

# Scout

---

System Monitor  
Scout 37.105 (Release 2.2)  
Edition 2.2  
April 1995

von Andreas Gelhausen

---

# 1 Einleitung

## Was ist Scout?

**Scout** ist ein Systemmonitor, d.h. viele für den reibungslosen Betrieb des Rechners notwendige Strukturen — wie z.B. Tasks, Ports, Assigns, System-Erweiterungen, residente Befehle, Interrupts, usw. — können angeschaut und auf viele dieser Strukturen können auch bestimmte Aktionen ausgeführt werden.

Es können zum Beispiel Tasks und Prozesse eingefroren, Windows und Screens geschlossen, Semaphore freigegeben und Interrupts aus dem System entfernt werden.

**Scout** bietet zusätzlich die Möglichkeit, via **AmiTCP** auch andere Rechner beobachten und gegebenenfalls auch dort auf viele Strukturen zugreifen zu können.

Fast alle der implementierten Funktionen stehen auch als Shell-Parameter zur Verfügung. Das **Magic User Interface** ist nur für die grafische Benutzeroberfläche notwendig und demnach nicht unbedingt erforderlich.



## 2 Rechtliche Dinge

### Copyright

Scout 37.105 (Release 2.2) — Copyright © 1994 by Andreas Gelhausen, alle Rechte vorbehalten.

Scout ist *Giftware* und darf nur als vollständiges, unverändertes Archiv frei kopiert werden. Weder das Programm noch Teile davon dürfen ohne schriftliche Genehmigung des Autors zusammen mit kommerziellen Programmen vertrieben werden.

### Keine Garantie

Es wird nicht garantiert, daß das Programm fehlerfrei ist und immer ordnungsgemäß arbeitet. Sie benutzen es auf eigene Gefahr. Für Folgeschäden, gleich welcher Art, die durch die Benutzung des Programmes **Scout** entstehen, übernimmt der Autor keine Haftung.

### Giftware

Scout 37.105 ist *Giftware*, d.h. wenn Ihnen das Programm gefällt, und Sie es demnach auch benutzen, dann sollten Sie dem Autor für den Aufwand, den er bei der Entwicklung des Programmes gehabt hat, eine kleine Aufmerksamkeit zukommen lassen.

Für kleine Geschenke jeglicher Art stehe ich immer gern zur Verfügung. =: ^)

Ansonsten darf das Programm frei benutzt werden!



## 3 Vor dem Programmstart

### 3.1 Systemanforderungen

Scout benötigt mindestens die Kickstart Version 2.04.

Möchten Sie das Programm mit der grafischen Benutzungsoberfläche benutzen, dann müssen Sie die MUI-Version 2.1 oder eine höhere Version von MUI installieren. Siehe auch Abschnitt 3.2 [MUI], Seite 5.

Um die Netz-Funktionen von Scout benutzen zu können, sollten Sie mindestens die AmiTCP-Version 4.0 installiert haben. Siehe auch Abschnitt 3.3 [AmiTCP], Seite 5.

### 3.2 MUI - MagicUserInterface

© Copyright 1993/94 Stefan Stuntz

MUI ist ein System zum Erzeugen und Unterstützen von grafischen Benutzungsoberflächen. Mit der Hilfe eines Konfigurationsprogrammes bekommt der Benutzer einer MUI-Applikation die Möglichkeit das Aussehen dieser Applikation seinem Geschmack anzupassen.

MUI wird als Shareware vertrieben. Um ein vollständiges Programmpaket zu bekommen, das viele Beispiele und mehr Informationen über die Registrierung beinhaltet, sollten Sie auf lokalen Bulletin Boards oder Public Domain Disketten nach einem File namens 'muiXXusr.lha' Ausschau halten (XX steht für die letzte Versionsnummer).

Sie können sich auch direkt registrieren lassen, indem Sie 30.- DM oder 20.- US\$ an die folgende Adresse schicken:

Stefan Stuntz  
Eduard-Spranger-Straße 7  
80935 München  
GERMANY

### 3.3 AmiTCP

AmiTCP ist ein TCP/IP Protokoll-Stack für den Amiga. Die Demoversion 4.0 (oder neuer) sollte in jeder größeren Public-Domain-Sammlung oder auf dem AmiNet erhältlich sein. Fragen Sie den Amiga-Händler Ihres Vertrauens. =:~)

### 3.4 Installation

Für die Installation von Scout reicht es aus, nur das Programm 'scout' selbst und die Datei 'scout.data' in ein Verzeichnis Ihrer Wahl zu kopieren. Danach können Sie es sofort starten. Die Datei 'scout.data' beinhaltet Daten von System-Erweiterungen.



## 4 Wie wird Scout benutzt?

In diesem Kapitel wird die Benutzung von **Scout** über die grafische Benutzungsoberfläche beschrieben. Diese grafische Benutzungsoberfläche wurde mit MUI realisiert, das für die grafische Benutzung von **Scout** auch im System vorhanden sein muß. Siehe auch Abschnitt 3.2 [MUI], Seite 5.

Möchten Sie — aus welchem Grund auch immer — MUI nicht verwenden, dann sollten Sie sich den Abschnitt 4.20 [Scout ohne MUI], Seite 22 anschauen.

Wenn Sie das Programm starten, wird das Hauptfenster geöffnet, welches viele Gadgets beinhaltet. Jedes dieser Gadgets steht für eine bestimmte Art von für das Betriebssystem notwendigen Strukturen.

Sie können wählen zwischen:

Assigns, Devices, Expansions, Fonts, InputHandlers, Interrupts, Libraries, Locks, Memory, Mounted Devices, Ports, Resident Commands, Residents, Resources, Semaphores, Tasks, Vectors and Windows.

Betätigen Sie eines dieser Gadgets, dann wird ein weiteres Fenster geöffnet, welches die jeweils dazugehörige Liste von Strukturen beinhaltet.

**Beispiel:** Betätigen Sie das ‘Tasks’-Gadget, so wird ein Fenster mit der aktuellen Task-Liste des Systems geöffnet.

Diese ganzen Funktionen können auch jeweils über das Menu und durch eine Taste aufgerufen werden, die durch das unterstrichene Zeichen auf jedem Gadget bestimmt wird.

Mit diesem Programm können Sie auf viele dieser Strukturen bestimmte Aktionen ausführen lassen. Sollten Sie so etwas in Betracht ziehen, dann sollten Sie sich bewußt sein, was Sie tun.

**Achtung:** Unsachgemäße Manipulation der System-Strukturen kann zum Absturz des Systems führen. In schweren Fällen kann dies einen Datenverlust zur Folge haben.

**Hinweis:** Da es für die Anleitung eines solchen Programmes zu aufwendig wäre, die angegebenen Strukturen bis ins letzte Detail zu erklären, wundern Sie sich bitte nicht, daß einige Detail-Informationen fehlen.

Da über diese Dinge schon Bücher über Bücher geschrieben wurden, verweise ich an dieser Stelle auf die dafür vorgesehene Fachliteratur!

### 4.1 Assigns

Ein Assign weist einem Verzeichnis einen logischen Namen zu.

Wenn Sie zum Beispiel einem Verzeichnis ‘DH0:Daten/Dokumente’ den logischen Namen ‘Texte:’ zuweisen, dann können Sie auf eine Datei *Dateiname*, die sich in diesem Verzeichnis befindet, auch durch die Angabe von ‘Texte:*Dateiname*’ zugreifen.

#### Spalteneinträge

- ‘Address’ An dieser Adresse beginnt die Struktur eines Assign-Eintrages.
- ‘Name’ Logischer Name eines Verzeichnisses oder Gerätes
- ‘Path’ Hier steht der Pfad des Verzeichnisses.

#### Aktionen

- ‘Update’ Betätigen Sie dieses Gadget, dann wird die Liste erneut eingelesen.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der ‘Assigns’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.



- ‘Remove’ Mit dieser Funktion wird der ausgewählte Assign-Eintrag aus dem System entfernt.
- ‘Exit’ Das ‘Assigns’-Fenster wird geschlossen.

## 4.2 Devices

Ein Device, das sich in dieser Liste befindet, ist — wie auch eine Library (siehe Abschnitt 4.7 [Libraries], Seite 12) — eine Ansammlung von Funktionen bzw. Routinen, denen bestimmte Aufgaben zugedacht wurden.

Das ‘trackdisk.device’ zum Beispiel beinhaltet Funktionen für die Handhabung von Disketten bzw. der Laufwerke.

### Spalteneinträge

- ‘Address’ Adresse der Device-Struktur
- ‘ln\_Name’ Name eines Devices
- ‘ln\_Pri’ Priorität eines Devices
- ‘OpenC’ Zähler, der angibt, wie oft das Device geöffnet wurde.
- ‘RPC’ ‘RPC’ steht für ‘RAM Pointer Count’ und gibt an, wieviele Sprungadressen des Devices ins RAM zeigen. So eine ins RAM zeigende Einsprungadresse weist auf ein Programm (z.B. den SetPatch-Befehl) hin, welches die alte Funktion verbessern bzw. erneuern möchte, indem es einfach die Sprungadresse der Funktion durch die Adresse einer eigenen Funktion ersetzt.  
Viele Viren hängen sich auf diese Weise ins System. Diese Tatsache soll Sie aber jetzt nicht in Panik versetzen, da es sich in den meisten Fällen um kleine Patch-Programme — wie den SetPatch-Befehl von Commodore — handelt.  
Sollten alle Sprungadressen eines Devices ins RAM zeigen, dann hat es seinen Programmcode im RAM stehen. Ein solcher ‘RPC’-Eintrag besteht aus drei Sternen, da es in dem Fall unwichtig ist, wieviele Sprungadressen ins RAM zeigen.
- ‘ln\_Type’ Typ dieser Struktur (Hier sollte normalerweise ‘device’ stehen.)

