Emperor

COLLABORATORS			
	TITLE : Emperor		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 23, 2025	

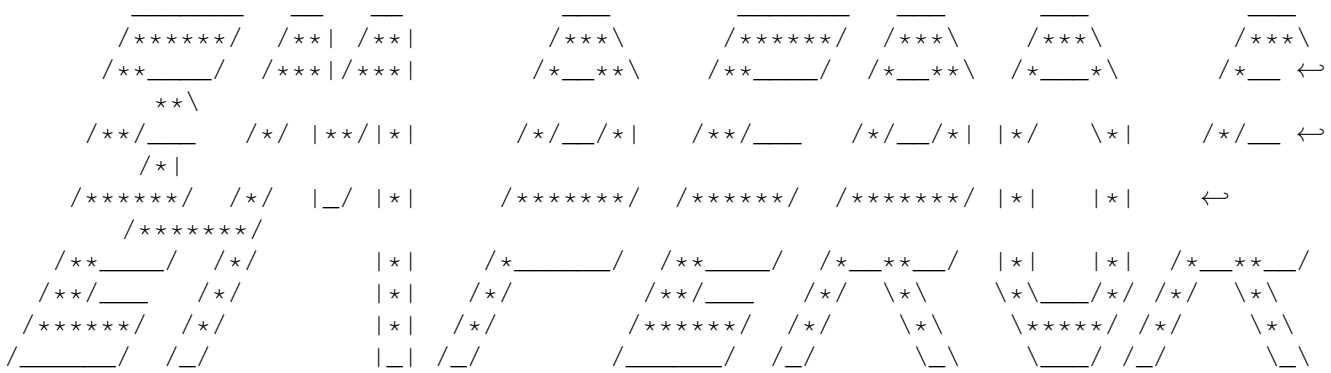
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Emperor

1.1 Die Dokumentation zu Emperor



The object-oriented programming language

Available at www.aminet.de ! (Drawer: dev/c)

Emperor - eine Objekt-orientierte Programmiersprache

1.2 Die Einführung

Bitte unterstützen Sie die Weiterentwicklung dieser Software und mailen Sie mir Ihre Meinung, Probleme und Kritik, denn die Programmierung hat mich viiieeelll Zeit gekostet (aber ich hab’s gern getan !).

Vielen Dank, daß Sie dieses Programm entpackt haben und ausprobieren möchten ! Dieses Programm soll die Software-Entwicklung auf dem Amiga "revolutionieren".

Tjaaa, nun ist’s vollbracht, das Meisterwerk !

Objekt-orientierte Programmiersprache (OOP) ist vielleicht ein wenig zu hoch gegriffen, da weiterhin "echte" C/C++-Quelltexte generiert und durch einen externen C/C++-Compiler übersetzt werden müssen. Im Amiga-Bereich gibt’s nur wenige Ausnahmen wie CanDo, Storm-Wizard u.a., aber irgendwann habe ich das Programm "Delphi" auf dem PC :(kennengelernt ! Diese OOP ist ausschließlich in Pascal gehalten (was mich arg störte) und erschlägt einen mit den Millionen Konfigurations-möglichkeiten. Ich kannte soetwas in der Art bisher auf dem Amiga nicht, und kurzentschlossen setzte ich mich selbst daran.

Ich wollte die Parametrierung der Oberfläche und der Makro-Elemente so einfach wie möglich machen. Folgende Standard-Oberflächenelemente sind implementiert:

* Fenster
* OS2.0 Gadgets (GadTools)
* OS3.5 Gadgets (Reaction)
* Requester (EasyRequest)
* Menüs (NewMenu)
* ASL-Requester (File, Font, Screenmode)
* Locale (Übersetzungen)
* vorgefertigte Funktionsmakros, die das Programmieren unter OS3.5 wesentlich erleichtern.

Man generiert mit Emperor einen komplett fertigen C/C++-Quelltext, der nur noch durch einen externen Compiler (z.B. StormC auf der Developer CD2.1) übersetzt werden muß. Dabei gibt es verschiedene Dateien, in denen folgendes gespeichert wird:

PROJEKTNAME.c	Programmquelltext ansich (mit Verwaltung etc.)
PROJEKTNAME.cd	Katalog-Roh-Daten (meist in Englisch)
PROJEKTNAME.h	Eigener geschriebener Code
PROJEKTNAME.project	Projektdatei für Emperor ansich
PROJEKTNAME.¶	Storm-Projektdatei
PROJEKTNAME_SPRACHE.ct	übersetzter Katalog in Form von Rohdaten

Ziel dieses Programms soll sein, Softwareentwickler zu überzeugen mit Emperor ihre Programme (deren Quelltext) zu erstellen, da die Verfahrensweise bei der Erstellung von Software (Öffnen von Fenstern, Datei- und Schriftauswahlrequestern, Menüs etc.) oft gleich ist und nicht weiter durch den Programmierer selbst mühsam aus anderen Quelltexten herausgefischt werden muß. Emperor verfügt für jedes Element schon ein oder mehrere vorgefertigte, aber veränderbare, Makros. So z.B. bei den Menüs: hier muß für das Projekt-Menü (Öffnen, Speichern, Drucken etc.) und dessen Übersetzung (für Locale - ".catalog"-Datei) nur der entsprechende Eintrag gewählt werden. Dies gilt auch für Requester z.B. für das QuitRequest. Diese müssen nur noch im eigenen Quelltext als Funktion "REQUESTERNAME();" aufgerufen werden.

