0027b238-0

COLLABORATORS

	TITLE : 0027b238-0	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

0027b238-0

1.1 No title

```
                 Rohtexte für UMS-Tutorial.
                23.12.1996
```

sm@mk2.maus.sauerland.de

1.2 Inhalt

```
                Ein Workshop für alle, die UMS anwenden möchten:

  - User
  - Programmierer


Gliederung

Kapitel:                          (Arbeitstitel)


                A.
                 Einführung.


                B.
                 Grundlagen.

 Die Messagebase-Datei.

 Messagebase-Konfiguration und Verwaltung.

 UMS Anwenderprogramme.

 Tools für den UMS Anwender.
```

C.
Einstieg in die UMS-Programmierung

Hello World. Ein erstes UMS Programm.

Programmierung: ein Überblick.

Welche Sprache: Rexx, C, Modula?

Interna: umsserver, ums.library.

Hilfreich: Die umssupport.library.

Kleine Tools schnell programmiert.

D.
UMS-Funktionen Tutorials

Ein- und ausloggen.

Suchen und finden.

Lesen und schreiben.

Voreinstellungen holen und ablegen.

Fehler auswerten und dokumentieren.

E. Do it yourself.
Benutzeroberfläche.

Listenverwaltung.

Anhang

UMS-Ressourcen.

Literatur.

Ausblick.

In eigener Sache:

- Feedback-Formular.

1.3 A

A.

Einführung.

UMS bedeutet "Universal Message System".

UMS ist ein System um Nachrichten zu verwalten. Es benutzt ein einheitliches Format, welches netzunabhängig ist.

This means that UMS allows to treat and read all kinds of messages (’e-mail’ and ’news’) as universally and efficiently as possible.

The user should not need to care about what network a message comes from or goes to/through and what format is used, he or she should be able to concentrate totally on the messages’ contents.

In order to achieve this goal, UMS does two things:

1) define an universal format for messages where messages in all formats and from all networks can be stored in without loss of information and nevertheless being randomly interchangeable.

2) implement a central, network-independent database-mamagement that allows to store and read/retrieve messages in the UMS format as efficiently as possible.

Netzspezifische Programme, sogenannte Importer und Exporter, sorgen fuer die Umsetzung vom/ins UMS Nachrichtenformat. Dieses einheitliche Format bedeutet auch, dass die Newsreader und Hilfsprogramme unabhaengig von den Netzen implementiert werden können. Momentan gibt es Importer/Exporter fuer MausNet, FidoNet oder Netze mit Fido-Technologie, UUCP, QWK, Z-Netz (Zerberus und ZConnect) und diverse RFC-Protokolle.

UMS unterstuetzt mehrere Benutzer. Jeder Benutzer erhaelt gesonderte Zugriffsrechte, die von UMS automatisch ueberwacht werden. Ueber diese Zugriffsrechte laesst sich regeln, welche Gruppen der Benutzer lesen, in welche er schreiben und an welche Adressen er Mails schicken darf.

Importer und Exporter werden wie normale UMS Benutzer in der UMS.config eingetragen, haben aber besondere Zugriffsrechte.

Im Normalzustand kann ein Exporter fuer ein System nicht die Nachrichten von dem Importer eines anderen Systems lesen. Dies ist eine Sicherheitsbarriere, um die Entstehung von nicht autorisierten Gateways zu unterbinden. Erst durch Vergabe eines Schluessels wird UMS in dieser Hinsicht voll funktionsfaehig.

UMS stellt eine einheitliche Schnittstelle zur Verfuegung, z.B. die ums.library auf AmigaOS, damit die Programme unabhaengig von der Implementierung von UMS sind. *Alle* Aktionen geschehen ueber diese Schnittstelle.

UMS ist momentan nur auf Amiga Rechnern implementiert. Im Prinzip spricht aber nichts dagegen, es auch auf anderen Rechnern oder

Betriebssystemen zu implementieren.

1.4 B

B.

Die Messagebase-Datei.

Messagebase-Konfiguration und Verwaltung.

UMS Anwenderprogramme.

Tools für den UMS Anwender.

1.5 B1 Messagebase

Die Messagebase-Datei.

Messagebase ist der Ort, wo UMS eine Datei aller Nachrichten anlegt.

F: Nach der Installation von UMS wird meine Messagebase nicht gefunden, obwohl alle Pfade gesetzt sind. Änderungen an der UMS.config brachten auch keine Änderung.

A: Die ENV-Variable UMSMB.<Name der Messagebase> wurde nicht gesetzt. Hier muß der Pfad zum Messagebase-Verzeichnis eingetragen werden.

F: Ich benutze jetzt seit einigen Tagen UMS... meine MessageBase ist ca. 4MB gross. Nur zum Einscannen einer Newsgroup mit ca. 250 Nachrichten braucht IntuiNews sage und schreibe 1:20 Minuten. Ist das normal?

A: Das FFS des Amiga hat gewisse Probleme mit großen Files, in denen viel an verschiedenen Stellen mittendrin gelesen wird ("seek&read").

Es benötigt dafür nämlich sog. "file-extension blocks", die leider nur linear verkettet sind. Damit das ganze noch effizient funktioniert, sollte die jeweilige Partition genug "buffer" für alle benötigten file-extension blocks haben. Das sind die buffer, die man mit HDToolBox oder in der Mountlist konfigurieren und mit "C:addbuffers" nachträglich verändern kann.

Die folgende Tabelle zeigt für verschiedenen Blockgrößen, für wieviel KB Filegröße ein buffer reicht:

blocksize	one file-extension block each
512	36 KB
1024	200 KB
2048	912 KB
4096	3.87 MB
8192	15.94 MB
16384	64.64 MB

```
  32768    | 260.35 MB
```

Bei einer MB von 4MB und einer Partition mit den üblichen 512 Bytes/Block brauchst Du also mindestens 4MB / 36 KB = 114 buffer.

Wer ein neues HDToolBox hat oder fit im Schreiben von Mountlist-Einträgen ist, kann für die UMS-MB eine Partition mit größeren Blockgrößen verwenden. Das ist sehr empfehlenswert!

F: Welche Dateien der Messagebase müssen bei einem Backup mitgesichert werden?

A: Alle außer ".idcount". Diese Datei hat *nichts* mit der Konsistenz der Daten in der Messagebase zu tun. Wenn Dir der .idcount hopps geht oder Du ihn loescht oder sonstwas damit machst, legt UMSServer das File neu an und generiert eine neue, aktuelle MessageID (z.B. unter Beruecksichtigung des aktuellen Datums/Zeit). Die Messagebase wird dadurch *nicht* beschaedigt oder unbrauchbar. Wenn Du also ein Backup zurueckholst, in dem *kein* .idcount vorhanden ist, dann wird UMSServer dieses File neu generieren und Du kannst Deine Messagebase ohne weitere Probleme benutzen.

F: Könnte man die Messagebase auf einer muFS-Partition installieren, und nur dem UMSServer die write/delete-Rechte auf die Datenfiles erteilen?

A: Klar, funktioniert einwandfrei! Einfach eine muFS-User "news" (o.ä.) einrichten und l:umsserver mit dieser User-ID und dem U-Flag ausstatten.

F: Warum kann UMS nicht die gebräuchlichen Atrribute *fett*, /kursiv/ und _unterstrichen_ schon beim Schreiben einer Mitteilung in die Messagebase z.B. durch Amiga-Escape-Sequenzen oder auch MUI-Steuerzeichen ersetzen (je nach bevorzugtem Reader)?

A: Es kommt zu Problemen mit der Multiuserfähigkeit von UMS, da es möglich ist, daß die User, die auf die Mitteilung Zugriff haben, verschiedene Reader verwenden: sind die Attribute z.B. als Amiga-Escapes in der Messagebase, so muß IntuiNews bzw. MUI diese beim Anzeigen sowieso umsetzen. Genauso wäre es problematisch, eine Nachricht mit MUI-Attributen z.B. im RUMS anzuzeigen. Es sollte also Sache des anzeigenden Programms sein, die Attribute zu interpretieren.

1.6 B2

---Messagebase-Konfiguration und Verwaltung.

The implementation of the Message Base Processor ('MBP') is based on the server/client- concept. Clients address the server to get or put messages. The server manages the storage and retrieval of messages and controls the different client's access-rights to the system. The common interface between client and server is a set of functions described in this document.

Clients are

- simple USERS that read and write messages (using programs called 'newsreaders'),
- a special kind of user, the 'SYSOP', which has additional rights and tools to manage the system,
- IMPORTERS, that get messages form other systems and put them into the local system after converting them to the UMS format, and
- EXPORTERS, that look for all NEW messages in the system and send them to other systems after converting them into the specific format.

The MBP controls which user may read what message and, in the same way, which exporter needs to export or forward what message. It cares about whether a message can or cannot be correctly sent to its destination.

F: Nach der Installation von UMS wird meine Messagebase nicht gefunden, obwohl alle Pfade gesetzt sind. Änderungen an der UMS.config brachten auch keine Änderung.

A: Die ENV-Variable UMSMB.<Name der Messagebase> wurde nicht gesetzt. Hier muß der Pfad zum Messagebase-Verzeichnis eingetragen werden.