### Aktionen

- ‘Update’ Die Device-Liste wird erneut ausgelesen.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der ‘Devices’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Remove’ Mit dieser Funktion wird das ausgewählte Device entfernt. Voraussetzung hierfür ist allerdings, daß es von keinem Programm mehr benutzt wird bzw. der ‘OpenC’ gleich Null ist.
- ‘Priority’ Die Priorität des Devices kann hier von Ihnen verändert werden. Hierzu erscheint ein kleines Fenster, in dem Sie eine neue Priorität angeben können. Durch die veränderte Priorität bekommt das Device eventuell einen neuen Platz in der Device-Liste.
- ‘More’ Ein zusätzliches Fenster wird geöffnet, in dem Sie weitere Details des selektierten Devices finden.  
Sie erreichen dasselbe, indem Sie einfach einen Doppelklick auf den jeweiligen Device-Eintrag ausführen.
- ‘Exit’ Das ‘Devices’-Fenster wird geschlossen.

### 4.3 Expansions (System-Erweiterungen)

In dieser Liste werden alle Ihre System-Erweiterungen angezeigt, die zur Zeit dem System zur Verfügung stehen (Grafikkarten, Speichererweiterungen usw.).

#### Spalteneinträge

**‘BoardAddr’**

Das ROM der Karte ist ab dieser Adresse im Speicher zu finden. Sollte es sich bei der Karte um eine Speichererweiterung handeln, ist hier die Anfangsadresse des konfigurierten Speichersegmentes zu finden.

**‘BoardSize’**

Handelt es sich bei dem Listen-Eintrag um eine Speichererweiterung, dann steht hier die Byte-Anzahl, die dem System durch diese Karte als Speicher zur Verfügung gestellt wird.

Bei *normalen* Karten wird hier nur die Größe des zur Karte gehörenden ROMs angegeben.

**‘Manufacturer’**

Herstellernummer, die von Commodore vergeben wird.

**‘Product’** Produktnummer, die der System-Erweiterung vom Hersteller gegeben wird.

**‘Serial#’** Seriennummer der Karte (Dieser Eintrag wird von den meisten Karten nicht benutzt.)

#### Aktionen

**‘Print’** Mit Hilfe dieser Funktion können Sie die Liste der **‘Expansions’** zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.

**‘More’** Beim Betätigen dieses Gadgets erhalten Sie mehr Informationen über die selektierte System-Erweiterung in einem zusätzlichen Fenster.  
Sie erreichen dasselbe, indem Sie einfach einen Doppelklick auf den jeweiligen Eintrag der Liste ausführen.

**‘Exit’** Das **‘Expansions’**-Fenster wird geschlossen.

#### Unbekannte System-Erweiterungen

Wenn Sie eine System-Erweiterung durch einfaches Anklicken des jeweiligen Eintrages mit der Maus selektieren, dann erhalten Sie den Namen der Herstellerfirma und die Bezeichnung der Karte in dem dafür vorgesehenen Textfeld unterhalb der Liste. Das passiert natürlich nur, sofern mir diese Daten bei der Erstellung der jeweiligen Programmversion von Scout bekannt waren!

Sollten diese Angaben fehlen oder nicht mit den Daten Ihrer System-Erweiterungen übereinstimmen, so möchte ich Sie bitten, mir die folgenden Daten zuzusenden, damit ich sie dem Programm beifügen bzw. sie korrigieren kann. In der nächsten Version der Datei **‘scout.data’** sollten diese Angaben dann vorhanden sein.

#### Daten zur Erfassung einer nicht namentlich genannten Erweiterung:

1. Herstellernummer (Manufacturer)
2. Produktnummer (Product)
3. Name des Herstellers
4. Bezeichnung der Hardware

Seien Sie hierbei bitte so genau wie möglich. Die Version der Erweiterung oder auch noch andere Angaben können hierbei nicht schaden.

## 4.4 Fonts

Alle Zeichensätze, die sich zur Zeit im System befinden bzw. von Programmen benutzt werden, sind in dieser Liste zu finden.

### Spalteneinträge

‘YSize’	Vertikale Größe des Zeichensatzes
‘Count’	Zähler, der angibt, von wievielen Programmen der Zeichensatz gerade benutzt wird.
‘Type’	Steht an dieser Stelle ‘ROMFONT’, so befindet sich dieser Zeichensatz im ROM. Bei ‘DISKFONT’ wurde er von Diskette bzw. Festplatte geladen.
‘Name’	Name des Zeichensatzes

### Aktionen

‘Update’	Die Liste der Zeichensätze wird aktualisiert.
‘Print’	Mit Hilfe dieser Funktion können Sie die Liste der ‘Fonts’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
‘Close’	Hiermit kann ein Zeichensatz geschlossen werden. ‘Count’ verringert sich dann um eins.
‘Remove’	Mit dieser Funktion kann ein Zeichensatz aus dem System (Speicher) entfernt werden, vorausgesetzt er wird von keinem Programm mehr benötigt und befindet sich nicht im ROM.
‘Exit’	Das ‘Fonts’-Fenster wird geschlossen.

## 4.5 Inpuhandler

Inpuhandler kümmern sich um die Benutzereingaben, die im System ankommen (Tastendrücke, Mausklicks, usw.). Sie stehen wie an einem Fließband in einer Reihe und werten diese Eingaben aus. Der Inpuhandler mit der höchsten Piorität bearbeitet diese Eingaben zuerst. Kann er mit den Eingaben nichts anfangen, reicht er sie in der Regel an den nächsten Inpuhandler weiter.

Das System benutzt normalerweise für seinen Inpuhandler die Priorität 50. Möchte also ein Inpuhandler die Benutzereingaben vor dem System bekommen, braucht er eine höhere Priorität.

### Spalteneinträge

‘ln_Name’	Name des Inpuhandlers
‘ln_Pri’	Priorität des Inpuhandlers
‘is_Data’	Ab dieser Adresse sind die Daten des Inpuhandlers im Speicher zu finden.
‘is_Code’	Diese Adresse zeigt zum Programmcode des Inpuhandlers. Sollte diese Adresse ins RAM zeigen, so wird sie andersfarbig dargestellt. Der Inpuhandler des Betriebssystems hat seinen Programmcode im ROM.  Ein paar Viren klinken sich als Inpuhandler ins System. Bei denen zeigt dann auch die ‘is_Code’-Adresse ins RAM. Wiederum gilt auch in einem solchen Fall: Nicht gleich die Panik bekommen, es gibt genug <i>normale</i> Programme, die so verfahren.

## Aktionen

- ‘Update’ Die Liste der Inputhandler wird auf den neuesten Stand gebracht.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der ‘InputHandlers’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Remove’ Ein Inputhandler kann mit Hilfe dieser Funktion aus dem System entfernt werden. Hierbei zieht man dem System aber eventuell den Stuhl unter dem Hintern weg. Das System kann dabei leicht abstürzen!
- ‘Priority’  
Die Priorität des Inputhandlers kann auf einen bestimmten Wert gesetzt werden. Wird die Priorität eines Inputhandlers verringert, kann es passieren, daß Programme nicht mehr auf bestimmte Dinge (z.B. das Drücken einer bestimmten Taste) reagieren, da ein Inputhandler mit einer höheren Priorität diese absorbiert.  
Auch diese Liste wird vom System nach den Prioritäten sortiert. Ändern Sie also die Priorität eines Inputhandlers, dann bekommt dieser eventuell einen neuen Platz in der Liste.
- ‘Exit’ Das Fenster wird geschlossen.

## 4.6 Interrupts

Interrupts sind bestimmte Ereignisse, auf die das Betriebssystem reagieren muß. Für jeden Interrupt-Typ stehen meist sogar mehrere Interrupt-Routinen zur Verfügung. Diese Interrupt-Routinen werden in einer Liste nach Prioritäten sortiert.

Sobald also ein bestimmter Interrupt auftritt, wird das laufende Programm solange unterbrochen, bis die zum jeweiligen Interrupt gehörende Liste der Interrupt-Routinen abgearbeitet wurde.

### Spalteneinträge

- ‘ln\_Name’ Diesem Text kann normalerweise entnommen werden, von welchem Programm die Interrupt-Routine installiert wurde und auch benötigt wird.
- ‘ln\_Pri’ Priorität der Interrupt-Routine
- ‘is\_Data’ Ab dieser Adresse sind im Speicher Daten zu finden, die zur Interrupt-Routine gehören.
- ‘is\_Code’ Der Programmcode der Interrupt-Routine ist hier zu finden. Sollte diese Adresse ins RAM zeigen, so wird sie andersfarbig dargestellt.
- ‘NUM’ Diese Nummer beschreibt das Ereignis, bei dem die Interrupt-Routine aufgerufen wird. Eine kleine Information hierzu finden Sie im ‘IntName’-Eintrag des Interrupt-Detail-Fensters, das durch das Betätigen des ‘More’-Gadgets geöffnet wird.  
**Beispiel:** Nummer 5 bedeutet, daß die Interrupt-Routine bei jedem neuen Bildaufbau ihres Monitors aufgerufen wird, was bei einem 50 Hz Monitor 50 mal in der Sekunde passiert. (‘VERTB (vertical blank interval)’)

## Aktionen

- ‘Update’ Die Liste der Interrupt-Routinen wird aktualisiert.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der Interrupt-Routinen zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.