Das Programm soll aber auch für Amiganer sein, die noch nie programmiert haben, und so einen einfachen Einstieg in die Programmierung der AmigaOS-Oberfläche bieten.

1.3 Das Copyright

Was ist Freeware ?

Freeware dürfen Sie allen Personen zugänglich machen und kopieren,
(Von mir ausdrücklich gewünscht !! :)) solange für das Programm kein Geld
verlangt und vollständig weitergegeben wird, d.h. es muß dabei sein:

- das Programm "Emperor"
- das Startbild "Emperor.pic"
- die Prefs-Datei "Emperor.prefs"
- das Install-Skript "Emperor.install"
- diese Anleitung "Emperor.guide" (english und deutsch)
- die Katalog-Datei "Catalogs/deutsch/Emperor.catalog"
- die rohe Katalog-Datei "Catalogs/Emperor.cd"
- das Projekt-Verzeichnis "Projects/" mit bereits vorgefertigten Beispiel-Projekten
- das Scripts-Verzeichnis "Scripts/" für die interaktive StormC-Anbindung per ARexx

Sollten Sie eine dieser Dateien nicht im Archiv vorfinden, kontaktieren
Sie mich bitte umgehend !

1.4 Die Systemanforderungen

```
- einen Amiga ;-)
- Speicher mindestens 4 MByte; empfohlen 16 MByte oder mehr
- Workbench Version 3.5 oder höher
- Prozessor mindestens 68020; empfohlen 68060
- Amiga Developer CD 2.1 von der HAAGE & Partner GmbH
- C/C++-Compiler (von der Developer CD "ADCD_2.1:Contributions/Haage_&_Partner/ ↩
   StormC/")
- CatComp (von der Developer CD "ADCD_2.1:NDK/NDK_3.5/Tools/CatComp")
- xen.font in der Größe 8 im "Fonts:"-Verzeichnis des Bootlaufwerks, :-)
  da einige Listen so viele Informationen enthalten, daß ein normaler Screenfont
  viiieeelll zu groß wäre, um alles auf den Bildschirm zu bringen (z.B.  ↩
     Programmierhilfe).


Ausreichend getestet mit folgenden Konfigurationen:

- A 1200
         * 68060/50MHz (BlizzardPPC-Karte)
         * 603e+/240MHz
         * 100 MB Speicher
         * BlizzardVisionPPC - Grafikkarte
         * AmigaOS 3.9 mit BoingBag1

- A 4000
         * 68040/25MHz
         * 64 MB Speicher
         * CyberVision 64/3D - Grafikkarte
         * FastLane Z3
         * AmigaOS 3.5

- A 2000
         * 68040/30MHz
         * 18 MB Speicher
         * GVP Series-II SCSI Filecard
         * AmigaOS 3.5

- A 2000
         * 68040/40MHz
         * 1 MB Speicher Chip
         * 58 MB Speicher Fast
         * Delfina Lite Soundkarte
         * CyberVision 64/3D Grafikkarte
         * 10 GByte Festplatte

- A 1200
         * 68060/50MHz (Blizzard 1260)
         * 1230 SCSI-Kit
         * int. IDE  HD 60MB
         * ext. SCSI HD 3.3GB
         * ext. SCSI CD-Player
         * ext. SCSI ZIP-Laufwerk (250 MB)
         * AmigaOS 3.9 mit BoingBag1
         * AGA-Screen (800x600 Pixel)
```

1.5 Die Installation

Hierzu sollte beiliegender Install-Skript verwendet werden.

Möchten Sie die Installation per Hand ausführen, dann kopieren Sie den

Katalog ("Emperor.catalog") in das Verzeichnis "SYS:Locale/Catalogs/deutsch";

die Programm-Datei ("Emperor"),
das Startbild ("Emperor.pic"),
die Prefs-Datei ("Emperor.prefs"),
diese Guide-Datei ("Emperor.guide")
das Skripte-Verzeichnis ("Scripts/")
und
das Projekt-Verzeichnis ("Projects/")

an einen Ort Ihrer Wahl.

Abschliessend fügen Sie die Zeile:

"Assign Emperor: PROGRAMMPFAD:"

an beliebiger Stelle in Ihre User-startup-Datei ein und starten das System neu.
Danach steht Ihnen das Programm uneingeschränkt zur Verfügung.

"PROGRAMMPFAD:" soll natürlich der Pfad sein, in welches Verzeichnis Sie das Programm kopiert haben.

1.6 Der Programmstart

Der Programmstart kann per CLI (Shell) ohne weitere Argumente erfolgen.

Aber der altbewährte Doppelklick auf's Piktogramm (Icon) des Programms tut's auch.

Außerdem müssen diverse Dinge gewährleistet sein:

* AmigaOS 3.5 installiert ?
* vom Systemspeicher müssen mehr als 4 MBytes frei sein
* Assign auf <PFAD>:Emperor
* RexxMast gestartet ?
* Stack auf mehr als 50000 Bytes eingestellt ?
  - unter CLI: 'stack 50000' eintippen
  - unter WB: in den Piktogramminformationen einen Stack von >50000 Bytes einstellen
  - unter DOpus:in Menü 'Einstellungen-Umgebung' den Punkt CLI-Start anwählen und
    Stack ändern !