F: Nachdem mir gestern der Rechner beim Maillesen abgestuerzt war, musste ich nach dem Reset feststellen, dass meine MsgBase defekt ist. Ich hab dann mit UMSDoctor alles auf eine andere Partition gerettet. Anschließend habe ich die Nachrichten mit Import wieder in eine neue Messagebase eingefügt. Klappte auch hervorragend, nur dass jetzt scheinbar alle importierten Mails an mich adressiert zu sein scheinen, stoerte mich etwas (IntuiNews hob diese Mails farbig hervor). "Naja, solange das nur bei den alten Mails ist", dachte ich. Aber Pustekuchen: Nach dem Importieren neuer Mails wurden auch zahlreiche Mails von IntuiNews farbig hervorgehoben, auch wenn diese nicht an mich adressiert sind! Muss ich jetzt damit leben oder kann man das irgendwie beheben?

A: sumsset <YOURACCOUNT> <PASSWORD> "FromAddr!=*"*"" "" OWNER

F: Nachdem ich meine defekte MessageBase mit UMSDoctor (Version 1.3) gerettet und mit import wieder zurückges pielt habe, wie es im Readme zum UMSDoctor beschrie ben ist, sind die Exported- Flags verschwun den. Jetzt möchte ich also sämtliche Messages als exportier t markieren (denn sonst können [sollen] sie ja nicht expired werden). Das habe ich versucht mit

sumsset sysop passwd "exported=0" exported ""

... und sumsset meldet auch brav, zwölftausend Messages (oder so) gefunden zu haben, allerdings ist das exported-Flag hinterher trotzdem nicht gesetzt...

A: Exported ist ein Server-Flag, das von außen nicht geändert werden kann. SUMSSet bzw. der UMSServer sollten IMHO eine entsprechende Fehlermeldung erzeugen.

F: Cleanums funtioniert bei mir nicht/stürzt ab/macht die MB kaputt...

A: Wenn Du "UnixDirs" benutzt, wirf es weg. Leider weiß ich nicht, _was_ UnixDirs da macht und was sich der Autor dabei gedacht hat. Es muß jedenfalls ein übler Hack sein. Auch PFS macht (noch) Probleme. Allgemein müssen die FileSystems für die MB zumindest SetFileSize() unterstützen.

1.7 B3

---UMS Anwenderprogramme.

Newsreader =-=-=-=-=-

RUMS (2.02) [mh] [kai] Dieses Paket enthält einen Newsreader. Dieser ist nur über die Tastatur, aber dadurch auch remote zu bedienen. Es werden nicht alle Features von UMS unterstützt!

IntuiNews (1.4a) [tron] [ms] Dieses Paket enthält einen Newsreader mit graphischer Oberfläche. Dieser benötigt zusätzlich das Magic User Interface (MUI) Version >= 2.0.

READ (?.?) [tt] Dieses Paket enthält einen weiteren Newsreader mit graphischer Oberfläche. MUI wird ebenfalls benötigt. Öffentliche Beta-Version!

Mailboxen =-=-=-=-=

KMS (2.0) [ts] Das Karfunkel Mailbox System bietet folgende Fetaures: Kommandozeilenorientierte Steuerung, hierarchische Brettstruktur, interner Befehlssatz beliebig durch externe Befehle in Form von ARexx-Skripts erweiterbar, beliebige AmigaDOS-Befehle/-Programme einbindbar, beliebig aenderbare Systemtexte, eine Vielzahl an Zugriffskontrollmoeglichkeiten fuer den Systembetreiber, Ein- und Ausgaben komplett ueber die Standardein-/-ausgabe, eigenes Adressbuch, Laberfilter, Vormerker, Verzeichnis fuer Batchdateien fuer jeden User, Chat, Screeneditor, Lesen aller neuen Nachrichten mit nur einer Taste (Space oder Return) moeglich, Anzeige von Textattributen (*fett*, _unterstrichen_, /kursiv/) in Nachrichten, ...

1.8 B4

---Tools für den UMS Anwender.

Gruppentools =-=-=-=-=-=-

UMSMapsTool (1.4) [olio] Mit UMSMapsTool kann man als Z-Netz Point Bretter bestellen bzw. abbestellen. Benötigt MUI.

UMSGroup (2.11) [mk] [ml] UMSGroup ist ein Programm, um im MausNet Gruppen (Newsgroups) Offline zu bestellen/abzustellen. Benötigt MUI.

UMSFix (0.8) [ps] UMSFix ist ein Areafix für Fido-kompatible Netze. Er nimmt Area-Bestellungen der User entgegen und bestellt diese gegebenenfalls beim

Uplink.

UMSFeeder (0.31b) [msi] UMSFeeder ist ein Tool, um in RFC-Netzen Newsgroups zu bestellen oder abzustellen. Es unterstützt Feeder, Changesys und ein paar andere Formate. Öffentliche Beta-Version!

Sonstiges =-=-=-=-=

SUMSTools (2.10) [ss] [kmel] [zza] [tron] [olio] [us] Enthält _vier_ kleine Utilities: sumsset, sumswrite, sumsprint und sumsfolder, zum Lesen und Schreiben von Nachrichten, zum Ändern der Message-Flags und zum Verschieben von Nachrichten in Folder.

ConfUMS (1.2) [olf] Erlaubt die Modifikation der Variablen in der ums.config (Server- und Uservariablen). Vorteil gegenüber der Bearbeitung der ums.config mit einem Texteditor: der UMSServer braucht nicht neu gestartet werden, d.h. Clients könnnen weiterhin eingeloggt bleiben. Benötigt MUI und UMS V11.

Bouncer (1.5) [mh] Der Bouncer sucht in der Messagebase nach Nachrichten, die entweder nicht exportiert werden konnten oder nicht einem bestimmten User zugeordnet werden können.

TopTen (2.0) / UMSMailStat (1.18) [jg] / [olf] Beide Tools erstellen eine Statistik über die Messagebase und zeigen sie nach verschiedenen Kriterien sortiert an.

UMSPasswd (0.11) [sb] [kmel] Tool zum Ändern der Passwörter in der UMS.config. Ein mit UMSPasswd gesetztes/geändertes Passwort wird anschließend verschlüsselt in der UMS.config eingetragen. (Sicherheitshalber sollte man sein Passwort deshalb stets mit UMSPasswd setzen/ ändern und nicht einfach von Hand (und damit unverschlüsselt) in die UMS.config eintragen.)

UMSDoctor (1.3) [mh] UMSDoctor rettet so viele Nachrichten wie möglich aus einer zerstörten Messagebase.

UMSSalve (1.0) [as] UMSSalve basiert auf dem Quellprogramm von UMSDoctor und versucht so viele Nachrichten wie möglich aus einer zerstörten Messagebase zu retten.

UMS Im-/Export (2.4/2.6) [mh] Das sind 2 Programme, um Nachrichten aus einer UMS-Messagebase zu exportieren und sie in eine andere Messagebase wieder zu importieren. Sowas ist z. B. nötig, wenn die Messagebase mal zerstört wird und ein Teil der Nachrichten noch zu retten ist.

FixAccess [mh] Zugriffsrechte für einen (neuen) User updaten. Wenn ein neuer User eingetragen wird, so hat er ja noch keinen Zugriff auf die alten Nachrichten. Mit FixAccess kann man ihm Zugriff auf diese alten Nachrichten geben.

SetFolder (1.3) [kmel] SetFolder schiebt Mails in Folder und benutzt dabei einen schnellen Algorithmus. Evtl. kann ein ReplyName/ReplyAddr gesetzt werden, beinhaltet 'MList2Folder' und benutzt die UMS.config.

Child2Folder (1.0) [olf]

Sortiert neue persönliche Nachrichten in den gleichen Folder ein, in dem die Ursprungsnachricht steht.

SetExpire (1.0) [kmel]

SetExpire gibt Nachrichten, bei denen das "Verfalldatum" überschritten ist, zum Löschen durch CleanUMS (ARexx-Script) bzw. Servercontrol frei.

Servercontrol (1.0) [tron] Servercontrol ersetzt die ARexx-Scripts QuitUMS.rexx, CleanUMS.rexx und FlushUMS.rexx. Benötigt UMS Version 11!
Ab UMS Version 11 wird nur noch ServerControl unterstützt. Ein Befehl mit dem man Kommandos wie Quit, Flush, Cleanup, usw an den UMSServer schickt. Die beiden Befehle QuitUMS und CleanUMS sprechen direkt Funktionen der ums.library an, die es ab Version 11 nicht mehr gibt.

ListWizard (1.4) [ab] Mit ListWizard kann man neue MailingList-Nachrichten in Gruppen verschieben bzw. Kommentare von Gruppen zu den pers. Nachrichten kopieren.

ReplyWizard (1.2) [ab] Mit diesem Tool können automatische Replys auf Mails erzeugt werden. Der Reply-Text ist konfigurierbar und die Mails können nach verschiedenen Kriterien ausgewählt werden.

Away from Mail (0.5) [amb] AFM dient dazu, automatisch eine Empfangungsbestätigung zu verschicken, wenn der Benutzer z. B. im Urlaub ist oder aus einem anderen Grund seine Mail nicht beantworten kann.

UFM (0.7) [chr] UFM ist ein Mailserver. Files können als Mail(s) via UMS angefordert, verpackt und versendet werden.

SortINA (1.3) [us] SortINA ist ein Programm, um das Adressbuch von IntuiNews zu sortieren.

IntuiBook (1.01) [mlu] Ein Adressbuch für den Newsreader IntuiNews mit eingebautem Editor.