- ‘Remove’** Mit dieser Funktion kann eine Interrupt-Routine aus der Liste entfernt werden. Sollte es sich bei der Interrupt-Routine allerdings um einen Interrupt-Handler handeln, kann **Scout** diese Aktionen nicht ausführen. Ist dies der Fall, dann steht in der Spalte **‘IntType’** der Text **‘Handler’**.  
Bei den Interrupt-Handlern vom **audio.device** kann dieses Problem z.B. gelöst werden, indem das **‘audio.device’** entfernt wird. Das passiert unter anderem durch den Aufruf von **‘avail flush’**, wenn das **audio.device** von keinem Programm mehr benutzt wird.
- ‘More’** Ein Fenster mit weiteren Informationen über den selektierten Interrupt wird geöffnet.
- ‘Exit’** Betätigen Sie dieses Gadget, dann wird das Fenster geschlossen.

## 4.7 Libraries

Eine Library ist eine Ansammlung von Funktionen/Routinen (Bibliothek), denen bestimmte Aufgaben zugedacht wurden.

Die **graphics.library** zum Beispiel beinhaltet Funktionen für die Grafikdarstellung.

### Spalteneinträge

- ‘Address’** Adresse einer Library
- ‘ln\_Name’** Name einer Library
- ‘ln\_Pri’** Priorität einer Library
- ‘OpenC’** Zähler, der angibt, wie oft die Library geöffnet wurde.
- ‘RPC’** **‘RPC’** steht für **‘RAM Pointer Count’** und gibt an, wieviele Sprungadressen der Library ins RAM zeigen. So eine ins RAM zeigende Einsprungadresse weist auf ein Programm (z.B. den **SetPatch**-Befehl) hin, welches die *alte* Funktion verbessern bzw. erneuern möchte, indem es einfach die Sprungadresse der Funktion durch die Adresse einer eigenen Funktion ersetzt.  
Viele Viren hängen sich auf diese Weise ins System. Diese Tatsache soll Sie aber jetzt nicht in Panik versetzen, da es sich in den meisten Fällen um kleine Patch-Programme — wie den **SetPatch**-Befehl von Commodore — handelt.  
Sollten alle Sprungadressen einer Library ins RAM zeigen, dann hat sie ihren Programmcode im RAM stehen. Ein solcher **‘RPC’**-Eintrag besteht aus drei Sternen, da es in dem Fall unwichtig ist, wieviele Sprungadressen ins RAM zeigen.
- ‘ln\_Type’** Typ dieser Struktur (Hier sollte normalerweise **‘library’** stehen.)

### Aktionen

- ‘Update’** Die Library-Liste wird erneuert.
- ‘Print’** Mit Hilfe dieser Funktion können Sie die Liste der **‘Libraries’** zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Remove’** Mit dieser Funktion wird die selektierte Library entfernt. Voraussetzung hierfür ist allerdings, daß sie von keinem Programm mehr benutzt wird bzw. der **‘OpenC’** gleich Null ist.  
Einige Libraries lassen sich nicht mehr ohne einen Reset aus dem System entfernen. Es ist also nicht unbedingt verwunderlich, wenn **Scout** es einmal nicht schaffen sollte, eine Library zu entfernen!

- ‘Close’** Um eine Library aus dem System entfernen zu können, muß sie von allen Programmen wieder geschlossen worden sein. Dies ist der Fall, wenn der **‘OpenC’**-Eintrag den Wert Null hat.
- Wenn Sie mit dieser Funktion eine Library schließen möchten, werden Sie gefragt, ob Sie die Library nur einmal oder gleich für alle Programme schließen möchten, die diese Library geöffnet haben.
- Wählen Sie hier also **‘all’**, dann wird die Library so oft geschlossen, bis der **‘OpenC’** gleich Null ist.
- ‘Priority’** Die Priorität der Library kann von Ihnen verändert werden. Hierzu erscheint ein kleines Fenster, in dem Sie die neue Priorität angeben können. Durch die veränderte Priorität bekommt die Library eventuell einen neuen Platz in der Liste.
- ‘More’** Ein Fenster mit weiteren Informationen zur Library wird geöffnet.
- ‘Exit’** Das **‘Libraries’**-Fenster wird geschlossen.

## 4.8 Locks

Ein Lock symbolisiert den Zugriff eines Programmes auf eine Datei oder ein Verzeichnis. Auf diese Weise wird z.B. verhindert, daß eine Datei gelöscht wird, während irgendein anderes Programm noch auf die sich in der Datei befindenden Daten zugreift.

Bei etwas umfangreicheren Systemen kann der Aufbau der Liste etwas länger dauern! Mein eigenes System hat z.B. im Durchschnitt ca. 500 Lockeinträge, was gemessen an anderen Systemen noch nicht allzu viel ist. =:~)

### Spalteneinträge

- ‘Access’** Hier wird die Zugriffsart des Lock-Zugriffes angegeben. Dies kann ein Lese- (**‘READ’**) oder ein Schreibzugriff (**‘WRITE’**) sein. Sollte hier **‘OWN’** stehen, dann handelt es sich nur um einen Lock, der zum Aufbau dieser Liste von Scout angefordert wurde.
- ‘Path’** Pfad der Datei oder des Verzeichnisses

### Aktionen

- ‘Update’** Die Liste der Locks wird aktualisiert.
- ‘Print’** Mit Hilfe dieser Funktion können Sie die Liste der **‘Locks’** zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Remove’** Ein Lock wird mittels der **UnLock()**-Funktion der dos.library wieder freigegeben.
- ‘Pattern’** Geben Sie hier ein Namensmuster an, so werden nur die Locks angezeigt, deren Pfad mit dem Namensmuster übereinstimmt.
- ‘Exit’** Das Fenster wird geschlossen.

## 4.9 Memory (Speichersegmente)

Die Einträge dieser Liste stellen die Speichersegmente Ihres Rechners dar. Sie finden dort mindestens den Eintrag Ihres Grafik-Speichers (**‘CHIP-MEMORY’**), der fest in Ihren Rechner eingebaut ist.

## Spalteneinträge

<code>'ln_Name'</code>	Name des Speichersegmentes (z.B. <code>'chip memory'</code> )
<code>'ln_Pri'</code>	Priorität des Speichersegmentes
<code>'mh_Lower'</code>	Anfangsadresse des Speichersegmentes
<code>'mh_Upper'</code>	Endadresse des Speichersegmentes

## Aktionen

<code>'Print'</code>	Mit Hilfe dieser Funktion können Sie die Liste der Speichersegmente zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
<code>'Priority'</code>	<p>Mit dieser Funktion können Sie bestimmen, welches Speichersegment bevorzugt vom System und den anderen Programmen benutzt werden soll, indem Sie diesem eine höhere Priorität geben als den anderen Speichersegmenten.</p> <p><b>Ausnahme:</b> Wird der Typ des Speichers direkt bei der Anforderung eines Programmes angegeben, wird das erste Speichersegment benutzt, das die Anforderungskriterien erfüllt.</p>
<code>'More'</code>	Ein neues Fenster wird geöffnet. Dieses Fenster enthält weitere Daten zum selektierten Speichersegment.
<code>'Exit'</code>	Das <code>'Memory'</code> -Fenster wird geschlossen.

## 4.10 Mounted Devices

In dieser Liste finden Sie alle Ihre ansprechbaren Geräte (Laufwerke, Festplatten usw.).

## Spalteneinträge

<code>'Name'</code>	Name des Gerätes
<code>'Unit'</code>	Kennziffer des Gerätes (Bei DF2: steht hier z.B. normalerweise eine Zwei.)
<code>'Heads'</code>	Anzahl der vorhandenen Lese- bzw. Schreib-Köpfe
<code>'Cyl'</code>	Anzahl der Zylinder
<code>'State'</code>	Zustand eines Gerätes, der z.B. angibt, ob eine Diskette im Laufwerk liegt oder ob die Diskette unlesbar ist.
<code>'DiskType'</code>	Typ der Diskette (z.B. <code>'OFS'</code> (OldFileSystem), <code>'FFS'</code> (FastFileSystem), ...)
<code>'Handler or Device'</code>	<p>Hier wird angegeben, welcher Handler oder welches Device sich um den Zugriff auf das jeweilige Gerät kümmert.</p> <p>Beim Laufwerk DF0: wäre es z.B. in der Regel das <code>'trackdisk.device'</code>. Um also direkt auf die Sektoren von DF0: schreiben zu können, müßten Sie das <code>'trackdisk.device'</code> benutzen.</p>

## Aktionen

- ‘Update’ Die Liste wird erneut eingelesen.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der Geräte zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘More’ Ein weiteres Fenster mit mehr Informationen zum ausgewählten Gerät wird geöffnet.
- ‘Exit’ Das Fenster wird geschlossen.

### 4.11 Ports

Ports dienen der Kommunikation von Programmen. Dem Port eines Programmes können Mitteilungen gesendet werden, auf die das Programm reagieren soll.

## Spalteneinträge

- ‘Address’ An dieser Adresse ist die Port-Struktur zu finden.
- ‘ln\_Name’ Name des Ports
- ‘ln\_Pri’ Priorität des Ports
- ‘mp\_SigTask’  
Name des Tasks, der für diesen Port zuständig ist.

## Aktionen

- ‘Update’ Die Liste der Ports wird aktualisiert.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der ‘Ports’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Remove’ Der Port wird aus dem System entfernt.
- ‘Priority’  
Mit Hilfe dieser Funktion kann die Priorität des Ports verändert werden.
- ‘More’ Ein neues Fenster wird geöffnet. Dieses Fenster enthält weitere Daten zum selektierten Port.
- ‘Exit’ Das ‘Ports’-Fenster wird geschlossen.

### 4.12 Resident Commands (Residente Befehle)

Alle Kommandos, die durch den Shell-Befehl **resident** resident gemacht wurden, und die Befehle, die schon im ROM enthalten sind, werden hier angezeigt.