1.7 Die Bedienung

```
 Wie das Menü bedient wird...

      Menü


 Wie die einzelnen Fenster Emperors bedient werden...

      Hauptfenster
            Gadgets-Seite
            Requester-Seite
            Menü-Seite
            ASL-Seite
            Locale-Seite
            Arrays-Seite
            GlobVars-Seite
            Windows-Seite
            GadTools-Seite
            InterConnection-Seite

      sonstige Fenster
            Programmierhilfe
            Nachrichtenbehandlung
```

1.8 Das Menü

An dieser Stelle sollen die Menüeinträge aufgelistet und deren Funktion nur kurz beschrieben werden.

```
Neu       - Entfernt das aktuelle Projekt aus dem Speicher und somit alle  ↩
   erstellten Listen (lädt zusätzlich noch das "template.project")
Löschen           - wie "Neu", lädt aber nicht das "template.project"
Öffnen        - Öffnet ein bereits erstelltes Projekt ("*.project"-Datei)
Anhängen      - Fügt ein Projekt dem Aktuellen hinzu

Speichern     (Untermenü)
Projekt speichern   - Speichert das aktuelle Projekt ("*.project"-Datei)
Storm-Projekt speichern - Generiert ein Storm-Projekt ("*.¶"-Datei)
Katalog-Roh-Dateien speichern - Generiert die Katalog-Roh Dateien ("*.cd" und "* ↩
   _SPRACHE.ct"-Datei)
Eigenen Quelltext speichern - Speichert den eigenen Quelltext ("*.h"-Datei)
Katalog generieren    - Generiert die Katalog Datei ("*.catalog"-Datei)
          ACHTUNG ! Funktioniert nur, wenn das Programm
          "CatComp" installiert ist.
Programmquelltext generieren  - Generiert den kompilierbaren C/C++-Quelltext ("*.c ↩
   "-Datei)
Speichern als     - Speichert das aktuelle Projekt unter neuem Namen ("*.project"- ↩
   Datei)
Speichern als Vorgabe   - Speichert das aktuelle Projekt als Vorgabeprojekt ("*. ↩
   project"-Datei)
Clipboard speichern   - Speichert den Clipboardinhalt (also ausgeschnittene  ↩
   Programmteile oder Listen)
Alles speichern   - Speichert folgende Dateien:
          ("*.c"-Datei)    generierter Programm-Code
          ("*.h"-Datei)    Eigener Quelltext
```

```
          ("*.catalog"-Datei)  Locale-Katalog
          ("*.projekt"-Datei)  Emperor-Projekt
          ("*.¶"-Datei)    Storm-Projekt
          ("*_SPRACHE.ct"-Datei) Katalog-Übersetztungsdatei


Ausdrucken      (Untermenü)
Objekttabelle           - Druckt die aktuelle Objekttabelle
Katalog-Roh-Dateien ausdrucken  - Druckt die Katalog-Roh Dateien ("*.cd" und "* ↩
   _SPRACHE.ct"-Datei)
Eigenen Quelltext ausdrucken  - Druckt den eigenen Quelltext ("*.h"-Datei)
Programmquelltext generieren  - Druckt den kompilierbaren C/C++-Quelltext ("*.c"- ↩
   Datei)
Alles ausdrucken    - Druckt folgende Dateien aus:
          ("*.c"-Datei)   generierter Programm-Code
          ("*.h"-Datei)   Eigener Quelltext
          sowie die Objekttabelle


Verwaltung      - Wechselt auf Seite "Windows"
Einstellungen     - Öffnet ein Fenster für diverse Programmeinstellungen
Programmeinstellungen   - Öffnet ein Fenster für diverse Programmspezifische  ↩
   Einstellungen
Information       - Gibt eine Information über das Programm aus
Beenden     - Beendet das Programm



Ausschneiden      - Schneidet den aktuellen Listeneintrag aus
          und speichert ihn im Clipboard
Kopieren      - Speichert das aktuelle Objekt ins Clipboard
Einfügen      - Fügt den Inhalt des Clipboards in eine Liste ein
Eingriff zurück   - Letzter Eingiff wird zurückgenommen
Eingriff erneut   - Eingriff wird erneut durchgeführt
Löschen     - Löscht den aktuellen Listeneintrag


*** Dann folgen diverse Menüpunkte, die nur vom Texteditor aus angewählt werden  ↩
   können.  ***
*** Diese Punkte erklären sich quasi von selbst, deshalb gehe ich nur auf  ↩
   Besonderheiten ***
*** des Punktes "Text einrücken" ein.  ↩
                                                          ***

Text einrücken      - dieser Punkt besitzt 2 Modi:
                                                            * ist das 1.Zeichen (Zeile ↩
                                                               1) ein Leerzeichen,
                                                              rückt Emperor den Text  ↩
                                                                 normal ein.
                                                            * ist das 1.Zeichen KEIN  ↩
                                                               Leerzeichen,
                                                              wird der Text links an  ↩
                                                                 den Rand verschoben
                                                              (es soll Programmierer  ↩
                                                                 geben, die das gern  ↩
                                                                 haben....)
```

```
Testfenster           - Öffnet ein Fenster, das (etwa) dem kompilierten Projekt ↩
   entspricht
Attributfenster   - Öffnet das Fenster, in dem objektabhängige Parametrierungen ↩
   vorgenommen werden können
Zeige eigenen Quelltext - Öffnet den internen Texteditor
Programmierhilfe    - Öffnet das Programmierhilfe-Fenster
Nachrichtenhandhabung   - Öffnet das Nachrichtenbehandlungs-Fenster
```

1.9 Das Hauptfenster

Das Hauptfenster nimmt als Titel den aktuellen Namen des Projektes an. Es besitzt eine Statusleiste, in der jeweils eine kleine Hilfe angezeigt wird, wenn der Mauszeiger über einem Gadget steht.
Weiterhin gibt es Karteikarten, die jeweils eine bestimmte Komponente des Systems beschreiben. So sind hier 9 verschiedene Seiten zu finden:

- Reaction-Seite
- Requester-Seite
- Menü-Seite
- ASL-Seite
- Locale-Seite
- Arrays-Seite
- GlobVars-Seite
- Windows-Seite
- GadTools-Seite
- InterConnection-Seite

Allgemein ist das Fenster für (fast) alle Karteikarten gleich.

