren erweiterten Typen deklariert (C).  
(Wunsch: Klaas Hermanns)

CHANGED: In arb/advanced.arb habe ich bei den Kommandos REQUESTNUMBER und REQUESTSTRING jeweils das Argument "DEFAULT" in "DEFAULTNUM" bzw. "DEFAULTSTR" geändert, um Konflikte mit den Schlüsselwörtern in C zu vermeiden.  
(Report: Klaas Hermanns)

ENHANCED: Die RXIF-Funktionen bekommen nun einen weiteren Parameter, struct RexxMsg \*rexxmsg, welcher bei einem Aufruf von REXX aus den Zeiger auf die RexxMessage enthält (was sonst). Dies ermöglicht es, z.B. GetRexxVar() im RXIF-Code zu benutzen. Bei Aufruf von einer CommandShell enthält der Parameter den Wert NULL, ist somit auch als

Unterscheidungsmerkmal ARexx <-> CommandShell benutzbar.  
(Report: Klaas Hermanns)

FIXED: Für den GCC wird nun toupper() als Funktion definiert. Das  
Makro führte wegen mehrfacher Auswertung des Ausdrucks zu Fehlern.

FIXED: Die CommandShell hatte noch einen Bug beim Parsen der Parameter  
(ReadArgs-Puffer wurde nicht gelöscht).

----- Release 3 -----

## 1.45 ARexxBox.guide/Danksagungen

Danksagungen

\*\*\*\*\*

Ich möchte mich hiermit besonders bedanken bei:

- \* Der Amiga--Crew bei Commodore, für den Amiga und AmigaOS Release 2.
- \* William S. Hawes für seine hervorragende REXX--Portierung.
- \* Jan van den Baard, für seine exzellente GadToolsBox, die mir als Anregung diente und mir auch bei der Erstellung der Oberfläche gute Dienste geleistet hat.
- \* Nico Francois, für seine ReqTools.Library mit ihren wirklich Programmierer-- und Anwenderfreundlichen Requestern.
- \* hartmut Goebel für den Oberon-2--Source.
- \* Meinen aktiven Betatestern und allen, die mir Verbesserungsvorschläge geschickt haben. (Siehe History)
- \* Christoph Teuber und Oliver Wagner für das AWorld Mailbox System.

## 1.46 ARexxBox.guide/Index

Index

\*\*\*\*\*

Adresse  
amiga.lib  
amiga.lib  
ANSI--C  
ARB--Eingabe  
ARexxBox/ARexxDispatch

Adresse des Autors  
Systemanforderungen  
Systemanforderungen  
Systemanforderungen  
Eingabe  
ARexxBox-ARexxDispatch

ARexxBox/CloseDownARexxHost  
ARexxBox/CommandShell  
ARexxBox/CommandToRexx  
ARexxBox/CreateRexxCommand  
ARexxBox/DoShellCommand  
ARexxBox/ExpandRXCommand  
ARexxBox/FindRXCommand  
ARexxBox/FreeRexxCommand  
ARexxBox/ReplyRexxCommand  
ARexxBox/SendRexxCommand  
ARexxBox/SetupARexxHost  
ARexxBox/StrDup  
Argumente  
ATT: Interface changed!  
Beispiel test  
Bug reports  
CommandShell  
Copy  
Copyright  
Cut  
Danksagungen  
Dateiformat  
Datenstruktur  
Designrichtlinien  
Distribution  
E-Mail  
Einbindung des ARB--Srcs  
Erase  
Fähigkeiten in Stichpunkten  
Fehler--Rückgaben  
FreeWare  
Generate Source  
Geschichte  
History  
Interfacefunktionen  
InterNet Adresse  
Kommentare  
Konzept  
Konzept: Closedown  
Konzept: Expandierung  
Konzept: Initialisierung  
Konzept: Kommandos an ARexx  
Konzept: Kommandos von ARexx  
Kritik  
Lokaler Speicher  
Mark Range  
Merge in  
MsgPort Basename  
Oberon-2  
OLink  
Paste  
Postadresse  
Print  
RC  
RC2  
ReadArgs  
ReadArgs

ARexxBox-CloseDownARexxHost  
ARexxBox-CommandShell  
ARexxBox-CommandToRexx  
ARexxBox-CreateRexxCommand  
ARexxBox-DoShellCommand  
ARexxBox-ExpandRXCommand  
ARexxBox-FindRXCommand  
ARexxBox-FreeRexxCommand  
ARexxBox-ReplyRexxCommand  
ARexxBox-SendRexxCommand  
ARexxBox-SetupARexxHost  
ARexxBox-StrDup  
Argumente&Resultate  
Generate Source  
Einbindung  
Adresse des Autors  
CommandShell  
Clipboard  
Copyright  
Clipboard  
Danksagungen  
Dateiformat  
Interfacefunktionen  
Einbindung  
Copyright  
Adresse des Autors  
Einbindung  
Clipboard  
Einleitung  
Fehler  
Wichtig  
Generate Source  
Geschichte  
Geschichte  
Interfacefunktionen  
Adresse des Autors  
Dateien  
Konzept  
CloseDown  
Expandierung von Befehlen  
Initialisierung  
Kommandos an ARexx  
Kommandos von ARexx  
Adresse des Autors  
Interfacefunktionen  
Clipboard  
Merge in  
MsgPort Basename  
Oberon-2  
Systemanforderungen  
Clipboard  
Adresse des Autors  
Print  
Fehler  
Fehler  
Argumente&Resultate  
ReadArgs

ReadArgs: Ergebnisse	ReadArgs
ReadArgs: Template--Syntax	ReadArgs
ReadArgs: Weitere Features	ReadArgs
Rechtliche Dinge	Copyright
regtools.library	Systemanforderungen
RESULT	Argumente&Resultate
Resultate	Argumente&Resultate
rexxvars.o	Systemanforderungen
RVI	Systemanforderungen
rxif--Beispiele	Interfacefunktionen
Schichtenmodell	Einbindung
Shortcuts	Eingabe
Source: ARB--Kommentare	Dateien
Source: C	Dateien
Source: Oberon	Dateien
Spenden	Adresse des Autors
static	Interfacefunktionen
STEM	Argumente&Resultate
Transferstruktur	Interfacefunktionen
Typen	Argumente&Resultate
up & do	Eingabe
VAR	Argumente&Resultate
Wichtige Bemerkungen	Wichtig