Dabei werden auch die Positionen und die Größen aller Hunks der jeweiligen Befehle aufgelistet.

Die hier behandelten ‘residenten Befehle’ haben nichts mit den im nächsten Abschnitt beschriebenen ‘residenten Strukturen’ zu tun.



## Spalteneinträge

<b>'Name'</b>	Name des Befehls
<b>'UseCount'</b>	Zähler, der angibt, wieviele Instanzen des Befehls zur Zeit des Listenaufbaus im System aktiv sind.
<b>'Lower'</b>	Startadresse eines Hunks im Speicher
<b>'Upper'</b>	Endadresse eines Hunks im Speicher
<b>'Size'</b>	Größe des Hunks ('Upper' - 'Lower' - 8 Bytes Overhead)

## Aktionen

<b>'Update'</b>	Die Liste der residenten Befehle wird erneut eingelesen.
<b>'Print'</b>	Mit Hilfe dieser Funktion können Sie die Liste der residenten Befehle zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
<b>'Remove'</b>	Mit dieser Funktion wird der ausgewählte residente Befehl aus der Liste entfernt. Voraussetzung hierfür ist allerdings, daß er nicht mehr benutzt wird bzw. der <b>'UseCount'</b> gleich Null ist.
<b>'Exit'</b>	Das Fenster wird geschlossen.

## 4.13 Residents (Residente Strukturen)

Residente Strukturen (Residents) sind Code- bzw. Daten-Segmente (wie zum Beispiel Libraries), die einen Reset überstehen. Sie sind *reset-fest*.

Die hier behandelten 'residenten Strukturen' haben nichts mit den im vorigen Abschnitt beschriebenen 'residenten Befehlen' zu tun.

Ein Programmierer hat nun die Möglichkeit sein Programm reset-fest zu machen, indem er unter anderem eine Resident-Struktur initialisiert und diese über die Kick-Vektoren (siehe Abschnitt 4.17 [Vectors], Seite 20), die sich in der ExecBase-Struktur (Basis der exec.library) befinden, ins System einklinkt.

Diese residenten Strukturen liegen demnach im RAM und ihre Adressen werden andersfarbig dargestellt, um sie von den anderen residenten Strukturen abzuheben. Die residenten Strukturen, die über die Kick-Vektoren ins System gekommen sind, werden, sofern überhaupt solche residenten Strukturen vorhanden sind, am oberen Ende der Liste eingefügt.

Sollten Sie hier eine residente Struktur finden, die ins RAM zeigt, dann ist Vorsicht geboten. Schauen Sie sich ihren Namen an, und wenn Sie nicht ganz sicher wissen, worum es sich handelt, sollten Sie lieber einmal den Virenkiller Ihres Vertrauens das System überprüfen lassen.

Viele Viren machen sich auf diese Weise reset-fest!

## Spalteneinträge

<b>'Address'</b>	An dieser Adresse ist die residente Struktur zu finden.
<b>'In_Name'</b>	Name der residenten Struktur
<b>'rt_Pri'</b>	Priorität der residenten Struktur
<b>'rt_IdString'</b>	Identifikationstext der residenten Struktur

## Aktionen

‘Update’	Die Liste der residenten Strukturen wird aktualisiert.
‘Print’	Mit Hilfe dieser Funktion können Sie die Liste der ‘Residents’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
‘More’	Ein neues Fenster mit mehr Informationen über die Resident-Struktur wird geöffnet.
‘Exit’	Das ‘Residents’-Fenster wird geschlossen.

## 4.14 Resources (Ressourcen)

Eine Ressource ist — wie auch eine Library (siehe Abschnitt 4.7 [Libraries], Seite 12) und ein Device (siehe Abschnitt 4.2 [Devices], Seite 8) — eine Ansammlung von Funktionen bzw. Routinen, denen bestimmte Aufgaben zugedacht wurden.

### Spalteneinträge

‘Address’	Adresse der Ressource
‘ln_Name’	Name der Ressource
‘ln_Pri’	Priorität der Ressource
‘OpenC’	Zähler, der angibt, wie oft die Ressource geöffnet wurde.
‘RPC’	<p>‘RPC’ steht für ‘RAM Pointer Count’ und gibt an, wieviele Sprungadressen der Ressource ins RAM zeigen. So eine ins RAM zeigende Einsprungadresse weist auf ein Programm hin (wie z.B. den <code>SetPatch</code>-Befehl), welches die ‘alte’ Funktion verbessern bzw. erneuern möchte, indem es einfach die Sprungadresse der Funktion durch die Adresse einer eigenen Funktion ersetzt.</p> <p>Sollten alle Sprungadressen einer Ressource ins RAM zeigen, dann hat sie ihren Programmcode im RAM stehen. Ein solcher ‘RPC’-Eintrag besteht aus drei Sternen, da es in dem Fall unwichtig ist, wieviele Sprungadressen ins RAM zeigen.</p>
‘ln_Type’	Typ der Struktur (Hier sollte normalerweise ‘resource’ stehen.)

### Aktionen

‘Update’	Die Ressource-Liste wird neu eingelesen.
‘Print’	Mit Hilfe dieser Funktion können Sie die Liste der ‘Resources’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
‘Remove’	Mit dieser Funktion wird die gewählte Ressource entfernt. Voraussetzung hierfür ist allerdings, daß sie von keinem Programm mehr benutzt wird bzw. der ‘OpenC’ gleich Null ist.
‘Priority’	Die Priorität der Ressource kann von Ihnen verändert werden. Hierzu erscheint ein kleines Fenster, in dem Sie die neue Priorität angeben können.
‘More’	Wird dieses Gadget betätigt, dann erscheint ein zusätzliches Fenster mit weiteren Daten zur selektierten Ressource.
‘Exit’	Das ‘Residents’-Fenster wird geschlossen.

**Beachte:** Sollte bei ‘OpenC’ und/oder ‘RPC’ ein Strich stehen, so besitzt die Ressource keine typische Library-Struktur (Hintereinanderreihung von Sprungbefehlen und deren Sprungadressen). Das passiert z.B. beim Eintrag der ‘`FileSystem.resource`’.

## 4.15 Semaphores (Semaphore)

Semaphore sind normalerweise dafür da, den Zugriff auf bestimmte Geräte zu handhaben, auf die nur eine bestimmte Anzahl von Programmen zur Zeit zugreifen darf.

### Beispiele:

1. Auf einen Drucker darf nur ein Programm zur Zeit zugreifen, da sonst die zu druckenden Texte 'gemischt' würden.
2. Wenn der **SetPatch**-Befehl von Commodore z.B. schon die Routinen des Betriebssystems gepatcht hat, dann soll er diese Patches beim nächsten Aufruf ja nicht nochmal ausführen. Zu diesem Zweck wird ein Semaphor eingerichtet. Der **SetPatch**-Befehl kann dadurch bei einem erneuten Start prüfen, ob er schon einmal ausgeführt worden ist.

### Spalteneinträge

'ln_Name'	Name des Semaphors
'Nest'	Dieser Zähler zeigt, wie oft der 'Owner'-Task den Semaphor benutzt.
'Queue'	Hier wird angezeigt, wieviele Tasks den Semaphor besitzen möchten.
'Owner'	Hier ist der Name des Tasks zu finden, dem der Semaphor zur Zeit gehört.

### Aktionen

'Update'	Die Liste der Semaphore wird erneut eingelesen.
'Print'	Mit Hilfe dieser Funktion können Sie die Liste der Semaphore zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
'Obtain'	Hierdurch wird dem System vorgegaukelt, daß das Gerät, das File oder wofür der Semaphor sonst eingerichtet wurde, gerade benutzt wird. Der 'NestCnt'-Eintrag erhöht sich hierbei um Eins.
'Release'	Sollte ein Semaphor gerade benutzt werden, so machen Sie dem System mit dieser Funktion weis, daß dem nicht mehr so ist. Ein Programm, das den Semaphor beachtet, kann so eventuell versuchen, ein weiteres Mal auf das entsprechende Gerät zuzugreifen.
'Remove'	Sofern der Semaphor nicht mehr benutzt wird, können Sie ihn anhand dieser Funktion aus dem System entfernen.
'Exit'	Das 'Semaphores'-Fenster wird geschlossen.

## 4.16 Tasks

In dieser Liste befinden sich alle Tasks und Prozesse. (Prozesse sind erweiterte Task-Strukturen.) Sie repräsentieren die Programme, die im Augenblick im System ablaufen bzw. auf ein Ereignis warten.