Auf der rechten Seite findet sich eine Liste mit Vorgaben für die jeweilige Komponente,
links daneben befinden sich die Buttons:
mit "Hinzufügen" wird ein bereits modifiziertes Element der eigenen Liste hinzugefügt;
mit "Hoch"/"Runter" rutscht der aktive Eintrag in der eigenen Liste nach oben/ unten;
mit "Löschen" löscht man den aktiven Eintrag aus der eigenen Liste;
mit "Test" kann man das Ergebnis der Arbeit bewundern, ohne zu kompilieren.
Auf der linken Seite befindet sich beim Start eine leere Liste, die mit den eigenen
Objekten aufgefüllt wird.

1.10 Das Hauptfenster - die Reaction - Seite

Auf der Gadgets-Seite befinden sich 3 Gruppen von Gadgets:

* Rechts ist die Liste der derzeit implementierten Gadgets. Wenn Sie einen Eintrag
  anwählen, wird das "Attributfenster für Gadgets" aktualisiert, in dem
  dann die gewünschten Eigenschaften zu setzen sind. Die Eigenschaften selbst sind in

dieser Anleitung kaum weiter erläutert, da ich denke das die AutoDocs und Includes
genug Informationen enthalten. Sollten sich aber bei den Attributen Abweichungen
ergeben, werden Sie selbstverständlich ausführlich erläutert.

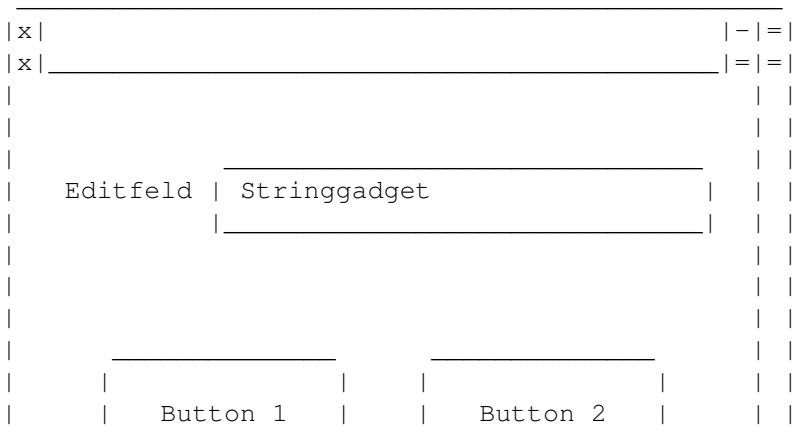
* In der Mitte sind verschiedene Buttons, deren Wirkungsweise sich eigentlich von
selbst klärt. Dennoch will ich deren Funktion kurz beschreiben:
-Add fügt die Daten, die gerade im Attributfenster eingestellt wurden, in die eigene Liste ein
-Up verschiebt den angewählten Eintrag in der eigenen Liste um eins nach oben (um eine Genration höher)
-Down verschiebt den angewählten Eintrag in der eigenen Liste um eins nach unten (um eine Genration runter)
-Delete Löscht den angewählten Eintrag in der eigenen Liste
mit -Test kann man das Ergebnis der Arbeit bewundern, ohne zu kompilieren.

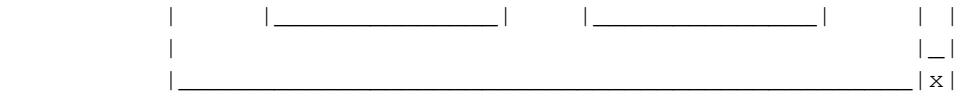
* Links befindet sich eine leere Liste, die später mit der eigenen hierarchischen
Liste von Gadget-Objekten aufgefüllt werden soll.
Als Einträge werden jeweils die Namen der Gadgets angenommen.
Ein hinzugefügtes Gadget wird jeweils auf die nächsthöhere "Generation" gestellt ;
so ist es immer ein abgeleitetes Gadget des Vorausgegangenen.

Ziel dieser "neuen" Art, Oberflächen zu erstellen ist es, mehr und mehr Arbeit dem Programmierer abzunehmen. Die Positionen und Ausmaße werden automatisch durch das System, in Relation zu der Fenstergröße, bestimmt.
Sie können die Anordnung der Gadgets beeinflussen, indem sie vertikale bzw. horizontale
Gruppen (Layouts) einfügen. Gadgets, die einem bestimmten Layout zugehörig sein sollen, müssen von dem Layout-Gadget abgeleitet werden, d.h., daß sie sich in der Liste weiter rechts anordnen und somit eine eigene Unterklasse bilden.
So können auch horizontale und vertikale Layout-Gruppen wiederum in horizontale und vertikale Layout-Gruppen aufteilen, wie es das 2. Beispiel zeigen wird.

