ARexxBox

ARexx Interface Designer

Michael Balzer

Internet: balzer@heike.informatik.uni-dortmund.de
Zerberus: M.BALZER@AWORLD.ZER

Wildermuthstraße 18
W-5828 Ennepetal 14
GERMANY

20. Juni 1992

Inhaltsverzeichnis

1 Rechtliches

1.1 Copyright

ARexxBox ist copyright ©1992 by Michael Balzer. Alle Rechte vorbehalten. ARexxBox ist Public Domain Software, das heißt in diesem Fall, daß Gebühren, Bezahlung, Lizenzvergabe o.ä. weder für die ARexxBox selbst, noch für den von ihr erzeugten Code notwendig sind.

Gegen Spenden jeglicher Art habe ich (als armer Student :-) natürlich nichts einzuwenden.

Wenn Sie die ARexxBox oder die von ihr erzeugten Routinen und Verfahren benutzen, um ein ARexx–Interface für ein Programm (kommerziell, Shareware oder Public Domain) zu erstellen, dann muß dies in der Dokumentation des Programms oder in dessen "About"–Requester erwähnt werden.

Außerdem würde ich es sehr begrüßen, wenn Sie mir eine Notiz über die Verwendung der ARexxBox, die Art des Programms und den Befehlsumfang des ARexx–Interfaces zukommen lassen könnten (vorzugsweise per EMail).

1.2 Verbreitung

ARexxBox ist Public Domain, das heißt sie darf frei kopiert und weitergegeben werden solange alle Dateien des Originalarchives unverändert zusammen bleiben. Die Verbreitung auf elektronischen Netzen ist erwünscht.

Mit Ausnahme der Verteilung im Rahmen der Commodore Native Developer's Upgrades und deren elektronischer Äquivalente darf die ARexxBox *nicht* kommerziell vermarktet werden.

Unter den deutschen PD–Autoren gibt es in letzter Zeit Bemühungen, eine für alle PD–Händler verbindliche Weisung zum Vertrieb von PD–Software zu erstellen. Diese Weisung wird sich vorraussichtlich sowohl auf die Preisgestaltung einzelner Disks als auch auf die Preisgebung ganzer Sammlungen wie den Fish-CD-ROMs erstrecken.

Da ich diese Bemühungen (obwohl ich sie teilweise als ungerechtfertigt betrachte) nicht unterlaufen will, unterstelle ich hiermit die ARexxBox in allen auf Urheberrechtlich geschützte PD–Software anwendbaren Punkten dieser (noch nicht existenten) Weisung.

Vorraussichtlich wird durch die Weisung der Vertrieb auf Disketten zu Preisen von mehr als fünf DM und der Verkauf als Teil eines CD-ROMs für mehr als 30 (dreißig) DM illegal, so daß alle betroffenen Händler schon jetzt von einem Vertrieb der ARexxBox Abstand nehmen sollten.

2 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 ;-)[1].

[1] bei TEX bekommt der Smily immer so ein breites Grinsen, finden Sie nicht auch? :-)

2.1 Warum ARexxBox?

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.

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 C–Source: Ein Modul mit den nötigen Grundfunktionen und ein Modul mit den Interfaceroutinen, in die nur noch der spezifische Code einzusetzen ist.
- 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 so viele ARexx–Ports und CommandShells öffnen, wie das Amiga–Messagepassing–System MessagePorts zuläßt.
- Die vom Style Guide vorgeschlagenen Standard-Kommandos habe ich schon für Sie abgetippt. Beispielcode für einige dieser Befehle liegt auch bei.

2.2 Systemanforderungen

Die einzige Bedingung zur Lauffähigkeit sowohl der ARexxBox als auch des erzeugten Codes ist AmigaOS 2.x, d.h. mindestens Kickstart und Workbench 2.04. An die Hardware werden so gut wie keine Anforderungen gestellt, 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...

Sie sollten sich außerdem eine Kopie von Nico François' reqtools.library besorgen, wenn Sie noch keine haben.

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.

Die Version 37.32 ist im Native Developer's Upgrade V37.4 (11/91) enthalten, vielleicht wird sie auch schon bei manchen Compilern mitgeliefert.

3 Bedienung des Programms

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

3.1 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 direkt nach der Eingabe oder 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 Templateoptionen werden im ReadArgs()–Stil (siehe AutoDocs) an die Namen angehängt, also zum Beipiel ARG1/K/N oder LISTE/M. Bei den Resultatfeldern sind nur die Optionen /M für Liste und /N für numerisch erlaubt.