### Spalteneinträge

'ln_Name'	Name des Tasks
'ln_Type'	Typ der Struktur ('task' oder 'process')
'ln_Pri'	Priorität des Tasks

- ‘NUM’ Hier steht die Nummer eines Prozesses, sofern dieser sich mit Hilfe des Befehles **run** abgekoppelt hat oder noch in einer Shell läuft. Ein Programm, das über die Workbench gestartet wurde, hat als ‘NUM’-Eintrag einen Strich, wie auch ein Programm, das sich selbständig von der Shell abgekoppelt hat.
- ‘State’ Dieser Eintrag zeigt den Zustand eines Tasks/Prozesses an. Der eigene Prozess von **Scout**, der ganz oben in der Liste zu finden ist, hat dort immer ‘run’ stehen, weil er immer aktiv ist, wenn er die Task-Liste ausliest. =:~)  
 Ein ‘wait’ bedeutet hierbei, daß ein Task auf ein bestimmtes Ereignis wartet. Dies kann zum Beispiel das Betätigen eines Gadgets sein.  
 Sollte sich ein Task im Zustand ‘ready’ befinden, dann hat er zwar gerade etwas zu tun, wurde aber von der Abarbeitung eines anderen Prozesses unterbrochen (Multitasking-Prinzip).
- ‘SigWait’ Signalmaske, auf die der Task wartet. Sollte ein Task im Zustand ‘wait’ sein und diese Signalmaske den Wert Null (\$00000000) haben, dann handelt es sich mit großer Wahrscheinlichkeit um einen Task, der sich ‘aufgehängt’ hat und vom Betriebssystem in der Schwebe gehalten wird. (‘suspend’ or ‘reboot’)

## Aktionen

- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der ‘Tasks’ zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Freeze’ Hiermit wird ein Task *eingefroren*. Er befindet sich zwar dann noch in der Task-Liste, bekommt aber keine Rechenzeit mehr vom System.  
**Achtung:** Wenn Sie versuchen Tasks einzufrieren, die für das System *lebenswichtig* sind (wie z.B. der Task ‘input.device’), sollten Sie alle wichtigen Daten abgespeichert haben, da durch den folgenden Systemabsturz diese Daten sonst verloren sind.
- ‘Activate’ Ein eingefrorener Task kann hiermit wieder aktiviert werden.
- ‘CPU’ Hier finden Sie ein Textfeld und ein Cycle-Gadget. Das Textfeld gibt — abhängig von dem Zustand des Cycle-Gadgets — die verbrauchte CPU-Auslastung in Prozent an.  
 Für das Cycle-Gadgets gibt es drei Zustände:
- ‘off’ In diesem Zustand wird die CPU-Auslastung nicht berechnet.
  - ‘full’ Wurde dieser Zustand gewählt, dann setzt **Scout** die verbrauchte CPU-Auslastung auf 100%, d.h. die Summe der CPU-Auslastungsprozente aller in der Liste stehenden Tasks und Prozesse ergibt immer 100%. Dies ist unabhängig von der wirklich verbrauchten Rechenzeit.
  - ‘in %’ In diesem Fall wird die wirklich verbrauchte CPU-Auslastung gemessen und in dem dafür vorgesehenen Textfeld angegeben. Dafür startet **Scout** den Task ‘<< Scout’s cheat task >>’, der mit der Priorität -128 die ganze nicht verbrauchte Prozessorzeit beansprucht.
- ‘Secs’ Mit Hilfe dieses String-Gadgets können Sie bestimmen, in welchen Intervallen die CPU-Auslastung gemessen wird, sofern Sie diese Funktion beim Cycle-Gadget mittels ‘full’ oder ‘in %’ überhaupt ausgewählt haben. Dieses Intervall sollte nicht zu klein gewählt werden, da es zu Ungenauigkeiten kommen kann und **Scout** dann die meiste Rechenzeit beansprucht. Intervalle kleiner 0.5 Sekunden machen nicht viel Sinn!
- ‘Update’ Die Liste der Tasks und Prozesse wird erneut eingelesen.

- ‘Remove’ Ein Task wird aus der Liste entfernt. Sollten Sie sich nicht ganz sicher sein, ob Sie den Task noch einmal brauchen, dann sollten Sie lieber die ‘Freeze’-Funktion benutzen. (Siehe auch ‘Break’!)
- ‘Signal’ Sie können beim Benutzen dieser Funktion eine Signalmaske angeben, die darauf dem ausgewählten Task geschickt wird.
- ‘Break’ Einem Task wird ein Break-Signal gesendet. Viele Tasks reagieren auf dieses Signal und beenden sich selbst. Reagiert der Task, der mit Hilfe von Scout aus dem System entfernt werden soll, auf dieses Signal, dann sollte er normalerweise den von ihm angeforderten Speicher wieder freigeben. Wird ein Task durch die ‘Remove’-Funktion entfernt, wird der von ihm benutzte Speicher nicht wieder freigegeben. Es bleiben dann sogenannte ‘Speicherleichen’ im System zurück.
- ‘Priority’ Die Priorität eines Tasks kann hiermit verändert werden. Ein Task mit einer niedrigen Priorität bekommt erst vom System Rechenzeit zur Verfügung gestellt, wenn kein Task mit einer höheren Priorität Rechenzeit benötigt.
- ‘More’ Ein weiteres Fenster wird geöffnet, das, je nachdem ob ein Task oder ein Prozess selektiert wurde, weitere Informationen zu dem Task oder dem Prozess beinhaltet.
- ‘Exit’ Das Fenster mit der Task-Liste wird geschlossen.

## 4.17 Vectors (Spezielle Vektoren)

### Aktionen

- ‘Update’ Die Vektoren werden erneut ausgelesen.
- ‘Print’ Mit Hilfe dieser Funktion können Sie die Liste der Vektoren zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
- ‘Exit’ Das ‘Vectors’-Fenster wird geschlossen.

### Reset Vectors

Mit Hilfe der Reset-Vektoren kann sich ein Programm reset-fest ins System einhängen. Sie haben einen Wert von Null, wenn sie nicht verbogen wurden. Benutzt ein Programm die Kick-Vektoren (KickTagPtr, KickMemPtr und KickChecksum) um sich reset-fest zu machen, dann ist es auch in der Liste der residenten Strukturen zu finden. Siehe auch Abschnitt 4.13 [Residents], Seite 16.

### Auto Vector Interrupts

Die sieben Auto-Vektor-Interrupts, die hier angezeigt werden, sind bei einem System mit MC68000-Prozessor von Adresse \$64 bis \$7c zu finden. Die Prozessoren MC68010 und aufwärts besitzen ein Vektor-Basis-Register (VBR), das eine Verlegung der Interrupt-Tabelle ins FAST-RAM ermöglicht. Durch diese Verlegung ins FAST-RAM wird das System etwas beschleunigt. Scout berücksichtigt das VBR bei der Darstellung dieser Vektoren, vorausgesetzt es ist vorhanden und wird benutzt.

## Interrupt Vectors

Die hier angezeigten 16 Interrupt-Vektoren (IntVecs) befinden sich in der ExecBase-Struktur (der Basisstruktur der exec.library). Welche Aufgabe sie haben bzw. wie das Zusammenspiel der Auto-Vektor-Interrupts, der Interrupt-Vektoren und der Interrupt-Handler bzw. Interrupt-Server (siehe Abschnitt 4.6 [Interrupts], Seite 11) funktioniert, entnehmen Sie bitte der Fachliteratur.

### 4.18 Windows (Fenster)

In dieser Liste werden alle Screens mit den auf ihnen befindlichen Fenstern angezeigt. Screens werden andersfarbig dargestellt, damit sie sich besser von den Fenstern unterscheiden.

#### Spalteneinträge

'Pos(x,y)'	Horizontale (X) und vertikale (Y) Position des Screens/Fensters
'Size(x,y)'	Horizontale (X) und vertikale (Y) Größe des Screens/Fensters
'Title'	Titel des Screens/Fensters

#### Aktionen

'Update'	Die Liste wird erneut eingelesen.
'Print'	Mit Hilfe dieser Funktion können Sie die Liste der Fenster zum Drucker schicken oder in eine Datei Ihrer Wahl ausgeben lassen.
'Close'	Ihnen wird hiermit die Möglichkeit gegeben, Fenster und Screens zu schließen. Ein Screen wird dann mit all den Fenstern geschlossen, die sich auf ihm befinden.
'ToFront'	Das von selektierte Element der Windows- und Screensliste wird in den Vordergrund geholt.
'More'	Je nachdem, ob ein Screen oder ein Fenster in der Liste selektiert wurde, wird ein weiteres Fenster geöffnet, das weitere Daten zum Screen oder zum Fenster enthält.
'Exit'	Das 'Windows'-Fenster wird geschlossen.

### 4.19 Scout und AmiTCP

Dieser Abschnitt soll Ihnen kurz erläutern, was Sie machen müssen, um Ihren Rechner durch Scout und AmiTCP von einem anderen Rechner aus beeinflussen zu können.

Es werden hier bestimmte Kenntnisse zu AmiTCP vorausgesetzt. Wenn Sie mit diesem Thema absolut nichts anfangen können, dann können Sie der Anleitung von AmiTCP entnehmen, was es alles damit auf sich hat! (Siehe auch Abschnitt 3.3 [AmiTCP], Seite 5.)

Das Programm Scout dient unter AmiTCP als Client und als Server. Demnach brauchen Sie also neben der AmiTCP-Installation kein zusätzliches Programm, um Scout zusammen mit AmiTCP benutzen zu können.

Möchten Sie Ihren Rechner einem anderen System via Scout zugänglich machen, dann müssen Sie die nun folgenden zwei Schritte ausführen:

1. Fügen Sie dem File 'AmiTCP:db/services' die Zeile 'scout 6543/tcp' hinzu.
2. Jetzt fügen Sie bitte dem File 'AmiTCP:db/inetd.conf' die Zeile 'scout stream tcp nowait root dh0:scout' hinzu. Hierbei ist zu beachten, daß unter dem Pfad am Ende der Zeile wirklich das Programm Scout zu finden ist. Korrigieren Sie ggf. diesen Pfad in der Textzeile!

Das war's! Wenn Sie nun **AmiTCP** starten, dann ist Ihr Rechner prinzipiell von anderen System aus über **Scout** unter Verwendung der Optionen 'HOST', 'USER' und 'PASSWORD' erreichbar.

**Beispiel:** Wenn ich die Systemstrukturen meines Rechners von einem anderen System aus warten möchte. Dann müßte ich (natürlich mit einem anderen Passwort) **Scout** wie folgt aufrufen:

```
1> scout HOST crash.north.de USER atte PASSWORD secret
```

Wird die Option 'PASSWORD' weggelassen, dann werden Sie nachträglich aufgefordert, das Passwort in der Shell einzugeben. Diese Variante ist sicherer, falls Sie nicht allein sind und Ihr Passwort nicht preisgeben möchten, da das Passwort, das Sie in der Shell eingeben, nicht dargestellt wird.