Horizontale Gruppe bedeutet, daß sich alle Gadgets horizontal, also nebeneinander, aufbauen. Entsprechend gilt dasselbe für die vertikalen Layouts.

Beispiel 1:
Einfaches Beispiel mit einem String-Gadget (Eingabefeld) und zwei Buttons.



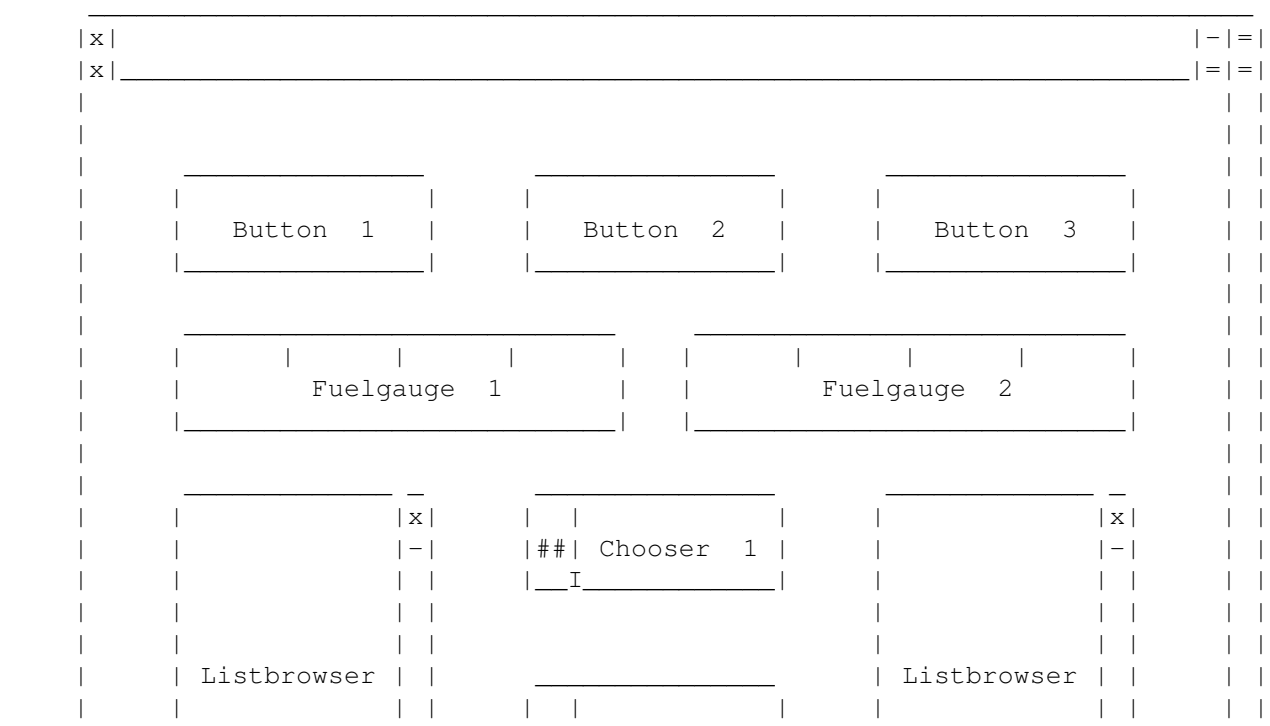


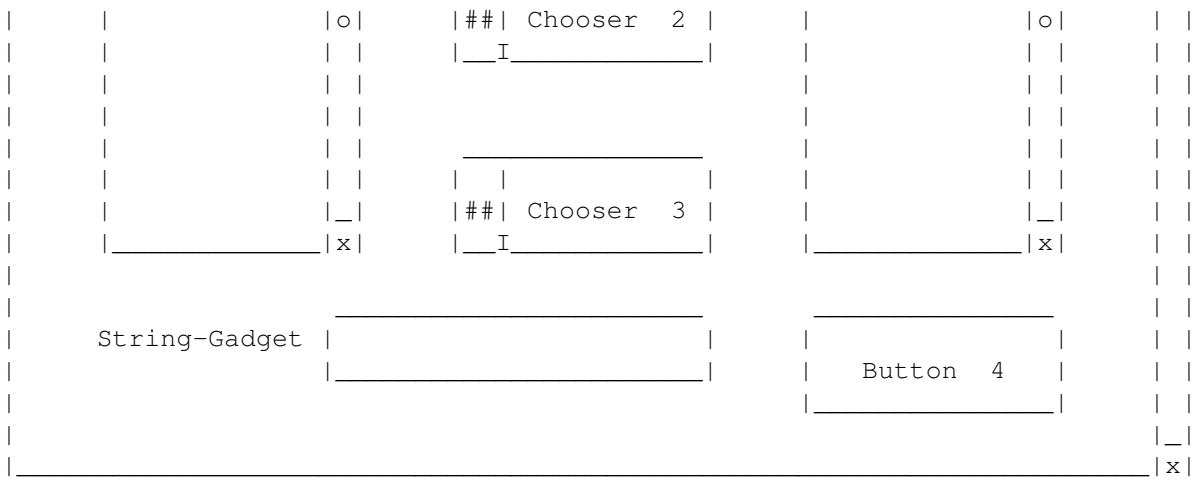
Die Liste für dieses Beispiel muß nun so aussehen:

```
Layout1       das sog. "Root-Layout" bestimmt jeden Beginn
|       einer Gadgetliste. In diesem Fall sollte es
|       mit dem Attribut "Orientation-vertical"
|       ausgestattet sein, weil die 2 Gadget-Gruppen
|       (1. String-Gadget / 2. beide Buttons)
|       untereinander angeordnet sind.
|
+- String1      das String-Gadget ist das erste sichtbare
|       Objekt in unserer Liste.
|
+- horizontal Layout2   eine erneute Aufteilung des Raumes findet
   |        hier statt. Zuerst hatten wir das gesamte
   |        Fenster aufgeteilt in untereinander liegende
   |        Gruppen; in der einen befand sich das String-
   |        Gadget und in der anderen erstellen wir jetzt
   |        eine Gruppe, die nebeneinander liegende Gadgets
   |        beinhalten wird.
   |
   +- Button1     dies ist der linke Button
   +- Button2     das ist der rechte
```