3.2 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 höhere 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 kann auch entsprechend durch eine angehängte Nummer ergänzt werden.

Der 'MsgPort Basename' wird außerdem als Standard–Dateiextension für von der Applikation gestartete ARexx–Skripten eingesetzt.

3.3 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.

3.4 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.

3.5 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.

3.6 Generate Source

Hiermit wird der Sourcegenerator gestartet. Der im FileRequester eingegebene Name wird um die evtl. vorhandene Extension (`.c, .h oder _rxif.c`) gekürzt, dann wird überprüft, ob schon eine Datei *name*`_rxif.c` existiert.

Falls eine solche existiert fragt die ARexxBox nach, ob diese wirklich überschrieben werden soll. Diese Frage ist wichtig, da in diesem Modul auch Ihr eigener Code stehen kann, Sie sollten sie also nicht leichtfertig bestätigen.

Gegebenenfalls kopieren Sie einfach Ihren alten Interface–Sourcecode in ein anderes File und setzen die Funktionen nachher einzeln sinnvoll zusammen.

Es werden drei Dateien erzeugt: *name*`_rxif.c`, *name*`.c` und *name*`.h`.

4 Der generierte Source

4.1 Dateien

4.1.1 Basismodul

Das Basismodul wird unter dem Namen *name*`.c` abgespeichert und sollte nicht editiert werden. Es enthält die Routinen zum Senden von ARexx–Befehlen, zur Verwaltung der Rexx–Hosts, den ARexx–Dispatcher und falls gewünscht die Funktionen zur Command-Shell.

Eine genaue Beschreibung dieser Funktionen finden Sie im mitgelieferten Dokument `ARexxBox.doc`, welches im AutoDocs–Format gehalten ist.

4.1.2 Headerfile

Das Headerfile wird unter dem Namen *name*`.h` abgespeichert. Es enthält die Deklarationen der Basis– und Interfacefunktionen sowie die zu den Interface–Routinen gehörenden Datenstrukturen zur Übergabe der Parameter und Resultate.

Die Definition `REXX_EXTENSION` legt die Standard–Dateiextension für von Ihrer Applikation gestartete ARexx–Skripten fest. D.h. ein `SendRexxCommand(host, "test");` wird versuchen, das ARexx–Skript `test.<rexx_extension>` zu starten. Die Extension ist gleich dem Messageport–Basisnamen, der nach Möglichkeit dem allgemein verwendeten Projektkürzel (siehe Style Guide) entsprechen sollte.

Die RexxHost–Struktur enthält einen Zeiger auf den Messageport, den Portnamen, einen Reply–Zähler für noch ausstehende Replies vom RexxMaster und einen Zeiger auf eine `RDArgs`–Struktur für das Parsen der Parameter. Für jeden neuen Host wird von `SetupARexxHost()` eine neue Instanz dieser Struktur angelegt, so daß beliebig viele Hosts eröffnet werden können.

Die Stringvariable `RexxPortBaseName` enthält den im Stringgadget eingegebenen Basisnamen für die Ports.

4.1.3 Die Datenstrukturen

Für jede eingegebene ARexx–Funktion wird eine Parameterstruktur definiert, die zumindest aus den beiden Returncode–Feldern (rc und rc2) besteht. Falls das Kommando Argumente entgegennimmt oder Resultate liefert enthält die Struktur außerdem eine Unterstruktur 'arg' bzw. eine Unterstruktur 'res'.

Die Unterstrukturen enthalten zu jedem Eintrag in der jeweiligen Liste (Argumente oder Resultate) ein Feld, das den gleichen Namen wie der Eintrag in der Liste, nur in Kleinbuchstaben (und natürlich ohne die Templateoptionen) hat.

Wenn also zum Beispiel ein Kommando ein Argument `FILENAME/A/K` erwartet, dann steht nach dem Parsen der im Aufruf genannte Filename in `rxd_struct.arg.filename`.

Für Textparameter wird ein Feld vom Typ `char *`, für numerische Werte (Option /N) ein Feld vom Typ `long *` erzeugt. Boolesche Werte (Option /S oder /T) werden als einfache `long`–Typen repräsentiert.

Listen (Option /M) werden durch einen Zeiger auf ein Array von `char *` bzw. `long *` (also `char **` bzw. `long **`) repräsentiert. Die Listen können beliebig lang sein, das Ende einer Liste wird durch einen NULL–Zeiger gekennzeichnet.