Auch die Option 'USER' kann weggelassen werden. In diesem Fall nimmt **AmiTCP** an, daß derselbe Username verwendet werden soll, unter dem Sie sich derzeit in Ihrem System aufhalten.

Auch bei der Verwendung von **AmiTCP** sind Sie nicht daran gebunden **MUI** installiert zu haben. Alle Shell-Befehle (siehe Abschnitt 6.2 [ARexx- und Shell-Befehle], Seite 26) können auch zusammen mit **AmiTCP** verwendet werden.

**Beispiel:** Möchte ich z.B. die aktuelle Taskliste meines Rechners von einem anderen System aus ausgeben lassen. Dann müßte ich (natürlich wieder mit einem anderen Passwort) **Scout** wie folgt aufrufen:

```
1> scout HOST crash.north.de USER atte PASSWORD secret Tasks
```

Um die Angabe des korrekten Passwortes kommen Sie, wie jeder andere Benutzer, in 'keinem' Fall herum. Jeder, der Ihr System durch **Scout** beeinflussen möchte, muß ein Login auf Ihrem Rechner haben und sich korrekt identifizieren. Desweiteren gibt es bei **AmiTCP** durch einen Eintrag in der Datei '**AmiTCP:db/inet.access**') auch die Möglichkeit, bestimmte Services für beliebige Systeme zu sperren. Wenn Sie mehr darüber wissen möchten, dann sollten Sie sich die Anleitung von **AmiTCP** mal ein wenig genauer zu Gemüte führen. =;^)

Um weitere Informationen über die Optionen bzw. die durch **Scout** benutzbaren Befehle zu erhalten, siehe auch Kapitel 5 [Optionen], Seite 23 und Abschnitt 6.2 [ARexx- und Shell-Befehle], Seite 26.

## 4.20 Scout ohne MUI

**Scout** bietet dem Benutzer die Möglichkeit, fast alle über die grafische Benutzungsoberfläche angebotenen Funktionen auch über die Shell zu verwenden, wobei **MUI** von **Scout** dann natürlich nicht benötigt wird.

Demzufolge müssen Sie **MUI** nicht unbedingt installiert haben, um **Scout** benutzen zu können! Wenn Sie allerdings eine grafische Benutzungsoberfläche bevorzugen, kommen Sie bei **Scout** nicht um **MUI** herum.

## 5 Optionen

Für das Programm stehen ein paar Optionen zur Verfügung, die Sie benutzen können, wenn Sie das Programm starten. Diese Optionen können als Shell-Parameter oder als Tool Types von der Workbench benutzt werden. Dieser Abschnitt soll Ihnen den Verwendungszweck der Optionen erläutern.

**Beispiel:** In einer Shell werden die Optionen wie folgt benutzt:

```
1> scout option(s)
```

‘ICONIFIED’

**Format:** ICONIFIED

Wird diese Option verwendet, dann startet **Scout** iconifiziert.

‘PORTNAME’

**Format:** PORTNAME=*portname*

Der ARexx-Port von **Scout** kann mit Hilfe dieser Option in *portname* umbenannt werden. Wird diese Option nicht benutzt, dann bekommt der ARexx-Port von **Scout** den Namen ‘SCOUT.X’, wobei das ‘X’ die Nummer der **Scout**-Inkarnation angibt.

‘TOOLPRI’

**Format:** TOOLPRI=*value*

Diese Option erlaubt es Ihnen, die Task-Priorität von **Scout** auf einen bestimmten Wert *value* zu setzen. Dieser Wert *value* darf nur Werte von -128 bis 127 annehmen.

‘STARTUP’

**Format:** STARTUP=*scriptname*

Benutzen Sie diese Option, dann wird das ARexx-Skript *scriptname* ausgeführt, wenn **Scout** gestartet wird.

Auf diese Weise kann zum Beispiel bei jedem Start des Programmes das ‘Tasks’-Fenster automatisch geöffnet werden. Dafür braucht das ARexx-Skript nur den Befehl ‘OpenWindow Tasks’ zu beinhalten.

‘INTERVALTIME’

**Format:** INTERVALTIME=*time*

Diese Option erlaubt es, die Intervallzeit einzustellen, an der die Liste der Tasks regelmäßig erneuert wird, wenn die CPU-Funktion gewählt wurde.

‘CPUDISPLAY’

**Format:** CPUDISPLAY=*value*

Durch die Variable *value* ist es möglich den Zustand des Cycle-Gadgets, das sich im ‘Tasks’-Fenster befindet, einzustellen. (Siehe auch Abschnitt 4.16 [CPU], Seite 18.)

‘1’ bedeutet ‘CPU: full’

‘2’ bedeutet ‘CPU: in %’

‘HOST’

**Format:** HOST=*hostname*

Möchten Sie via **AmiTCP** auf einen anderen Rechner zugreifen, dann geben Sie hier bitte als *hostname* den Namen des gewünschten Rechners an.

‘USER’

**Format:** USER=*username*

Diese Option dient dazu, um mit *username* den Namen des Accounts auszuwählen, über den Sie die Systemstrukturen des anderen Rechners verwalten möchten.

‘PASSWORD’

**Format:** PASSWORD=*password*

Hier sollten Sie das notwendige Passwort angeben, das für das Einloggen an dem anderen Rechner notwendig ist.



**‘COMMAND’**

**Format:** COMMAND=*commandline*

Diese Option, die als Shell-Option auch ohne das Schlüsselwort ‘COMMAND’ benutzt werden kann, bietet Ihnen die Möglichkeit, einen der vielen Befehle zu benutzen, die Scout Ihnen via ARexx und Shell zur Verfügung stellt.

Siehe auch Abschnitt 6.2 [ARexx- und Shell-Befehle], Seite 26.

**‘SORTTASKSBYNAME’**

**Format:** SORTTASKSBYNAME

Wird diese Option benutzt, dann sortiert Scout die Liste der Tasks und Prozesse nach dem Alphabet.

**‘SINGLEWINDOWS’**

**Format:** SINGLEWINDOWS

Diese Option sorgt dafür, daß jeweils nur ein Listenfenster und ein Detailfenster geöffnet sind. Leute, die nicht gewohnt sind, mit vielen Fenstern zu arbeiten, werden diese Option wohl zu schätzen wissen. =^)

## 6 ARexx- und Shell-Befehle

Bei **Scout** gibt es zwei Arten von Befehlen:

1. Befehle, die nur als Shell-Parameter von **Scout** zur Verfügung stehen
2. Befehle, die zusätzlich auch über die ARexx-Schnittstelle aufgerufen werden können

### ARexx-Schnittstelle:

MUI gibt jeder seiner Applikationen automatisch eine ARexx-Port (ARexx-Schnittstelle). Demnach besitzt **Scout** also auch einen ARexx-Port, der normalerweise den Namen `'SCOUT.X'` hat, wobei das `'X'` die Nummer der Programm-Inkarnation angibt.

Der jeweilige Name des ARexx-Ports jeder **Scout**-Inkarnation wird auch in dem Fenster angezeigt, welches Sie durch die Auswahl des `'Project/About'`-Menüpunktes erhalten.

### Verwendung von Tasknamen:

Ein Task oder ein Prozess, der von einer Shell aus gestartet wurde und sich nicht abgekoppelt hat, hat meistens einen Namen wie `'Background CLI'` oder `'CLI Process'`. **Scout** verwendet in der Task-Liste in einem solchen Fall nicht den 'richtigen' Namen des Tasks, sondern den Namen des jeweils ausgeführten Programmes.

**Beispiel:** Starten Sie zum Beispiel das Programm `DH0:Debug/Sushi` ohne den Befehl `run`, dann wird bei **Scout** als Taskname `'DH0:Debug/Sushi'` angezeigt.

Einige Befehle von **Scout** erwarten als Parameter auch einen Tasknamen. Dieser Taskname muß auf die gleiche Weise angegeben werden, wie er bei **Scout** angezeigt wird.

### Verwendung von Adressen:

Viele der folgenden Befehle benötigen als Parameter die Adressen bestimmter Strukturen. Diese Adressen können als hexadezimale Zahlen mit im Befehlsaufruf angegeben werden.

**Beispiel:** Die folgenden drei Aufrufe sind syntaktisch korrekt:

1. `'scout FreezeTask AmiTCP:AmiTCP'`
2. `'scout FreezeTask 0x00204508'`
3. `'scout FreezeTask $00204508'`

Der erste Aufruf friert den Prozess `'AmiTCP:AmiTCP'` ein, sofern dieser überhaupt vorhanden ist. Die beiden anderen Aufrufe können nur erfolgreich ausgeführt werden, wenn jeweils ein Task existiert, der an der Adresse `'$00204508'` im System zu finden ist.

## 6.1 Befehle via Shell

`'Help'`

### **Format:** Help

Dieser Befehl **Help**, der keine Parameter benötigt, ist wohl der wichtigste der nun folgenden Befehle. Er veranlaßt **Scout**, die Liste der verfügbaren Befehle auszugeben. `=:^)`

Die nun folgenden 18 Befehle sind dazu da, dem Benutzer alle Listen, die **Scout** anbietet, auch in der Shell auszugeben. Dadurch ist es nicht mehr unbedingt erforderlich, MUI zu installieren, wenn man **Scout** benutzen möchte. Möchte man allerdings die vielen Fenster von **Scout** benutzen, kommt man um MUI nicht herum!

Für jeden dieser Befehle steht auch eine Kurzform zur Verfügung, die jeweils hinter dem Befehl in Klammern zu finden ist.