Eigentlich gar nicht so schwer, oder ?
Naja, dann zu umfangreicherem:

Beispiel 2:






```
Layout1
|
+- vertical Layout
   |
   +- horizontal Layout
   |  |
   |  +- Button1
   |  +- Button2
   |  +- Button3
   |
   +- horizontal Layout
   |  |
   |  +- Fuelgauge1
   |  +- Fuelgauge2
   |
   +- horizontal Layout
   |  |
   |  +- Listbrowser
   |  |
   |  +- vertical Layout
   |  |  |
   |  |  +- Chooser1
   |  |  +- Chooser2
   |  |  +- Chooser3
   |  |
   |  +- Listbrowser
   |
   +- horizontal Layout
      |
      +- String
      +- Button4
```

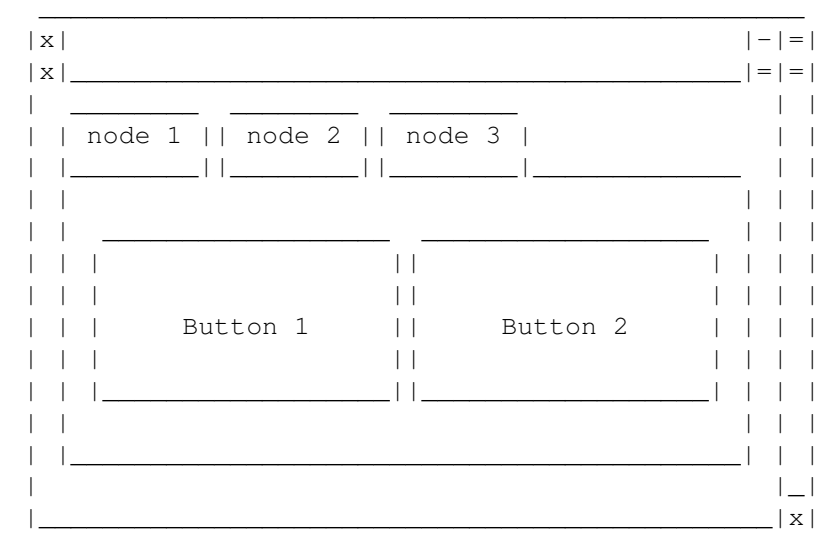
Spielen Sie ruhig ein wenig mit den horizontalen und vertikalen Gruppen herum, damit Sie ein Gefühl dafür bekommen. Wenn Sie bisher immer mit der Gadtools- ↩ Library gearbeitet und absolute Werte für Ausmaße und Positionen eingegeben haben, ist die Umstellung machmal gar nicht so einfach ! Aber, ...

Übung macht den Meister !

SPEZIELLE INFORMATIONEN...

ad 1 (Benutzung von Clicktab-Gadgets):

Falls Clicktab-Gadgets benutzt werden sollen, MUSS die Gadget-Liste in etwa so ↩ aussehen:




```
Clicktab
|
+- vertical Page1
   |
   +- horizontal Page2
   |  |
   |  +- Button1
   |  +- Button2
   |  |
   +- vertical Page3
   |  |
   |  +- Integer1
   |  +- Integer2
   |  |
   +- horizontal Page4
      |
      +- Listbrowser1
      +- Listbrowser2
```

"Pages" werden generiert, indem ein Layout-Gadget in den "PageMode" versetzt wird. (Attribut-Fenster)

ACHTUNG ! Niemals das erste Page-Gadget unter dem Clicktab vergessen !!!!

ad 2 (Einbinden von Images):

Images sind spezielle Gadgets, die Bildinformationen (geladene Bilder, ↩ Systemgadgets etc.)

enthalten. Möchten Sie ein solches Image auch in Ihr Programm integrieren, sollten ↩
Sie
darauf achten, daß das Image vom abhängigen Gadget entweder abgeleitet ist, bzw. ↩
dem Gadget
die Zuordnung via Image-Verbindung im Attributfenster erfolgt. Sie dürfen jedoch ↩
ein Image
lediglich von einem Button, einem Layout oder einem Slider ableiten; alle anderen ↩
Gadgets
verbieten dies !!!!

Ein Beispiel könnte etwa so aussehen:

```
horizontal Layout1
|
+- vertical Layout2
|  |
|  +- Bitmap1
|  |
+- vertical Layout3
   |
   +- Button1
   |  |
   |  +- Glyph1
   |  |
   +- Slider1
      |
      +- Bitmap2
```

SPEZIELLE INFORMATIONEN...

1.11 Das Hauptfenster - die Requester - Seite

Auf der Requester-Seite finden sich in der rechten Liste verschiedene Einträge, ↩
über
die Requester (Dialogfelder) wie z.B. für Informationen, Beenden oder ↩
Fehlermeldungen
generiert werden können. Ein einfacher Klick auf den entsprechenden Eintrag
(und mit angewähltem Menüeintrag "Fenster-Attributfenster") und schon kann dieser
editiert werden, um ihn danach in die eigene Liste einzufügen.