Die Typgebung und Bedeutung der Felder bei den Resultaten erfolgt auf genau dieselbe Art und Weise.

4.1.4 Interfacemodul

Im Interface–Modul (unter dem Namen *name*`_rxif.c`) wird für jede definierte ARexx–Funktion ein Funktionsrumpf erzeugt, in den normalerweise nur noch der spezifische Code eingesetzt werden muß.

4.2 Interface–Funktionen

Jede Interface–Funktion wird drei mal vom Dispatcher aufgerufen, das erste Mal um den Speicher für die Parameterstruktur anzufordern und die Defaultwerte dort einzutragen, das zweite Mal um die eigentliche Aufgabe durchzuführen[2], und zum dritten Mal um den Speicher wieder freizugeben.

[2] Anm.: Dieser Durchlauf findet nur statt, wenn das Parsen der Argumente fehlerfrei war

Auf diese Art und Weise ist es möglich, eigene lokale Variablen zu benutzen. Diese müssen, damit der Code reentrant ist (für mehrere Hosts gleichzeitig), in die Parameterdatenstruktur mit eingebunden werden.

Dazu wird einfach eine neue, lokale Struktur definiert, deren erster Bestandteil eine Instanz der Parameterstruktur ist, also z.B.:

```
void rx_help( ..., struct rxd_help **rxd, ... )
{
    struct myrxd {
        struct rxd_help rxd;
        /* Hier die eigenen lokalen Variablen eintragen */
        ...
    } *rd = (struct myrxd *) *rxd;

    ...
}
```

Welcher Teil einer Interface–Prozedur wann aufgerufen wird geht eindeutig aus dem erzeugten Code hervor.

4.3 Resultate

Die Resultate eines ARexx–Befehls müssen — falls vorhanden — in der Substruktur `res` der `rxd`–Struktur eingetragen werden.

Die Namen und Datentypen der Felder entsprechen den Namen und Templateoptionen der Befehlsdefinition, wobei allerdings nur die Templates /M und /N zugelassen sind.

Es werden also immer *Zeiger* auf Daten übergeben, Zeiger auf Zeichen oder Longs oder Zeiger auf Arrays von solchen Zeigern, wobei in einem Array ein NULL–Zeiger das Ende markiert.

Die Routinen der ARexxBox übernehmen die Formatierung in das nachher vom Anwender im Befehlsaufruf angegebene Format. Wenn keines der beiden Schlüsselworte VAR bzw. STEM angegeben wurde, wird das Resultat im VAR–Format als normales ARexx–Resultat zurückgegeben, landet also in der allgemeinen ARexx–Variablen `RESULT`.

Das STEM–Format besteht aus einer Liste von Wertzuweisungen, die aus jeweils einem Variablennamen und dem dazu gehörenden Wert bestehen.

Jeder Variablennamen besteht aus dem im Aufruf übergebenen Wort gefolgt von dem Namen des Resultatfeldes gefolgt von einem Punkt und dann entweder dem Wort “count” oder einer Indexzahl.

Die erste Listeneintragung enthält immer die Zuweisung der Anzahl der in der Liste enthaltenen Werte an die Stemvariable mit der Extension “count”.

Beispiel: Der Befehl `INHALT` habe ein Resultat `INHLISTE/M`. Bei einem ARexx–Aufruf in der Form “`INHALT STEM TEST.`” wird eine Resultatliste der Form

```
TEST.INHLISTE.COUNT = <n>
TEST.INHLISTE.0 = <erster Eintrag>
TEST.INHLISTE.1 = <zweiter Eintrag>
...
TEST.INHLISTE.n-1 = <letzter Eintrag>
```

erzeugt. Das VAR–Format geht aus dem STEM–Format hervor, indem einfach alle Listeneinträge durch SPACEs getrennt zu einem String verkettet werden.

In diesem Zusammenhang muß ich darauf hinweisen, daß für eine Weiterverarbeitung des VAR–Formats bei Resultatstrings, die Spaces enthalten, die Strings durch geeignete Quotes begrenzt sein sollten. Dies wird *nicht* von der ARexxBox übernommen, so daß Sie dafür selbst Sorge tragen müssen.

4.4 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 dieder Variable können sowohl Fehlercodes als auch Fehlerstrings (Beschreibungen) zurückgegeben werden, die dann vom ARexx–Programm abgefragt werden können.

Zur technischen Seite siehe `ARexxBox.doc`.