UMSWatch (1.2) [ps] Zeigt an, wieviel ungelesene Mail & News in der Messagebase vorhanden sind und macht sich bei neuer Mail bemerkbar.

UMSFault (1.0) [fd] Gibt zu einer UMS-Fehlernummer den entsprechenden Fehlertext aus.

Scripts =-=-=-=

MultiMaus (1.3) [mh] ARexx-Script zum anrufen einer oder mehrerer MausNet-Mailboxen um dort für einen oder mehrere User den MausTausch durchzuführen. Benötigt das Terminalprogramm VLT.

ShowGroups (1.5) [kai] [hG] [olk] ARexx-Script, das eine Liste aller Newsgroups in einer UMS-Messagebase erstellt.

ReplyDaemon (1.1) [hG] [olk] ARexx-Script, das nach neuen Mails in angegebenen Foldern sucht und automatisch eine Empfangsbestätigung (receipt-reply) an den Absender erstellt Die bearbeitete Mail wird als ungelesen, archiviert und zurückgestellt markiert.

ReadMessage (1.4) [kai] [hG] [olk] ARexx-Script, liest eine Nachricht aus einer

UMS-Messagebase und gibt diese nach stdout.

WriteMessage (1.3) [kai] [hG] [olk] ARexx-Script, schreibt eine neue Nachricht in eine UMS-Messagebase.

ShowRexxCfg.intui (1.1) [us] Rexx-Script für IntuiNews, zeigt den Inhalt der Variable IntuiNews.Rexx an.

F: Ich suche einen Laberfilter für UMS. Wo gibt's sowas?

A: Mit SumsSet (aus den SUMSTools) kannst Du einfache Filterfunktionen erreichen, u.a. auch mit Wildcards etc. Eine ältere Version der SumsTools sollte eigentlich Deinem UMS-Paket beiliegen. Ein mächtigerer Filter befindet sich noch in der Betaphase.

1.9 C

C.

Hello World. Ein erstes UMS Programm. Programmierung: ein Überblick. Welche Sprache: Rexx, C, Modula? Interna: umsserver, ums.library. (hier auch Structs MInfo,TxtFlds) Hilfreich: Die umssupport.library. Kleine Tools schnell programmiert.

1.10 C1

Hello World. Ein erstes UMS Programm.

1.11 C2

Programmierung: ein Überblick.

Ich verweise auf/setze voraus die UMS-Developer Dokumentation. Diese Textsammlung findet sich bei den UMS-Includefiles für Entwickler.

UMS:Developer/Doc

Interessante Distributionen/Archive:

UMS-C Release 11.4 (ca. upload @MK2 Juni 1996) Die C-Includes zum Erstellen eigener Programme, die UMS unterstützen.

umssupport.library Version 0.5 (Februar 1996): Eine shared library mit nützlichen Funktionen für UMS-Tools. Enthält z.Z. Funktionen zum Erstellen von Reports und die mächtige Filterfunktion der SUMSTools.

```
Wichtig sind fürs Programmieren

- die UMS Autodocs <ums.doc>

+ Einführung in das UMS-Konzept + Beschreibung der Messagefelder (Messagetext,
FromAddr, ToAddr, Subject ...) + Alphabetische Beschreibung der UMS-Funktionen.

- das UMS Includefile <ums.h>

+ Definition der Strukturen MessageInfo und UMSMsgTextFields.
  (Man kann aber auch in einfachen Tools ohne diese Strukturen leben) +
Definitionen der Tags für die UMS-Library-Funktionen. Die UMS.library
  übersetzt diese Tags in Befehle an den UMS-Server. Die UMS.library
  tut sonst nichts -- im UMS-Server stecken die Routinen.

++++++++++++++
  Tags.
  Seit Amiga OS Version 2 sind Tags ein Standard zur Übergabe von Parametern
  an Funktionen. Tags werden als Tagitems in sog. Taglisten zusammengefaßt.

  Tagitem.
  Ein Tagitem besteht aus zwei Langwörtern (je 4 Bytes): 1. dem Feld
  ti_Code und 2. dem Feld ti_Data.

  Das Feld ti_Code.
  ti_Code enthält ein Argument, bzw. einen Befehl für die Funktion. Der
  Code UMSTAG_RMsgNum bedeutet für die Funktion UMSReadMsgTags: Lese
  eine Message mit Nummer aus ti_Data.

  Das Feld ti_Data.
  ti_Data, das zweite Feld im tagitem, enthält z.B. als Zahl das
  Langwort 502: also wird versucht, Nachricht Nummer 502 einzulesen.

  ti_Data kann nicht nur eine Zahl, sondern z.B. auch einen Zeiger
  enthalten. Beispiel: Ist ti_Code = RGroup, erwartet die UMS-Leseroutine
  eine Adresse, wo sie den Gruppennamen ablegen kann. Diese Adresse
  auf einen Stringbuffer übergibt man in ti_Data. Z.B. wäre vorab
  mit TheFirstCharOfMyStringBuffer = malloc (255) ein ausreichend
  großer Stringbuffer reserviert. Dann kann der Bufferbeginn so
  in ti_Data übergeben werden, wie im folgenden Beispiel.

  taglist       tagitem
                -- ti_Code       UMSTAG_RMsgNum
                -- ti_Data       502
                ----------
                tagitem
                -- ti_Code       UMSTAG_RGroup
                -- ti_Data       TheFirstCharOfMyStringBuffer
                ----------
                tagitem
                -- ti_Code       TAG_DONE
          ................
                -- ti_Data       -------
          ----------------

++++++++++++ Flags
```

+ wichtig: drei Gruppen von UMS-Flags. Die Flags pingelig auseinanderzuhalten
  ist entscheidend. Es ist eine ergiebige Fehlerquelle. Deshalb lohnt
  sich ein genauer Blick. Das spart Zeit und Ärger.

  Der Unterschied Klientprogramm und Server sollte geläufig ein. Er wird im
  <ums.doc> kurz erklärt. Kurz & platt: den Server gibts nur einmal,
  Klientenprogramme dafür beliebig viele.

  Es gibt aus Sicht eines UMS-PROGRAMMES pro Message je 1x diese Flags:

  1. Globale Flags -- gehören dem UMS Messagebase Prozessor
  2. User-Flags   -- gehören einem laut UMS-Konfiguration festgelegten User
  3. Login-Flags   -- gehören individuell zu jedem eigenen Login in das
                      UMS-System. Auch wenn sich ein User zweimal einloggt,
                      hat er zwar nur einen Satz UserFlags, aber pro Login
                      (= zweimal) einen individuellen Satz lokaler Loginflags.

  In den Autodocs heißt es manchmal auch Local Flags. Das sind Login Flags!
  Nichts anderes als zwei Namen für einen Satz Flags. Verwirrend.

  Merke:

                       Local Flags == Login Flags

  Wenn es aus Sicht eines UMS-Programmes, oder eines UMS-Klienten für
  eine Message drei verschiedene Flaggensätze gibt -- dann gibt es aus
  Sicht des Steuerprogramms, des UMS-Servers: für jede Message einen
  Satz globale Flags, zusätzlich für jeden User einen Satz User-Flags
  pro Message -- und darüber hinaus auch für jeden Login
  (= jedes Klientenprogamm) noch einen individuellen Satz an lokalen
  Flags, für jede Message!

  Weiter unten habe ich dazu eine kleine Skizze.

  Hier ein kurzes Beispiel:

  Gegeben sei eine UMS-Messagebase mit 1000 Messages, 10 Usern.
  Alle User sind mit je einem Klientprogramm in den Server eingeloggt.
  Frage: wieviele Flags verwaltet der Server insgesamt?

  Flags gesamt = Anzahl Messages *

                 (1 Globales Flag
                   + Anzahl User
                   + Summe (Anzahl eingeloggte, "lokale" Clients von User i)
)

  Lösung:

  Flags gesamt = 1000   Messages *

```
                ( 1 Globales Flag
                  +    10 User
                  +    10 Logins, weil 1 eingeloggter, lokaler Client pro User
                )

              = 1000   Messages * ( 1 global + 10 user + 10 lokal/login )

              = 1000   Messages *           21  Flags pro Message

  Flags gesamt = 21.000 Flags, die vom Server verwaltet werden.
                 ============================================

  bei 16 Bit pro Flag sind das 42 Kbytes (42.000 bytes) Speicherbelegung.

  [ ist das so richtig? ]
```