Es gibt bereits vorgefertigte Einträge, wie z.B. das Quitrequest (ein ↩
Dialogfenster,
daß eine eigenständige Erstellung zum Beenden überflüssig macht), das erfragt,
ob das Programm (hier geht es um das Selbstgeschriebene und nicht um Emperor)
wirklich beendet werden soll, wie man es von diversen Applikationen und Tools her ↩
kennt.

Der Eintrag Emptyrequest kann bei allen anderen (nicht vorgefertigten) angewandt
werden.

Die Liste mit den bereits erstellten Requestern wird in naher Zukunft noch stark
erweitert (z.B. "Möchten Sie die Datei überschreiben ?" etc.)

WICHTIG:Für den Requester-Bodytext (der Text, der die nähere Auskunft neben dem Titel
und den Gadgets definiert) darf maximal 500 Zeichen lang sein !

1.12 Das Hauptfenster - die Menü - Seite

Im Menüeditor sind in der rechten Liste (bisher) 6 Einträge zu finden, wobei die ersten
3 Einträge die Systemdefinitionen repräsentieren (Titel, Item, Subitem) -> siehe AutoDocs;

Die anderen drei Einträge sind komplette Menüs.
Project steht für das bekannte Projekt-Menü (Öffnen, Speichern, Drucken etc.).
Edit ist das Menü "Bearbeiten" (Ausschneiden, Kopieren, Einfügen etc.).
Prefs ist das Standart-Menü für Einstell-Programme wie "Time", "Font", "Serial" usw.

Der Eintrag muß nur angewählt und der eigenen Liste hinzugefügt werden. Schon hat man
das komplette Menü INKLUSIVE ÜBERSETZUNGEN parametriert. Die einzelnen Einträge können
dann auch selbstverständlich verändert und dem eigenen Geschmack angepaßt werden.

1.13 Das Hauptfenster - die ASL - Seite

Hier kann man die Requester (Dialogfenster) für Datei-, Schrift- und Bildschirmauswahl
verändern und nach Herzenslust ausprobieren, bis sie den Vorstellungen entsprechen .

Siehe AutoDocs !

1.14 Das Hauptfenster - die Locale - Seite

Auf der Locale-Seite muß für die Erstellung eines Kataloges selbst nichts getan werden.
Die Katalog-Identifikationen werden jeweils aus den anderen Teilen des Programms
(Requester, Menü etc.) in diese Liste überführt.

Nur die Übersetzung selbst wird hier vorgenommen. Hierzu muß die gewünschte Sprache
gewählt werden und zusätzlich die zu übersetzende Katalog-ID.
Dann ist einfach nur noch (bei geöffnetem Attributfenster Menü-"Fenster- Attributfenster")
das mittlere Feld anzuwählen und die darunter stehende Zeichenkette zu übersetzen.

1.15 Das Hauptfenster - die Arrays - Seite

```
Bei der Erstellung eines Arrays (Einträge für Listbrowser, Radiobutton, Chooser ↩
    und Clicktabs)
MUß das Menü-"Fenster-Attributfenster" geöffnet werden.
Hier, im Attributfenster ist wiederum eine Liste, die mit Einträgen des Arrays ↩
    gefüllt
werden muß, bevor der Eintrag (Name des Arrays) zu der Liste im Hauptfenster ↩
    hinzugefügt
werden kann.
```

1.16 Das Hauptfenster - die GlobVars - Seite

```
Hier kann zwischen diversen Variablentypen gewählt werden. Um sie später der
eigene Liste hinzuzufügen, müssen (wie immer) die spezifischen Einstellungen ↩
    vorgenommen
werden.

Das erste Gadget (von oben) kann nur genutzt werden, wenn eine Systemstruktur
(siehe AutoDocs) aufgerufen und initialisiert werden soll.
Das zweite Eingabefeld bestimmt den Variablennamen, mit dem im Quelltext selbst
gearbeitet werden kann.
Das dritte bestimmt den initial zugewiesenen Wert.
Das unterste ist nur als Hilfe gedacht, in dem der Wertebereich der aktuellen ↩
    Variable
angezeigt wird.
```

1.17 Das Hauptfenster - die Windows - Seite

```
Dies ist diejenige Seite, in der man die Fenster, die man in das eigene Programm
implementieren will, konfiguriert.

Hierbei ist zu beachten, daß das jeweils aktive Fenster auch mit "Active"
gekennzeichnet ist. Das wiederum bedeutet, daß, abhängig vom gewählten aktiven ↩
    Fenster,
die Gadgetlisten sich dem aktiven Fenster unterordnen und damit aktualisiert ↩
    werden.
D.h., NUR wenn ein Fenster "Active" ist, kann auch sein Inhalt (Gadgets) geändert ↩
    werden.


 !!WICHTIG!!

Im Attributfenster selbst kann nun festgelegt werden, ob ein Fenster im
AmigaOS 2.0 oder AmigaOS 3.5 - Look erscheinen soll. Ist ein OS 2.0 - Look gewählt ↩
    , können
ihm keine Reaction-Gadgets zugewiesen werden !

 !!WICHTIG!!
```

1.18 Das Hauptfenster - die GadTools - Seite

Da GadTools lediglich über Gadgets verfügt, die ihrerseits absolute Positionen verlangen, ist
die Handhabung etwas komplizierter (oder logischer ? :))).