Hier also die Befehle, die jeder für sich eine Liste ausgeben:

**Assigns** (a), **Commands** (c), **Devices** (d), **Expansions** (e), **Fonts** (f), **InputHandlers** (h), **Interrupts** (i), **Libraries** (l), **Memory** (m), **Mounts** (n), **Locks** (o), **Ports** (p), **Residents** (r), **Semaphores** (s), **Tasks** (t), **Resources** (u), **Vectors** (v) und **Windows** (w)

**Beispiel:** Um die Liste der Ports in der Shell auszugeben, müssen Sie einfach in der Shell `'scout ports'` oder `'scout p'` eingeben.

## 6.2 Befehle via ARexx und Shell

Dieser Abschnitt stellt die Befehle vor, die als ARexx-Befehl und als Shell-Parameter zur Verfügung stehen.

**'FindTask'**

**Format:** `FindTask task`

Mit diesem Befehl kann festgestellt werden, ob ein bestimmter Task im System vorhanden ist. Er liefert als Ergebnis die Adresse des Tasks *task*, sofern dieser gefunden wurde. Als Variable *task* kann entweder der Name eines Tasks oder eine Adresse angegeben werden.

**'FreezeTask'**

**Format:** `FreezeTask task`

Der Task *task* wird von Scout eingefroren. Er ist danach zwar noch in der Task-Liste zu finden, bekommt aber keine Rechenzeit mehr vom System. Die Variable *task* entspricht einem Tasknamen oder der Adresse eines Tasks.

**'ActivateTask'**

**Format:** `ActivateTask task`

Der eingefrorenen Task *task* kann durch diesen Befehl wieder aktiviert werden. Für die Variable *task* ist ein Taskname oder eine Adresse zu wählen.

**'RemoveTask'**

**Format:** `RemoveTask task`

Mit diesem Befehl wird der Task mit dem Namen oder der Adresse *task* unwiderruflich aus dem System entfernt.

**'BreakTask'**

**Format:** `BreakTask task`

Dem Task *task* wird mit Hilfe dieses Kommandos ein Signal geschickt, das dem Drücken von CTRL-C bzw. CTRL-D entspricht. Viele Programme reagieren auf dieses Signal, indem sie sich selbständig beenden. Als Variable *task* kann entweder der Name eines Tasks oder eine Adresse angegeben werden.

**'SignalTask'**

**Format:** `SignalTask task hexsignal`

Hiermit kann dem Task *task* ein gewähltes Signal *hexsignal* (bzw. eine Signalmaske) zugeschickt werden. Dieses Signal muß als Hexadezimalzahl (mit vorangestelltem '0x' oder '\$') angegeben werden.

**Beispiel:** Das Kommando `'SignalTask scout 0x001000'` sendet dem Scout-Prozess ein CTRL-C, worauf dieser sein Dasein beendet.

**'SetTaskPri'**

**Format:** `SetTaskPri task priority`

Der Task *task* bekommt mit Hilfe dieses Befehles die Priorität *priority*. Die Variable *task* entspricht einem Tasknamen oder der Adresse eines Tasks.

**'RemovePort'****Format:** RemovePort *port*

Der Port *port* wird von Scout aus dem System entfernt. Für *port* kann entweder der Name des zu entfernenden Ports oder dessen Adresse gewählt werden.

**'GetLockNumber'****Format:** GetLockNumber *lockpattern*

Dieses Kommando gibt die Anzahl der Lock-Einträge zurück, deren Pfade mit dem Namensmuster *lockpattern* übereinstimmen. So kann über ARexx nachgeschaut werden, ob noch auf ein bestimmtes File zugegriffen wird.

**'RemoveLocks'****Format:** RemoveLocks *lockpattern*

Alle Locks werden aus dem System entfernt, deren Pfade mit dem Namensmuster *lockpattern* übereinstimmen. Bei diesem Kommando ist höchste Vorsicht geboten! Will ein Programm einen Lock entfernen, der schon von Scout entfernt wurde, dann stürzt mit großer Wahrscheinlichkeit der Rechner ab.

**'RemoveLock'****Format:** RemoveLock *lockaddress*

Der Lock mit der Adresse *lockaddress* wird aus dem System entfernt.

**'FindNode'****Format:** FindNode *nodetype nodename*

Dieser Befehl erlaubt es Ihnen, eine Struktur *nodename* zu finden, die einen bestimmten Nodetypen *nodetype* besitzt.

Die Variable *nodetype* kann folgende Werte haben: 'LIBRARY', 'DEVICE', 'RESOURCE', 'MEMORY', 'SEMAPHORE', 'PORT' oder 'INPUTHANDLER'.

**Beispiel:** Wenn Sie die Adresse der 'dos.library' bekommen möchten, müssen Sie den Befehl wie folgt aufrufen:

```
FindNode LIBRARY 'dos.library'
```

**'GetPriority'****Format:** GetPriority *nodeaddress*

Dieser Befehl liefert die Priorität einer Struktur, die folgenden Typ haben kann: Task, Library, Device, Resource, Port, Resident, Inputhandler, Interrupt, Semaphor oder ein Element der Memory-List.

Die Struktur müssen Sie dabei durch ihre Adresse *nodeaddress* auswählen, die Sie z.B. durch das ARexx-Kommando FindNode erhalten.

**Beispiel:** Die folgenden ARexx-Befehle beschaffen die Priorität Ihres Grafik-Speichers und legen sie in der Variablen *pri* ab:

```
FindNode MEMORY 'chip memory'
addr = result
GetPriority addr
pri = result
```

**'SetPriority'****Format:** SetPriority *nodetype nodename priority*

Wenn Sie die Priorität einer Struktur *nodename* ändern möchten, können Sie dafür dieses Kommando benutzen. Wiederum kann die Variable *nodetype* folgende Werte haben: 'LIBRARY', 'DEVICE', 'RESOURCE', 'MEMORY', 'SEMAPHORE', 'PORT' oder 'INPUTHANDLER'. Die Variable *priority* muß dafür von Ihnen die Priorität bekommen, die die Struktur *nodename* bekommen soll.

**'CloseLibrary'****Format:** CloseLibrary *library*

Die von Ihnen mittels der Variablen *library* ausgewählte Library wird einmal geschlossen. Die Variable *library* sollte dafür mit dem Namen oder der Adresse der zu schließenden Library versehen werden.

**'RemoveLibrary'****Format:** RemoveLibrary *library*

Die durch ihren Namen oder ihre Adresse ausgewählte Library *library* wird geschlossen.

**'RemoveDevice'****Format:** RemoveDevice *device*

Das durch seinen Namen oder seine Adresse ausgewählte Device *device* wird geschlossen.

**'RemoveResource'****Format:** RemoveResource *resource*

Die durch ihren Namen oder ihre Adresse ausgewählte Resource *resource* wird geschlossen.

**'ObtainSemaphore'****Format:** ObtainSemaphore *semaphore*

Hierdurch wird dem System vorgegaukelt, daß das Gerät, das File oder wofür der Semaphor *semaphore* sonst eingerichtet wurde, von einem Programm mehr benutzt wird, als vorher. Die Variable *semaphore* kann dabei entweder den Namen oder die Adresse des Semaphors enthalten.

**'ReleaseSemaphore'****Format:** ReleaseSemaphore *semaphore*

Sollte ein Semaphor gerade benutzt werden, so machen Sie dem System mit dieser Funktion weis, daß ein Programm weniger das dem Semaphor entsprechende Gerät benutzt. Ein Programm, das den Semaphor beachtet, kann so eventuell versuchen, ein weiteres Mal auf das entsprechende Gerät zuzugreifen.

**'RemoveSemaphore'****Format:** RemoveSemaphore *semaphore*

Der durch seinen Namen oder seine Adresse ausgewählte Semaphor *semaphore* wird mit Hilfe dieses Befehles aus dem System entfernt.

**'RemoveInpuhandler'****Format:** RemoveInpuhandler *inputhandler*

Der Inpuhandler *inputhandler*, den sie durch seinen Namen oder seine Adresse ausgewählt haben, wird aus dem System entfernt.

**'FindResident'****Format:** FindResident *resident*

Mit diesem Befehl kann festgestellt werden, ob eine bestimmte residente Struktur im System vorhanden ist. Er liefert als Ergebnis die Adresse der residenten Struktur *resident*, sofern diese gefunden wurde. Als Variable *resident* kann entweder der Name oder die Adresse einer residenten Struktur angegeben werden.

**'FindInterrupt'****Format:** FindInterrupt *interruptname*

Dieser Befehl dient dazu, einen bestimmten Interrupt mit dem Namen *interruptname* zu finden. Wird der Interrupt gefunden, so wird seine Adresse zurückgeliefert.

**'RemoveInterrupt'****Format:** RemoveInterrupt *interruptname*

Der Interrupt *interruptname* wird aus dem System entfernt.

**'FlushDevs'****Format:** FlushDevs

Sollten sich noch Devices im System bzw. im Speicher befinden, die im Augenblick von keinem Programm mehr benötigt werden, so werden sie aus dem Speicher entfernt.

**'FlushFonts'**

**Format:** FlushFonts

Unbenutzte Zeichensätze, die von Diskette bzw. Festplatte nachgeladen wurden und nicht mehr benötigt werden, werden aus dem Speicher entfernt.

**'FlushLibs'**

**Format:** FlushLibs

Sollten sich noch Libraries im System/im Speicher befinden, die im Augenblick von keinem Programm mehr benötigt werden, so werden sie aus dem Speicher entfernt.

**'FlushAll'**

**Format:** FlushAll

Diese Funktion beinhaltet die Funktionen **FlushDevs**, **FlushFonts** und **FlushLibs**. Dementsprechend werden Devices, Libraries und Zeichensätze, die zur Zeit von keinem Programm benutzt werden, aus dem Speicher entfernt.