1.12 C3

```
Welche Sprache: Rexx, C, Modula?
```

1.13 C4

```
Interna: umsserver, ums.library.

Struktur MessageInfo Struktur UMSMsgTextFields

Flags Dateiverwaltung


Bild des UMS-Systems mit server und library. Programmierung

/* ================================================
 * MessageInfo
 * ================================================
 *
 * Was ist MessageInfo?
 * --------------------
 * Eine wichtige Struktur in <ums.h>:
 *
 * struct MessageInfo {
 *                  LONG      msgi_HeaderLength;
 *                  LONG      msgi_TextLength;
 *                  LONG      msgi_Date;
 *                  UMSMsgNum msgi_ChainUp;
 *                  UMSMsgNum msgi_ChainDn;
 *                  UMSMsgNum msgi_ChainLt;
 *                  UMSMsgNum msgi_ChainRt;
 *                  UMSSet    msgi_GlobalStatus;
 *                  UMSSet    msgi_UserStatus;
 *                  UMSSet    msgi_LoginStatus;
 *                  UMSMsgNum msgi_HardLink;
 *                  UMSMsgNum msgi_SoftLink;
```

```
*  [V11 extension, only filled by UMSTAG_RExtMsgInfo:]
*                  LONG      msgi_CDate;
*                  LONG      msgi_Reserved[3];
*                 };
*
* MessageInfo kann für jede Message mit UMSReadMsg ausgefüllt
* werden. Vorteile: ein Aufruf von UMSReadMsg(), ein Platz für
* die Struktur reicht, um alle wichtigen Daten der Message ab-
* zulegen, auszulesen.
*
* Was liefert MessageInfo nicht?
* ------------------------------
* MessageInfo liefert keine Zeiger auf Texte, Adressen,
* Titel usw.
*
* Dafür gibt es UMSMsgTextFields[UMSNUMFIELDS].
* Siehe <ums.h> für die Definition der einzelnen Feld-Indices.
*
*                       [               ]
*
*               #define UMSCODE_MsgText         0
*               #define UMSCODE_FromName        1
*               #define UMSCODE_FromAddr        2
*               #define UMSCODE_ToName          3
*               #define UMSCODE_ToAddr          4
*               ...
*
*
*       printf ( "FromName: %s",  myTextFields [ UMSCODE_FromName ]   );
*
*       ergibt:
*
*       FromName: Hans Hase
*
* wenn vorab ein
*      UMSMsgTextFields myTextFields
* mit
*      UMSReadMsg() erfolgreich ausgefüllt wurde:
*
* immer testen, ob Lesen erfolgreich war:
*
*      if (!UMSReadMsg(account,tags,...)) { Fehler!} else { OK! }
*
* Mehr über die MessageInfo-Struktur
* ----------------------------------
* Alle Werte in der Struktur sind für genau eine Nachricht
* gültig. Für jede Message müssen die Werte neu eingelesen werden.
*
* Wichtig:
* Die tatsächlichen Werte einer Message können sich jederzeit
* ändern. Angaben in MessageInfo sind nicht ewig gültig. Nur
* im Abruf-Zeitpunkt ist das einigermaßen sicher. Denn:
*
* Parallel laufende externe UMS-Programme (Exporter, Importer,
* Laberfilter, Newsreader) können leicht Daten einer Message abändern.
* Die historischen Angaben in MessageInfo sind dann veraltet.
* Aktuelle Werte der Message erfährt man nicht aus alten MessageInfos.
```

```
* Don't trust MessageInfo!
*
*
* Wie kommen UMS-Daten in MessageInfo?
* ------------------------------------
*
* Mit
*                                                        [ungetestet]
* {
* UMSMsgNum MessageNummer
* struct MessageInfo MInfo,
*
* success = UMSReadMsgTags(account,
*                UMSTAG_RMsgInfo, (ULONG) &MInfo,
*                UMSTAG_RMsgNum,  MessageNummer,
*                TAG_DONE
*                );
*
*
*
* Welche Daten sind in MessageInfo verfügbar?
* -------------------------------------------
* - HeaderLength, TextLength, Date, ChainUp, ChainDn,
*   ChainLt, ChainRt, GlobalStatus, UserStatus, LoginStatus, HardLink,
*   SoftLink.
* - Unter UMS V11 zusätzlich  CDate. Ein spezieller Tag muß beim
*   Einlesen gesetzt werden.
*   (siehe unten)
* - Darüber hinaus gibt es spezielle Daten. Nur mit UMSReadMsg() abrufbar.
*   Siehe <ums.h>.
*
* Wann ist MessageInfo zu umständlich?
* ------------------------------------
* Wer nur wenige Daten aus dieser Liste braucht, ist mit
*
*     LONG Info;
*     success = UMSReadMsgTags(account, UMSTAG_R......., &Info);
*
* besser bedient. Der Overhead einer kompletten MessageInfo-Struktur
* wird vermieden. Es wird nur Platz für die benötigten Felder
* vergeben. Wenn z.B. NUR globale Flags gelesen werden, reicht
* ein UMSSet myGlobalFlags, ein ReadMsg() mit entsprechendem
* Tag und & myGlobalFlags. Dann wird myGlobalFlags mit der passenden
* Info für Message Nummer x gefüllt. Auch hier wieder: Testen
* auf Fehler bei ReadMsg.
*
* Beispiel [ungetestet]
*
* Aufgabe: Gegeben ist die MsgNum 10. Ermittle die
*          übergeordnete Nachricht von Msg 10.
*
*       vorab: ums.lib öffnen, einloggen mit account = UMSRLogin()
*       {
*       UMSMsgNum  MeineMessage = 10;
*       UMSMsgNum  Ursprungsnachricht = NULL;
*
*       if (! UMSReadMsgTags(account,
```

```
 *                       UMSTAG_RMsgNum,  (LONG) MeineMessage,
 *                       UMSTAG_RChainUp, (LONG) &Ursprungsnachricht,
 *                       TAG_DONE)
 *           ) printf("FEHLER bei ReadMsg()")
 *        else
 *       {
 *       if (Ursprungsnachricht)
 *          printf ("Ursprungsnachricht meiner Message %ld ist Message %ld\n",
 *                  MeineMessage, Ursprungsnachricht);
 *       else
 *          printf ("Eine übergeordnete Nachricht zu %ld wurde nicht
gefunden!\n",
 *                  MeineMessage);
 *        }
 *      }
 *      ausloggen,
 *      ums.library schließen
 *
 * Beispiel Ende.
 *
 *
 * Erweiterungen in V 11?
 * ----------------------
 * Unter UMS Version 11 gibts mit UMSTAG_RExtMsgInfo
 * das Feld "msgi_CDate" sowie drei reservierte Langworte als
 * LONG msgi_Reserved[3];
 *
 *       UMSReadMsgTags(account,
 *                       UMSTAG_RMsgNum,      (LONG)   MeineMessage,
 *                       UMSTAG_RExtMsgInfo,           TRUE,
 *                       UMSTAG_RMsgInfo,     (LONG *) myMessageInfo,
 *                       TAG_DONE);
 *
 *
 * MESSAGE INFO - FELDER IM DETAIL
 * ===============================
 * Feldinhalte und mögliche Werte aufgelistet und beschrieben.
 *
 * Wie lang ist der Text?
 * ----------------------
 * LONG        HeaderLength  -- Länge des Message-Headers in bytes
 * LONG        TextLength    -- Länge des Message-Textes in bytes
 *
 * Wie alt ist die Nachricht?
 * --------------------------
 * LONG        Date          -- (obsolet) Datum der Message:
 * LONG        CDate         -- nur mit UMS V11 und Extension-Tag
 *
 * Verknüpfungen zu anderen Nachrichten?
 * -------------------------------------
 * UMSMsgNum   ChainUp       -- Ursprungsnachricht
 * UMSMsgNum   ChainDn       -- gibt es "unter mir" angehängte Nachrichten?
 * UMSMsgNum   ChainLt       -- weitere Nachrichten, die sich auch auf
 * UMSMsgNum   ChainRt       --  die Ursprungsnachricht beziehen
 *
 *
 * Die Stati dieser Nachricht
```

```
* -------------------------
* Hinweis:
* 1. Der Typ UMSSet ist vom Typ LONGBITS. Das sind 32 Bits. Unter UMS
* stehen dem Programmierer _nur die 16 unteren Bits_ zur Verfügung.
* Die 16 oberen Bits (Bit 16 bis 31) sind zur Zeit tabu.
*
* 2. Es gibt drei Gruppen von Statusangaben. Es ist wichtig,
* den richtigen Status zum Schreiben und zum Lesen auszuwählen!
*
* a) GlobalStatus        - nur einen GlobalStatus für jede Message
* b) UserStatus          - pro User und Message: ein UserStatus
* c) Login/LocalStatus   - pro Login, User und Message: ein LoginStatus
*
* Beispiele:
* Messagebase mit 10 Messages, zwei Usern, fünf UMS-Programmen.
*
* MB  Msg#          1  2  3  4  5  6  7  8  9 10
* a) GlobalStatus   g  g  g  g  g  g  g  g  g  g   UMS-Messagebase-Processor
* b) UserStatus1    u  u  u  u  u  u  u  u  u  u   User 1
*    UserStatus2    v  v  v  v  v  v  v  v  v  v   User 2
* c) Login User1    l  l  l  l  l  l  l  l  l  l   IntuiNews User1
*    Login User1    l  l  l  l  l  l  l  l  l  l   Sumsset User1
*    Login User2    l  l  l  l  l  l  l  l  l  l   NewsBreaker User2
*    Login User2    l  l  l  l  l  l  l  l  l  l   IntuiNews User2
*    Login User2    l  l  l  l  l  l  l  l  l  l   TopTen User2
*
* In diesem Beispiel hat eine Message einen GlobalStatus (+1),
* zwei UserStatus (+2 = 3), fünf LoginStatus (+5 = 8).
*
* Zu den Status-Flags
* ===================
*
* UMSSet      GlobalStatus  -- Deleted Expired Exported Orphan Link HardLink
*                              Parked HasFile und (Protected)
* ------------------------------------------------------------------------
* HINWEIS: VIELE FLAGS SIND "PRIVATE". Siehe Protected in ums.h.
*
* Deleted - als Gelöscht markiert.
* Expired - zum Löschen freigegeben beim nächsten Aufräumen der Messagebase
*           mit UMSServerControl()
* Exported - die Nachricht wurde von einem UMS-Exporter erfolgreich versandt.
* Orphan -  Exportieren ist fehlgeschlagen?
* Link - ?  Soft-Verknüpfung für Crosspostings?
* HardLink - Verknüpfung für Crosspostings??
* Parked - der User will die Nachricht erst später versenden. Bis dahin
*          ist die Nachricht geparkt. Exportieren nicht erlaubt.
* HasFile - zu der Nachricht gehört eine Datei. Der Dateiname ist mit
*         UMSCODE_FileName aus einem UMSMsgTextFields[] zu ermitteln,
*         oder alternativ mit Tag UMSTAG_RFileName in einem UMSReadMsgTags()
*         Aufruf.
*
*
*
* UMSSet      UserStatus    -- Archive Junk PostPoned Selected Filtered Old
*                              WriteAccess ReadAccess ViewAccess Owner
*                              und (Protected)
* ------------------------------------------------------------------------
```

```
 *
 * Archive   - die Nachricht soll etwas länger in der Messagebase bleiben
 * Junk      - auf Deutsch Müll. Unter ums: ???
 * PostPoned - ?
 * Selected  - die Nachricht wurde "ausgewählt", z.B. von einem Filtertool.
 *             Newsreader können Selected-Nachrichten gezielt darstellen.(?)
 * Filtered  - die Nachricht wurde "gefiltert", z.B. von einem Filtertool.
 *             Newsreader können Selected-Nachrichten gezielt verbergen.(?)
 * Old       - der User kennt diese Nachricht schon, sie ist "alt".
 *             Ist Old nicht gesetzt, dann ist die Nachricht "neu".
 *
 *             - Verschiedene Zugriffsrechte Deines UMSAccounts. Abhängig
 *              von der UMS-Konfiguration für den User:
 * WriteAccess - ich darf diese Nachricht überSchreiben
 * ReadAccess  - ich darf diese Nachricht und ihre Header lesen
 * ViewAccess  - ich kann diese Nachricht "sehen" (sie ist nicht "versteckt")
 * Owner       - diese Message gehört mir. Ich kann sie ändern, löschen usw.
 * (Protected) - die Flags WriteAccess, ReadAccess, ViewAccess, Owner zusammen-
 *               geORt. When writing global- or user-flags, you  are  not  allowed
to
 *                manipulate all possible flags. Protected Flags sind PRIVATE!
 *
 *
 * UMSSet      LoginStatus   -- 16 Bits (von Bit 0 bis Bit 15) zu Deiner freien
 *                              Verfügung. Werden im ums.doc auch als Local-Flags
 *                              bezeichnet. Bedeutet aber das gleiche wie Login.
 *
 * Diese LoginFlags kommen bei
 * UMSSelect() und UMSSearch() zur Anwendung.
 * Vor Gebrauch: mit UMSSelect zurücksetzen.
 * ------------------------------------------------------------------------
 * Ende StatusFlags. Weiter mit restlichen MessageInfo-Feldern:
 *
 * Für Crosspostings
 * -----------------
 * UMSMsgNum   HardLink
 * UMSMsgNum   SoftLink
 *
 */ struct MessageInfo myMessageInfo;


/* ================================================
 * UMSMsgTextFields
 * ================================================
 * Globaler Zeiger auf Messagefields. Siehe ums.h für die vorgesehenen
 * Feldinhalte. Beispiel: myFields[0] zeigt auf den Message-Body.
 * Wobei die Zahl "0" für UMSCODE_MsgText steht. Zweites Bespiel: Mit
 * myFields[ UMSCODE_FromName ] kommt man an den Absendernamen usw.
 *
 * Nullzeiger:
 * Wenn myFields [ UMSCODE_..... ] = NULL, ist der
 * Eintrag entweder nicht eingelesen, nicht verfügbar oder
 * es gab einen Fehler beim Lesen.
 *
 * 3. Beispiel:
 *
 * printf("Diese Message kommt von:  %s .\n", myFields[ UMSCODE_FromName ] ?
```

```
 *         myFields[ UMSCODE_FromName ] : "-- kein Name vorhanden!\n"  );
 *
 * ergibt:
 *        "Diese Message kommt von:  Stefan Müller"
 *
 * oder:
 *        "Diese Message kommt von:  -- kein Name vorhanden!"
 *
 * Es gibt insgesamt über 40 dieser UMSCODE-message array index definitions
 * (siehe "ums.doc/--message-format--" und das Includefile ums.h.)
 *
 * Warnung:
 * Die mit UMSReadMsg() gelesenen Daten müssen nach Gebrauch wieder
 * freigegeben werden (1 x UMSFreeMsg() ). Die Feldinhalte sind READ-ONLY.
 * Verboten: überschreiben, ändern.
 * (todo)
 */ UMSMsgTextFields myFields;
```