4.5 Anbindung an ein Programm

Im Hauptprogramm müssen nur noch wenige Schritte unternommen werden, um das ARexx–Interface in Gang zu bringen.

Es müssen die `exec.library`, `dos.library` und die `rexxsyslib.library` geöffnet werden. Die ersten beiden werden normalerweise schon vom Startup-Code des Compilers geöffnet, so daß nur die Rexx–Library von Hand geöffnet werden muß.

Ein ARexx–Host wird mit `SetupARexxHost()` initialisiert. Optional kann man einen anderen Basisnamen für den MsgPort übergeben. Dann muß man nur noch in die normale Event-Wait–Behandlung die Abfrage des Portsignals und den Aufruf des Dispatchers einbauen, und schon läuft die ARexx–Anbindung.

Einer CommandShell wird ein Eingabe–, ein Ausgabe–FileHandle (nicht notwendigerweise verschieden) und ein Prompt–String übergeben, dann läuft sie bis EOF. Es ist z.B. auch möglich, eine CommandShell (mit `dos.library / CreateNewProc`) als eigenen Prozeß asynchron laufen zu lassen. Wenn kein Ausgabe-FileHandle (`NULL`) übergeben wird, erfolgen keine Ausgaben.

Hier ein (minimales) Beispielprogramm:

```
#define HOSTSIG(host) (1L << host->port->mp_SigBit)

void main( int argc, char *argv[] )
{
    struct RexxHost *myhost;
    BPTR fh;

    /* Initialisieren */
```

```
    if( !(RexxSysBase = OpenLibrary( "rexxsyslib.library", 35 )) )
    {
        printf( "No RexxSysLib\n" );
        exit( 20 );
    }

    /* Host eröffnen */

    if( !(myhost = SetupARexxHost(NULL)) )
    {
        printf( "No Host\n" );
        exit( 20 );
    }

    /* Erst eine CommandShell... */

    if( fh = Open( "CON:////CommandShell/AUTO", MODE_NEWFILE ) )
    {
        CommandShell( myhost, fh, fh, "test> " );
        Close( fh );
    }
    else
        printf( "No Console\n" );

    /* ...und dann 'richtiger' ARexx-Betrieb */

    printf( "Address me on Port %s!\n", myhost->portname );
    printf( "Cancel me with CTRL-C\n" );

    while( 1 )
    {
        long s = Wait( SIGBREAKF_CTRL_C | HOSTSIG(myhost) );

        if( s == SIGBREAKF_CTRL_C )
            break;
        else
            ARexxDispatch( myhost );
    }

    CloseDownARexxHost( myhost );
    CloseLibrary( RexxSysBase );
    exit( 0 );
}
```

5 Sonstiges

Diese Anleitung kann keine detaillierte Einführung in die Sprache Rexx oder Programmierung von ARexx–Routinen bieten. Die zusätzliche Lektüre des ARexx–Kapitels im "User Interface Style Guide" von Commodore und des "ARexx User's Reference Manual" von William S. Hawes ist daher sicher keine schlechte Idee.

Die ARexxBox wurde auf einem Amiga 2000 mit A2630, 7 MB RAM, GVP Serie II mit

Quantum P210S und Seagate ST296N und MultiVision Flickerfixer plus NEC Multisync 3D entwickelt.

Compiliert wurde das Ganze mit dem Manx Aztec C 5.2a, ich habe mich halt an das Teil gewöhnt...

5.1 Danksagungen

Ich möchte mich hiermit besonders bedanken bei:

- Meinen aktiven Betatestern: Ralf Kaiser, Garry Glendown, Wolfgang Kueting und Stefan Zeiger.
- Der Amiga–Crew bei Commodore, für den Amiga und AmigaOS Release 2. Ich glaube inzwischen nicht mehr, daß es einen Amiga vor Release 2 gab.
- 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 François, für seine ReqTools.Library mit ihren wirklich Programmierer– und Anwenderfreundlichen Requestern.
- William S. Hawes für seine hervorragende Rexx–Portierung.

5.2 Bugreports und Vorschläge

Ich bitte darum, mir Bugreports und Vorschläge möglichst per EMail zuzuschicken.

Bei Bugreports bitte *immer* die genaue Konfiguration angeben, den Fehler möglichst *detailliert* beschreiben und versuchen, ihn auf ein *minimales* Programm zu reduzieren, das den Fehler verläßlich reproduziert.

Vorschlägen gegenüber bin ich immer aufgeschlossen.

Michael Balzer