**'ClearResetVectors'**

**Format:** ClearResetVectors

Bei Gebrauch dieser Funktion werden die sechs Reset-Vektoren gelöscht (siehe auch Abschnitt 4.17 [Vectors], Seite 20).

**'PopToFront'**

**Format:** PopToFront *winscr*

Der Screen oder das Fenster *winscr* werden in den Vordergrund gebracht. Die Variable *winscr* kann entweder den Title des Screens/Fensters oder die Adresse des Screens/Fensters enthalten.

**'CloseWindow'**

**Format:** CloseWindow *window*

Das Fenster mit dem Titel oder der Adresse *window* wird geschlossen.

**'CloseScreen'**

**Format:** CloseScreen *screen*

Der Screen mit dem Titel oder der Adresse *screen* wird geschlossen.

**'CloseFont'**

**Format:** CloseFont *address*

Der Zeichensatz mit der Adresse *address* wird einmal geschlossen.

**'RemoveFont'**

**Format:** RemoveFont *address*

Der Zeichensatz mit der Adresse *address* wird aus dem System entfernt, sofern er von keinem Programm mehr benutzt wird bzw. oft genug geschlossen wurde.

**'RemoveCommand'**

**Format:** RemoveCommand *address*

Der residente Befehl mit der Adresse *address* wird aus dem System entfernt.

**'RemoveAssign'**

**Format:** RemoveAssign *name*

Mit Hilfe dieses Befehles wird der Assign mit dem Namen *name* aus dem System entfernt.

**'RemoveAssignList'**

**Format:** RemoveAssignList *name address*

Dieser Befehl sorgt dafür, daß das Verzeichnis mit der Adresse *address* von dem Assign mit dem Namen *name* entfernt wird.

**'OpenWindow'**

**Format:** OpenWindow *windowid*

Mit diesem Kommando sind Sie in der Lage, alle Fenster über ARexx zu öffnen, die über das Hauptfenster von **Scout** durch das Betätigen eines Gadgets geöffnet werden können.

Die Fensteridentifikation *windowid* besteht aus dem gleichen Text, der auch auf den Gadgets im Hauptfenster zu finden ist.

**Beispiel:** Wird das Kommando ‘**OpenWindow ’Resident Cmds’**’ zu **Scouts** ARexx-Port geschickt, dann wird das Fenster mit der Liste der residenten Befehle geöffnet.

Sollte das Fenster schon geöffnet worden sein, dann wird es nach vorn geholt, und die jeweilige Liste wird neu eingelesen.

Aus der diesem Befehl zugedachten Aufgabe wird ersichtlich, daß dieser Befehl keinerlei Wirkung hat, sollte er als Shell-Parameter aufgerufen worden sein. Die grafische Oberfläche von **Scout** steht dort eben nicht zur Verfügung.

## Anhang A

### Wie und wo bekommt man Updates?

Die neueste Version von Scout sollte immer im "DEEP THOUGHT BBS" (siehe unten) erhältlich sein. Zusätzlich wird sie immer recht fix auf dem AmiNet landen, und dadurch ist sie etwas später in aktuelleren Public Domain Sammlungen zu finden.

### Support BBS

DEEP THOUGHT Bulletin Board System, Oldenburg, Germany

Node 1

+49-(0)441-383365    1200-21600 bps    v.32terbo, v.42bis

Node 2

+49-(0)441-383839    1200-19200 bps    v.32bis, v.42bis, ZyXEL

	Node 1	Node 2
FidoNet	2:2426/2020.0	2:2426/2021.0
AmigaNet	39:170/204.0	39:170/205.0

InterNet            cosinus@deepthought.north.de

Beide Nodes sind 24 Stunden am Tag online und auf beiden Nodes läuft ein FidoNet-Mailer, der Fido-File-Requests akzeptiert.

Benutzen Sie das Magic SCOUT für die neueste Version von SCOUT  
oder                                FILES für eine komplette Fileliste

### Wem ich zu danken habe

Nun habe ich noch ein paar Leuten zu danken, die mir bei der Entwicklung von Scout auf die unterschiedlichsten Weisen behilflich waren, als da wären:

- Klaus 'gizmo' Weber, der dieses Programm ein wenig unter die Lupe genommen hat und für meine Probleme bei der Entwicklung von Scout (es waren nicht wenige) meist ein freies Ohr hatte
- Christian 'cosinus' Stelter, der mir erlaubt hat, seine ganzen Manuals zu benutzen
- Stefan Stuntz für sein MagicUserInterface
- den ganzen Leuten, die Scout getestet und mir Bugs oder neue einzubauende Features gemeldet haben und es noch tun: Kai 'wusel' Siering, Martin Hauner, Peter Meyer, Karl 'Charly' Skibinski, Michael 'Mick' Hohmann, Thore Böckelmann, Bernardo Innocenti, ... und zum guten Schluß
- all den anderen, die ich evtl. vergessen habe, die mir Bugs, Anregungen und konstruktive Kritik zu Gehör gebracht haben.



## Wie erreicht man den Autor?

Wenn Sie Fragen, Verbesserungsvorschläge, Bug Reports oder Dinge dieser Art haben, dann können Sie mich unter den folgenden EMail-Adressen erreichen:

atte@crash.north.de (Andreas Gelhausen)  
oder  
2:2426/2020.24 (im FidoNet)

Wenn Sie nicht über die Möglichkeit verfügen, mich über die oben angegebenen EMail-Adressen zu erreichen, dann können Sie mir natürlich auch 'normale' Briefe schreiben.

Hier meine Adresse:

Andreas Gelhausen  
Graf Spee Str. 23b  
26123 Oldenburg  
- Germany -

Das war's! =: ^)

# Stichwortverzeichnis

Adresse des Autors .....	32	Mounted Devices .....	14
Adressen, Verwendung von .....	25	MUI .....	5
AmiTCP .....	5	Nutzungsgebühren .....	3
ARexx-Befehle .....	25	Optionen .....	23
ARexx-Schnittstelle .....	25	Ports .....	15
Assigns .....	7	Processes .....	18
Autor .....	32	Programmversion .....	31
Boards .....	9	Prozesse .....	18
CLI Optionen .....	23	RAM Pointer Count .....	8
Copyright .....	3	Rechtliche Dinge .....	3
Danksagungen .....	31	Resident Commands .....	15
DEEP THOUGHT BBS .....	31	Residente Befehle .....	15
Devices .....	8	Residente Strukturen .....	16
Disclaimer .....	3	Residents .....	16
DISKFONT .....	10	Ressources .....	17
Einleitung .....	1	Ressourcen .....	17
Ereignisse .....	10	ROMFONT .....	10
Erweiterungskarten .....	9	RPC .....	8
Expansions .....	9	Screens .....	21
Fenster .....	21	Semaphores .....	18
Festplatten .....	14	Semaphore .....	18
Fonts .....	10	Shell Optionen .....	23
Generelle Benutzung .....	7	Shell-Befehle .....	25
Giftware .....	3	Speichersegmente .....	13
Hardware .....	9	Support BBS .....	31
Hauptfenster .....	7	System-Erweiterungen .....	9
Hersteller .....	9	Systemanforderungen .....	5
Input Events .....	10	Tasknamen, Verwendung von .....	25
Inputhandler .....	10	Tasks .....	18
Installation .....	5	TCP/IP .....	5
Interrupts .....	11	Tool Types .....	23
Keine Garantie .....	3	Updates .....	31
Laufwerke .....	14	VBR .....	20
Libraries .....	12	Vectors .....	20
Locks .....	13	Vektoren .....	20
Logische Verzeichnisse .....	7	Vertical blank interrupt .....	11
MagicUserInterface .....	5	Was ist Scout? .....	1
Manufacturer .....	9	Windows .....	21
Memory .....	13	Zeichensätze .....	10



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung .....</b>	<b>1</b>
<b>2</b>	<b>Rechtliche Dinge.....</b>	<b>3</b>
<b>3</b>	<b>Vor dem Programmstart .....</b>	<b>5</b>
3.1	Systemanforderungen .....	5
3.2	MUI - MagicUserInterface .....	5
3.3	AmiTCP .....	5
3.4	Installation .....	5
<b>4</b>	<b>Wie wird Scout benutzt? .....</b>	<b>7</b>
4.1	Assigns .....	7
4.2	Devices .....	8
4.3	Expansions (System-Erweiterungen) .....	9
4.4	Fonts .....	10
4.5	Inputhandler .....	10
4.6	Interrupts .....	11
4.7	Libraries .....	12
4.8	Locks .....	13
4.9	Memory (Speichersegmente) .....	13
4.10	Mounted Devices .....	14
4.11	Ports .....	15
4.12	Resident Commands (Residente Befehle) .....	15
4.13	Residents (Residente Strukturen) .....	16
4.14	Resources (Ressourcen) .....	17
4.15	Semaphores (Semaphore) .....	18
4.16	Tasks .....	18
4.17	Vectors (Spezielle Vektoren) .....	20
4.18	Windows (Fenster) .....	21
4.19	Scout und AmiTCP .....	21
4.20	Scout ohne MUI .....	22
<b>5</b>	<b>Optionen .....</b>	<b>23</b>
<b>6</b>	<b>ARexx- und Shell-Befehle .....</b>	<b>25</b>
6.1	Befehle via Shell .....	25
6.2	Befehle via ARexx und Shell .....	26
<b>Anhang A</b>	<b>.....</b>	<b>31</b>
	Wie und wo bekommt man Updates? .....	31
	Wem ich zu danken habe .....	31
	Wie erreicht man den Autor? .....	32
<b>Stichwortverzeichnis</b>	<b>.....</b>	<b>33</b>