1.14 C5

Hilfreich: Die umssupport.library.

1.15 C6

Kleine Tools schnell programmiert.

1.16 D

D.

Ein- und ausloggen.

Suchen und finden.

Lesen und schreiben.

Voreinstellungen holen und ablegen.

Fehler auswerten und dokumentieren.

Der Weg nach draußen.

1.17 D1

Ein- und ausloggen. ums.library/UMSDupAccount ums.library/UMSLogin (obsolet! nie benutzen!) ums.library/UMSLogout ums.library/UMSRLogin ums.library/UMSServerControl

```
/*
 * Für ein erfolgreiches LOGIN benötigt UMS:
 *
 * ---muß---
 * user --   UMS-Username, z.B. "Stefan Müller", "sysop", ...
 * passwd -- Paßwort, z.B. "sesam", (oder "%", für "kein Paßwort")
 *
 * ---kann---
 * server -- UMS-Server, wird meist nicht angegeben. "default" für
 *           den Standardserver, "<andererServer>" für einen Fremdserver.
 *           Ein in der Regel bei Einplatz-System unbenutztes UMS-Feature.
 *           Forget it.
 *
 * Selbstverständlich muß der User & sein Paßwort bereits für
 * UMS konfiguriert sein. Das sieht in der Regel so aus:
 *

 <ums.config>
 ( Sysop       --- ich bin Sysop-"User"
 ( Alias   --- man kennt mich auch als: root, postmaster, ...
 root
 postmaster
 sysop
 Stefan
 "Stefan Mueller"
 "Stefan_Müller"
 )
 ( READACCESS "#?" ) --- ich kann alle Gruppen lesen
 ( WRITEACCESS "#?" ) --- ich kann in alle Gruppen schreiben
 ( NETACCESS "#?" ) --- ich habe uneingeschränkten Netzzugang
 ( IMPORT % ) --- ich kann nichts importieren (das macht der Importer)
 ( Name "Stefan Müller" ) --- mein persönlicher Name (realname)
 ( Password  ) --- mein Paßwort. Da ich keins habe: kein Eintrag.
 )

 *
 * Fehlt der Usereintrag, oder gibt es Fehler beim Login, bricht
 * NewsBreaker sofort ab. (todo: Abbruch erst bei Cancel)
 *
 */

STRPTR user = NULL;   /* UMS-Username */ STRPTR passwd = NULL;    /* geheimes  ↩
   Paßwort
*/ STRPTR server = NULL;    /* (optionaler) UMS-Server */

/* Account, globaler Handle für alle UMS-Funktionen */ /* wird nach erfolgreichem
Login gesetzt, und nach Logout
 * werden alle UMS-Puffer für den account freigegeben.
 */ UMSUserAccount account = NULL;  /* globaler Handle */
```

1.18 D2

Suchen und finden. ums.library/UMSSearch ums.library/UMSSelect

aus dem ums.doc:

ums.library/UMSSearch ums.library/UMSSearch

NAME
UMSSearch -- Search a message from the MB.

SYNOPSIS
msgNum = UMSSearch(login, tags)
D0 D2 D3

LONG UMSSearch(UMSAccount, struct TagItem *);

msgNum = UMSSearchTags(login, tag1, ...)

LONG UMSSearchTags(UMSAccount, ULONG, ...);

FUNCTION
Search the first (or next) message in the MB that fulfils certain criteria.

When you want to read certain messages from the MB, it is recommended that you first select these messages with UMSSelect() and then alternately use UMSSearch() and UMSRead() to get all these messages.

INPUTS
Allowed tags:

SearchLast (LONG)

specify the last message NOT to search. This tag allows you to cycle trough all messages fullfilling the same criteria: set this to zero and invoke UMSSearch() the first time. Check the result and if it's not zero, put it in this tag and invoke UMSSearch() again. Repeat this until it returns zero.

SearchDirection (LONG)

set the search direction. 1 means search forward (to higher numbers), -1 means search backwards (to lower numbers) and 0 lets the MBP decide what sequence to use. This needn't be exactly forwards or backwards. It might be in a completely different order.

When you don't depend on a certain search-direction, use 0 or omit this tag.

SearchGlobal (none)
SearchLocal (none)
SearchUser (STRPTR)
SearchMask (LONGBITS)
SearchMatch (LONGBITS)

search for a matching status; like SelReadGlobal, SelReadLocal, SelReadUser, SelMask and SelMatch with UMSSelect().

WMsgText, ..
WMsgText + 127 (STRPTR)

search for a matching text; as for UMSSelect(). Only one field can be searched for at a time.

SearchQuick (none)

Enable 'quick-search'. This must be combined with exactly one of WMsgText+1 .. WMsgText+31. quick-searches are possible for exact string searches only, they must not be combined with patterns and they are only possible for fields that have an index.

They don't guarantee that the returned message's field actually matches the given string, altough mistakes are very unlikely.

But they are fast! (see NOTES below)

SearchPattern (LONG)

indicate whether the string to be searched for is an exact string (0), an AmigaDOS pattern (1) or UMS should try to find out (2).

RESULT
msgNum - numer of a/the searched message; zero if not found.

NOTES
Although LONGBITS are used in the definition, the current implementation only uses/supports the lower 16 bits.

Performance: when searching for strings, different calls to UMSSearch() may significantly vary in performance. There are three general possibilities:

1) quick-searches:

very fast, no access to the hard-disk needed (once the right index is loaded into memory). Only possible if tag 'SearchQuick' specified.

2) indexed searches:

fast, in most cases only one, short access to hd is needed; a few more in really bad situations. If the 'header'-file is sufficiently buffered, no accesses to the hd may occur. Possible if searching for exact strings in indexed fields.

3) other searches (non-indexed or patterns):

slow, many data will have to be read from hd. If the field searched for is in the 'header'-file and it's heavily buffered, no accesses to the hd may occur. Nevertheless the search will consume much CPU-time.

```
        Search for status!!!

        Searching for a matching status only (i.e. not searching  for  a
      string) is always very fast.

        When doing non-indexed- or pattern-search, combine  with  status
      to reduce the amount of data to be searched through!

   SEE ALSO
       UMSSelect(), UMSReadMsg(), <ums.h>

?ums.library/UMSSelect                                  ums.library/UMSSelect

   NAME
       UMSSelect -- Select messages.

   SYNOPSIS
       count = UMSSelect( login, tags )
        D0                D2     D3

       LONG UMSSelect( UMSAccount, struct TagItem * );

       count = UMSSelectTags( login, tag1, ... )

       LONG UMSSelectTags( UMSAccount, ULONG, ... );

   FUNCTION
        Select  messages  in  the  MB according to various criteria.  To
      'select'  here means to set or unset some flags, which then can be
      used by UMSSearch(), stored, or transferred to another user.

        UMSSelect()  can  only  do  one operation upon every invocation.
      An  operation  usually  looks  for  all messages that fullfill the
      specified criteria and then selects them in a specified way.

        When  you  want  to  select  messages  by  different,  logically
      combined criteria, you may need to call UMSSelect() more than once
      and  use  some  temporary  flags.  However, very few calls to this
      functions usually should suffice.

   INPUTS
        The  following  tags  control  the selection of messages.  Thus,
      they somehow specify the 'output' of the select operation.

       SelWriteGlobal (none)

           manipulate global flags on the selected messages.

       SelWriteLocal (none)

           manipulate your local login-flags.

       SelWriteUser (STRPTR)

           manipulate another user's user-flags.  You  must  specify  the
         users name (or alias).
```

SelWriteGlobal, SelWriteLocal and SelWriteUser are mutually exclusive -- you can manipulate only one flag-table at a time. When specifying none of these tags, your user-flags will be manipulated as default.

SelSet,
SelUnset (LONGBITS)

on each selected message the 'SelUnset' flags are cleared and then the 'SelSet' flags are set. ['status = (status & ~unset) | set;']

When writing global- or user-flags, you are not allowed to manipulate all possible flags. See <ums.h> for protected flags.

The following tags control what and how messages are selected, the 'input' and 'modes' of the select operation.

SelStart,
SelStop (LONG)

Limit the number of messages to be processed. The select operation will start with the message indicated by 'SelStart' and stop before the 'SelStop' message. In other words, start is included and stop is excluded.

(0 < start <= messages to be processed < stop)

This was different and partly buggy in MBP versions prior to V10.16.

If no 'SelStart' is specified, the operation starts with the first message; if no 'SelStop' is specified, the operation stops at the last existing message.

The following operations are mutually exclusive:

1) select by status

SelReadGlobal (none)
SelReadLocal (none)
SelReadUser (STRPTR)

like SelWriteGlobal, SelWriteLocal, SelWriteUser, but specifys which flags to look at. Again, your user-flags are the default.

SelMask,
SelMatch (LONGBITS)

specify a mask and a match. If (status & mask) == match [status * mask = match], the message will be selected.

SelParent (none)

with this tag specified, each message's 'parent' (reference; -> reply-chaining) will be inspected instead of its own status.

SelMaxCount (LONG)

this tag specifies a maximum number of messages to select. Even if there are more message with matching status, only that much of them will be selected. Selects are done backwards, i.e. you'll get the last N matching messages.

SelMaxSize (LONG)

this tag specifies a maximum size for all selected messages. When selecting (from the end of the MB towards the beginning), the selected messages sizes are summed up. If the sum exceeds the specified number bytes, the operation stops.

2) select by date

SelDate (LONG)

the messages' dates are compared with the supplied date (in seconds since 1.1.1978) and only the younger ones will be selected.

3) select by creation date

SelCDate (LONG)

the messages' creation dates are compared with the supplied date (in seconds since 1.1.1978) and only the younger ones will be selected.

NOTE: binary creation date is an optional field. Only those messages that have this field can be selected whit this function!

4) select a tree

SelTree (LONG)

you must specify the number of a message here. Then all messages being in the same reply-tree will be selected.

5) select a sub-tree

SelSubTree (LONG)

like SelTree, but only the subtree (the one with the specified message as its root) is selected.

6) select a single message

SelMsg (LONG)

select only the specified message.

7) select by text

```
      WMsgText, ..
      WMsgText + 127 (STRPTR)


          when you specify one of these tags, the function  selects  all
        messages which have the supplied string in the specified  field.
        The strings are compared case-INsensitive.

      SelQuick (none)

          when this tag is  specified,  'quick-search'  is  enabled  for
        selecting texts. This means that only some CRCs on the texts are
        compared.  This  makes  it  possible  to  select also some wrong
        messages. Yet, due to the usage of 32-bit CRCs, the  probability
        of  selecting  wrong  messages is VERY low, you most likely will
        never experience this case.

          As 'quick-search'  does  not  usually  need  to  access  mass-
        storage, it is VERY FAST.

    8) select by size

      SelSize (LONG)

          the messages' sizes are  compared  with  the  supplied  number
        (bytes) and only biggers ones will be selected.

  RESULT
      count - how many messages have been selected.  Zero, when no
              message has been selected or an error has occured.

  EXAMPLE
        See SelectMail.c for examples on how to use this function.

  NOTES
        Although  LONGBITS  are  used  in  the  definition,  the current
      implementation only uses/supports the lower 16 bits.

  SEE ALSO
      UMSSearch(), <ums.h>
```

1.19 D3

Lesen und schreiben. ums.library/UMSDeleteMsg ums.library/UMSExportedMsg ums.library/UMSFreeMsg ums.library/UMSReadMsg ums.library/UMSWriteMsg

```
++++++++++++++++++++++++++++++++++++++++++++++++ /* Messagetypen für WriteMsg():
*//* todo --> globals.h */ enum
  {
    EMAIL, POSTING, REPLY, FOLLOWUP
  };

/* dazu passende Strings... */ STRPTR msgtypename[] = { "EMail", "Posting",
"Reply", "FollowUp" }; STRPTR msgtypeaction[] =   {   "Private EMail schreiben",
```

```
  "Öffentliches Posting schreiben",   "Reply auf Nachricht schreiben",  "FollowUp
auf Nachricht schreiben"  };
```

1.20 D4

```
Voreinstellungen holen und ablegen.

   UMS stellt allen Programmen eine zentrale Konfigurationsdatei zur
   Verfügung. Über Routinen der ums.library können dort gespeicherte
   Parameter -- ohne DOS- oder ENV-Abfragen -- gelesen werden. Daneben ist
   umfangreicher Vergleich von Namensmustern (Patternmatching) möglich.


ums.library/UMSFreeConfig ums.library/UMSMatchConfig ums.library/UMSReadConfig
ums.library/UMSWriteConfig

   - UMSReadConfig()
   - UMSWriteConfig()
   - UMSMatchConfig()

   einige UMSTAGs : (UMSTAG_CfgName ... )

CfgGlobalOnly CfgName CfgUser CfgQuoted CfgUserName CfgNextVar CfgNextAlias
CfgNextUser CfgNextExporter CfgNextNetGroup CfgNextNetGroupMember CfgLockVar
(Write-Tags weggelassen, siehe ums.h) MatchGlobalOnly MatchVarname MatchUser
MatchString MatchDefault

/* id of the application: NB_NAME "." <Config-Element-Bezeichner> eg. Config
Element: NewsBreaker.Window.Size
 == newsbreaker.window.size == NeWSBrEakER.WiNdoW.Size */

++++++++++++++++++ Beispiel

/*#define NB_UMSCONFIG  NB_NAME "."*/ #define NB_UMSCONFIG  "IntuiNews" "."


BOOL ReadConfig (void) {
  realname = UMSReadConfigTags (account,        UMSTAG_CfgUserName, user,
        TAG_DONE);

  printf ("Configlesetest: " NB_UMSCONFIG "Editor" " --%s--\n",
                                UMSReadConfigTags (account,         UMSTAG_CfgName ↩
                                   ,
NB_UMSCONFIG "Editor",        TAG_DONE) );

  return (TRUE);    /* dummy, todo */ }

++++++++++++++ ein paar <ums.config> Einträge

 Sysname
 Options
 Aka
 Errorfile
 Logfile
 Loglevel
```

```
Headerfields
Indexfields
HardFlush
SoftFlush
AutoFree
AutoQuit
Hdrfill
Txtfill
Maxmsgsize
Access
FullAccess
NodeMode
expire.grouplist
expire.mail
expire.private
IntuiNews.Book
IntuiNews.Editor
IntuiNews.FastScan
IntuiNews.ForceQuote
IntuiNews.ForwardHeader
IntuiNews.FupTo
IntuiNews.Headers
IntuiNews.Organization
IntuiNews.QuoteChars
IntuiNews.QuoteString
IntuiNews.QuoteWidth
IntuiNews.ReplyTo
IntuiNews.SaveDrawer
IntuiNews.SaveMark
IntuiNews.SaveNameType
IntuiNews.ScanJunked
IntuiNews.ScanMode1
IntuiNews.ScanMode2
IntuiNews.Signature
Intuinews.StartupNew
IntuiNews.Sort
IntuiNews.Sort.maus
IntuiNews.Sort.maus.mk2.comp.sys.amiga
IntuiNews.Sort.maus.mk2.de
IntuiNews.Sort.maus.mk2.de.comp.sys.amiga
IntuiNews.Rexx
UMSGroup.DefaultMaus
UMSGroup.InfoWindowOn
folder.folder
Exporter
    Alias

    READACCESS "maus.mk2.#?"
    WRITEACCESS "#?"
    NETACCESS "%|#?"
    IMPORT "#?"
    EXPORT "#?"
    DISTRIBUTION "#?"
    Name maus.mk2
    Password
    maus.sysopinfo N
```

```
Sysop
    Alias
       root
       postmaster
       sysop
       Stefan
       "Stefan Mueller"
       "Stefan_Müller"

    READACCESS "#?"
    WRITEACCESS "#?"
    NETACCESS "#?"
    IMPORT %
    Name "Stefan Müller"
    Password
    maus.mk2.info "ITG-2, ITI-17, IIL-19, IGT-15"
    uucp.username sm
```

1.21 D5

Fehler auswerten und dokumentieren. ums.library/UMSErrNum ums.library/UMSErrTxt ums.library/UMSErrTxtFromNum ums.library/UMSLog

1.22 D6

Export, Import: Der Weg nach draußen.

ums.library/UMSCannotExport ums.library/UMSExportedMsg

```
ums.library/UMSCannotExport                          ums.library/UMSCannotExport

   NAME
       UMSCannotExport -- Mark a message as not being exportable.

   SYNOPSIS
       UMSCannotExport( login, msgNum, error )
                         D2      D3     D4

       VOID UMSCannotExport( UMSAccount, LONG, STRPTR );

   FUNCTION
         Tell the MBP that you cannot export this  message,  due  to  any
       reason. The MBP will then look if other exporters still  could  be
       able to export the message or do some error processing otherwise.

         It may use the supplied error-string therein. This  error-string
       should be a  short  description  why  the  message  could  not  be
       exported. By convention it should not be longer than 80 bytes.

   INPUTS
       login  - Handle as returned by UMSLogin() or UMSRLogin()
```

```
       msgNum - Number of the message.
       error  - Short string (<80 chars).

   NOTES
         This function may only be called by exporters.

         The MBP currently does not write a bounce mail to the author  of
       the mail, but sets a  special  global  bit  instead,  which  would
       allows the 'bouncer' tool to automatically write bounce mails.

   SEE ALSO
       UMSExportedMsg()
```

1.23 E

```
E. Do it yourself.

Benutzeroberfläche.

Listenverwaltung.
```

1.24 E1

```
Benutzeroberfläche.

   LISTVIEWS:
   Variabeln für das BOOPSI ListView von Tim Stack. Visit his Website!
   Er hat ca. 20 BOOPSI Objekte online, als Exe und komplett mit Source.

   WWW: (Stand November 1996)
   ----
   http://www.cs.utah.edu/~stack/boopsi.html

   Tims BOOPSI-Hauptseite mit Links auf:

   -  http://www.cs.utah.edu/~stack/classsrc.lha
   Sehr großes (> 1.000 K!) Archiv mit allen Classes und Sourcen
   Aktuelle Dateien sind nicht im Archiv, sondern in dem Classes/ dir.

   -  http://www.cs.utah.edu/~stack/makealias.iff
   ein Hilfsprogramm.

   -  http://www.cs.utah.edu/~stack/classes
   Oberdirectory von Tims Sourcen
   (wie in <classsrc.lha>, hier können gezielt Einzelfiles geladen werden)
   - Classes/itemlist/demo.c  ---- idcmp loop demo

   Frage: wie kommt man an das Verzeichnis einer Web-URL???

   AMINET: (Stand November 1996)
   -------
   ListViewClass.lha  dev/gui     60K   1+BOOPSI listview gadget with source
```

```
   -- das ist eine ältere Version (August 1996) von Tims ListView Gadget und
      ListViewItems

   EMAIL: (Stand November 1996)
   ------
   Tim Stack <stack@eng.utah.edu>

++++++++++++++++++++++++++++++++++
```

Buttons können auf verschiedene Weise in einem Fenster dargestellt werden. Hier eine kurze Tabelle:

Teil vom Amiga OS: - intuition.library der traditionelle Weg, unhandlich - gadtools.library neu seit Amiga OS 2, etwas einfacher - BOOPSI Gadget Subsystem flexibel, aber nicht gerade leicht

Außerdem verfügbar: - Custom BOOPSI Classes vorgefertigte BOOPSI Gadgets, einfach - MUI Oberflächensystem flexibel, durchdacht, ohne Feinschliff - zahllose Gadget.libraries nicht standardisiert, ohne Feinschliff

Das button.gadget ist eine öffentliche Custom BOOPSI-Klasse. Quelle: Amiga Developer CD 1.1 Mai 1996.

In NewsBreaker verwende ich z.Zt weitere BOOPSI Klassen von Dritten: textfield.gadget von Mark Thomas, listview.class (+ subclasses) von Tim Stack. BOOPSI-Klassen sind imho das beste, denn sie können flexibel eingesetzt werden.

In NewsBreaker wird zur Zeit die Knopfleiste im Hauptfenster aus button.gadgets aufgebaut. Das kann auch ein anderes Gadget werden, evtl. schaue ich mir Tims Entwicklung an, wenn Zeit ist.

Implementation/ Bindung der Gadgets an NewsBreaker:

- In creategads() werden alle Buttons mit NewObject() erzeugt.

  Die Tagliste für NewObject, pro Gadget:
  Das UserData-Feld wird ausgefüllt mit einem Actioncode.
  (Siehe <actioncode.c> zum Actioncode-System in NewsBreaker.)

  Beispiel:
  Knopf "Nächste" -- blättert zur nächsten Nachricht.
  Für Gadget gads[GID_Naechste] wird in NewObject() das Feld
  GA_UserData auf Actioncode (DISPLAY_NEXT_MESSAGE) gesetzt.

Buttons erzeugen. In dieser Routine werden nur die Buttons erzeugt und ins Fenster eingebunden. Auswertung der Gadgets passiert erst in <mainloop.c>. Nachdem die Gadgets erzeugt wurden, ist die Aufgabe der Routine hier, in <AddButtons.c>, erfüllt.

Buttons auswerten.

- in der IDCMP-Schleife in <mainloop.c> wird bei einem
  GADGETUP-Ereignis
  a) die Gadgetadresse geholt (steht in msg.IAdress)
  b) UserData "dreckig" ausgelesen. (msg.IAdress->UserData)

  Dreckig, weil BOOPSI-Attribute mit BOOPSI-Methoden zu lesen sind.

Also GetGadgetAttr(...UserData,&actioncode_storage...) wäre 100% systemkonform.

c) Jetzt kennen wir den Actioncode der gewünschten Routine.
Nach unserem Beispiel oben wäre das der Actioncode
DISPLAY_NEXT_MESSAGE. Fehlt nur noch, die Routine tatsächlich
auszuführen. Und das passiert eben jetzt.

d) Ende der IDCMP-Schleife. Zurück zum Anfang, dort warten auf neue Signale,
IDCMP-Events.

- nach Abbruch aus der Hauptschleife MainLoop(), z.B. durch Quit-
Menüpunkt, müssen die Gadgets wieder gelöscht werden. Wichtig,
weil ja Speicherplatz der Gadgets dem System zurückgegeben werden
muß.

Diese Aufräumarbeiten werden von deletegads() in diesem File
<AddButtons.c> ausgeführt.

deletegads() wird von main() in <Newsbreaker.c> aufgerufen.

++++++++++++++++++++++++++++++++

BOOPSI
Sehr flexibles Tool für die GUI-Programmierung. BOOPSI ist
objektorientiert. Es existieren in BOOPSI verschiedene
und doch ähnliche Funktions-Klassen. Es gibt Klassen für
Gadgets, für Bilder (Image), für Mauszeiger (Pointer) und
eine Klasse, um alle Objekte miteinander zu verbinden. Der Vorteil:
vorhandene Klassen lassen sich leicht durch Unter-Klassen erweitern.
Nachteil: BOOPSI-Oberflächen können das System ausbremsen, weil
BOOPSI im input-Task läuft. Besonders auffällig bei langen Texten
im Textfield: der Mauszeiger stockt.

BOOPSI programmieren:
Die Programmierung ist einheitlich und leicht durchschaubar.
Mit NewObject() wird ein gewünschtes BOOPSI-Objekt erzeugt.
Das Aussehen wird über eine mehr oder weniger lange Tagliste
definiert. Ein BOOPSI-Gadget wird mit den gleichen Routinen
wie ein Standard-Intuition-Gadget zum Fenster hinzugefügt.
Unterschiede gibts bei der Änderung 'zwischendrin' sowie
Abfrage von Gadget-Werten. Das läuft über spezielle BOOPSI/
Intuition-Routinen. Und nicht über (mögliche, aber ungute!)
Zugriffe auf die (interne) Gadget-Struktur des BOOPSI-Objekts.

Es ist viel Arbeit, ein vernünftiges Interface zu bauen.
Deshalb greife ich auf fertige Klassen von anderen Programmieren
zurück.

Tim Stack <stack@eng.utah.edu> hat ein paar interessante BOOPSI-Klassen
geschrieben, die ich mehr und mehr verwenden werde. Seine Website ist
interessant für alle, die in diese Materie tiefer eindringen möchten.
(Adresse einfügen). Tim hat seine kompletten Sourcen ins Netz gestellt.
Auf dem Aminet liegt eine ältere Version seiner ListView-Klassen,
mit Source, einem kleinen Beispiel, und etwas Dokumentation.

Mark Thomas ist Autor des bekannten Textfield-Gadget. Diese BOOPSI-

Klasse ist speziell für mehrzeilige Textdarstellung und -eingabe ausgelegt. Darüberhinaus sind viele Einstellmöglichkeiten vorhanden, die Ausehen und Reaktion des Textfield flexibel konfigurierbar machen.

NewsBreaker ist aus dem Demo-Source der Textfield-Distribution gewachsen.

1.25 E2

Listenverwaltung.

Lists.c ----- Funktionen für die Exec-List-Verwaltung in Newsbreaker.

Während in früheren Versionen hier Quelle zahlreicher Fehler lag, sind die Routinen in Lists.c mittlerweile ausreichend stabil.

List-Funktionen verwende ich für die Verwaltung der Gruppen- und Messagelisten. Vorteile: Systemstandard. Nachteile: höherer Speicher- verbrauch, Speicherfragmentierung.

WAS SIND EXECLISTS?

Execlisten werden von der Amiga-Exec-Library unterstützt und durchgängig im OS verwendet. Execlisten bestehen aus einem Listen-Kopf (List, Listheader) und null bis unendlich vielen Listen-Knoten (Node).

Execlisten werden über den Listen-Kopf angesprochen. Ist die Speicheradresse vom Kopf bekannt, können die Standard-Exec-Funktionen auf die an den Kopf angehängten Listenelemente verwendet werden.

Die Nodes der Execliste, also die Elemente sind miteinander doppelt verkettet. Ein Element enthält erstens die Adresse seines Vorgänger- Nodes, zweitens die Adresse seines Nachfolger-Nodes. Durch spezielle Vergleiche dieser zwei Adressen kann -- von einem beliebigen Node aus -- das Listenende, der Listenanfang, der Listenkopf errechnet werden.

WIE WERDEN EXEC LISTS VOM BETRIEBSSYSTEM UNTERSTÜTZT?

Standard-Exec-Funktionen erlauben das Einbinden von neuen Nodes in eine Liste. Ebenso das Austragen (Entfernen) nicht mehr benötigter Nodes aus der Liste. Weitere, höhere Funktionalität gibt es durch sortiertes Einfügen eines Nodes, durch Suche nach einem Namen in den Nodes.

Durch diese sehr grundlegenden Listen-Bearbeitungsroutinen werden Fehlerquellen vermieden.

DIE VORTEILE DYNAMISCHER LISTEN

Software wird benutzerfreundlicher, wenn Datenstrukturen über Listen verwaltet wird. Eine Liste läßt sich beliebig erweitern. Ein festcodiertes Array (Feld) mit einer genau festgelegeten Elementeanzahl ist nicht so flexibel. Im Ergebnis schränken festgelegte Strukturen die Benutzerfreundlichkeit der Software ein. Ein

Texteditor, der bei 1.000 Zeilen Schluß macht, eine Datenbank-Software, die nur bis zu 100 Felder erfassen kann -- weil ihre Datenstrukturen willkürlich vom Programmierer auf eine Obergrenze festgelegt sind -- schränken den Benutzer ein.

WOZU DER AUFWAND?

Es ist mehr Programmieraufwand, die Daten dynamisch über eine Liste zu speichern als in einem Feld. Mehr Checks, mehr Code, mehr Fummelei mit Pointern, höhere Fehleranfälligkeit. Aber es lohnt sich, ist ein Qualitätsmerkmal.

```
/*
 * MyNodes -- (vorläufiger) Listnode für die Message- und Gruppenlisten
 *
 *  struct Node node  -- Standard-Exec-ListNode mit Name, Pri und Type
 *  UMSMsgNum num     -- Speicher für die Messagenummer (für GlobMsgList)
 *  STRPTR text       -- Speicher für Eintrag in Listview (MyNodes.node.ln_Name
=UMS-Intern) (todo)
 */ struct MyNodes
  {
    struct Node node;   // Eine Node-Struktur

    UMSMsgNum num;
    STRPTR text;    /* interner Text */
  };

/* Typ name_nodes... */ #define NAME_NODE   200


/* Messagedaten in Newsbreaker.
 * ============================
 *
 * Leider wird von ums.library etc. kein allgemeines Format angeboten,
 * in dem Applikationen Nachrichten- und Gruppenlisten speichern
 * und verwalten können.
 *
 * Deshalb hier die (vorläufige) Implementierung der Message- und
 * Gruppenverwaltung in NewsBreaker:
 *
 * Zur Zeit werden hier UMS-Messages in zwei unterschiedlichen Exec-Lists
 * gespeichert:
 *
 * 1. in einer globalen GRUPPEN-Liste
 * 2. in einer globalen MESSAGE- oder Nachrichten-Liste
 *
 * Dazu:
 *
 * 1. (List *) GlobGroupList
 *
 *   verwaltet eine Liste der zugänglichen Gruppen. Die Globale
 *   Gruppen Liste wird mit MakeGroupList() erzeugt. [ Zur Zeit
 *   wird hier noch über die globale Variable "GlobalFocus" gefiltert,
 *   ob alle vorhandenen Gruppen oder nur Gruppen mit neuen
 *   Nachrichten in die Liste aufgenommen werden (kludge) ].
 *
 *   Ein Node der GlobGroupList enthält (zur Zeit) im ln_Name-Feld einen
```

```
 *   STRPTR auf den Gruppennamen. Diese Info wird in anderen
 *   Routinen (z.B. Gadtools ListView-Handling) ausgelesen.
 *
 *   [ Hinweis: Es ist schon in der Nodestruktur (siehe Globals.h)
 *   ein weiteres STRPTR-Feld vorbereitet um zwischen internem UMS-Gruppenname
 *   und ListView-Darstellung der Gruppe zu unterscheiden. -todo.]
 *
 *
 *   2. (List *) GlobMsgList
 *
 *   verwaltet eine Liste der wählbaren Messages. Die Globale Message
 *   Liste wird mit MakeMsgList() erzeugt. Hier wird der "Focus", also
 *   die Beschränkung auf bestimmte Nachrichten (alle neuen, alle, ...)
 *   schon im Aufruf von MakeMsgList () übergeben.
 *
 *   Ein Node der GlobMsgList enthält neben einem STRPTR für die ListView-
 *   Darstellung die (eindeutige) UMS-Message-Nummer. Diese UMS-Message-Nummer
 *   wird aus dem Node ausgelesen:
 *   wenn User in einem ListView den entsprechenden Eintrag anklickt.
 *
 *   Allgemeines zu den globalen Listen:
 *
 *   - Dieses Konzept wird weiter optimiert. Siehe auch die Kommentare
 *     in den Funktionen.
 *   - GlobGroupList und GlobMsgList werden über die Funktionen
 *     in <Lists.c> verwaltet. Hier sollten noch speziellere Fallunter-
 *     scheidungen getroffen werden (todo)
 *   - Hinweis: Zu beiden globalen Listen gehören weitere globale Variablen.
 *     Diese werden unten behandelt.
 */ struct List *GlobGroupList = NULL;  /* Pointer auf globalen Listheader */
struct List *GlobMsgList = NULL;  /* Pointer auf globalen Listheader */

/* GlobMsgGroup : wenn nicht NULL, dann ist STRPTR der Name einer Gruppe.
   Diese Gruppe ist mit GlobMsgList verbunden. AktMsgGroup kann gleich
   GlobMsgGroup sein, muß aber nicht. */

STRPTR GlobMsgGroup = NULL; /* Message-Liste hat Nodes aus Gruppe "GlobMsgGroup"
*/ STRPTR AktMsgGroup = NULL; /* User wählt Gruppe AktMsgGroup */

UMSMsgNum GlobMsgNum = 0; /* Globale MsgNum in NewsBreaker */ UMSMsgNum tempNum =
0;    /* temporäre Messagenummer für einige Routinen (kludge) */

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