Für die Erstellung eines Fensters mit Gadtools-Gadgets (OS 2.0 - Look) muß das Fenster selbst
logischerweise ein OS 2.0 - Fenster sein.
Dies ist im Attributfenster auf der Windows-Seite einstellbar.
Wichtig ist nun, daß das Fenster, in welches man das Gadget positionieren will, auch offen ist.
Nun kann auf der GadTools-Seite das gewünschte Gadget gewählt werden, und im zweiten Schritt
die Lage und Größe mittels Mausklick (und Mausbewegung) in das Fenster übertragen werden.
Nachträgliche Änderungen sind ebenso, durch einfaches Fassen der gekennzeichneten
Randelemente, möglich. Will man ein anderes Gadget verschieben oder vergrößern/ verkleinern,
genügt für die Aktivierung schon ein einfacher Klick auf das Gadget selbst.

Die linksseitige Liste ist bei der GadTools-Seite deaktiviert.

Durch den Support von OS 2.0 - Fenstern und Gadgets ist es möglich, ein Programm zu generieren,
welches sowohl auf Basis von OS 2.0, als auch auf OS 3.5 läuft.

1.19 Mainwindow - Interconnection Map - Page

Interconnection Maps sind dazu da, Gadgets untereinander zu verknüpfen. Die Verknüpfung läßt es zu,
daß man verschiedene Tags (Eigenschaften) und deren Werte zwischen den Gadgets austauschen lassen kann.

Wird z.B. ein Fuelgauge-Gadget mit einem Paletten-Gadget verbunden, muß auch über die geforderten Tags
(Eigenschaften) Klarheit bestehen. So kann z.B. der Tag "FUELGAUGE_Level" mit " PALETTE_Colour"
verbunden werden. Jedesmal, wenn die Palette einen anderen Wert erhält (durch klicken) wird diese
Veränderung direkt auf die Fuelgauge übertragen.

Zur Unterscheidung gibt es unidirektionale und bidirektionale Maps;
unidirektionale Maps werden nur Informationen vom primären zum sekundären Gadget übertragen,
bidirektionale Maps werden Informationen vom primären zum sekundären sowie vom sekundären zum
primären Gadget übertragen.

Um die Gadgets nun Daten austauschen zu lassen, muß die ICMap der eigenen Liste hinzugefügt werden.
Dann sind die Gadgets untereinander verknüpft, ohne daß auch nur ein einziges Byte Quelltext
getippt wurde !

1.20 Die Programmierhilfe

Die Programmierhilfe befindet sich noch in einem sehr frühem Stadium der Entstehung. Von hier aus kann man die vorgefertigten Funktionsmakros auswählen, parametrieren und in den eigenen Quelltext einfügen.

1. Auswählen der gewünschten Funktion durch anklicken
2. Ein Fenster öffnet sich, in dem die Funktionsparameter eingestellt werden können.
3. Den Button "Funktion dem eigenen Quelltext hinzufügen" drücken
4. Zwei Fenster, der Texteditor und Nachrichtenbehandlung öffnen sich (4a) ↩ ,
   falls der Texteditor selbst noch nicht offen war (4b).
   - 4a. Im Fenster Nachrichtenbehandlung die Nachricht auswählen, bei der die Funktion eingefügt werden soll. Dies kann auch eine noch nicht bestehende Funktion sein.
   - 4b. Ist der Texteditor bereits geöffnet, wird die Funktion an der Position des Cursors eingefügt.

Wenn eine Funktion ausgewählt wurde, kann mit HELP die Onlinehilfe (Emperor.guide) ↩ gestartet
werden, das sich auf derjenigen Seite öffnet, auf der die Funktion beschrieben ist ↩ .

1.21 Die Nachrichtenbehandlung

Dieses Fenster teilt sich in zwei Listen auf. Die Liste auf der linken Seite enthält eine Vielzahl von möglicherweise auftretenden Ereignissen.

Die Einträge bedeuten im Einzelnen:

GadgetUp	- wenn ein Gadget angeklickt wurde
GadgetHelp	- wenn der Mauszeiger über einem Gadget steht
MenuPick	- wenn ein Eintrag aus dem Menü gewählt wurde
MenuHelp	- wenn der Mauszeiger über einem Menüeintrag steht
CloseWindow	- wenn das Fensterschließsymbol gedrückt wurde
Iconify	- wenn das Iconifizierungssymbol des Fensters gedrückt ↩ wurde
Uniconify	- wenn das Piktogramm (Icon) doppelt geklickt wurde
ActiveWindow	- wenn ein Fenster aktiviert wurde
InactiveWindow	- wenn ein Fenster in den Zustand "Inaktiv" gegangen ist
RawKey	- wenn eine Taste gedrückt wurde
VanillaKey	- wenn eine Taste gedrückt wurde (Menü-Shortcut)
MouseButtons	- wenn eine Maustaste gedrückt wurde
MouseMove	- wenn eine Mausbewegung ausgeführt wurde
NewSize	- wenn sich die Größe eines Fensters geändert hat
ChangeWindow	- wenn sich die Größe oder Position eines Fenster geändert ↩ hat
DisposedWindow	- wenn ein Fenster eine Aktion ausgeführt hat
IntuiTick	- alle 1/10-Sekunden eine Nachricht
ShowWindow	- wenn ein Fenster geöffnet wird
Startup	- bei Programmstart (zur Initialisierung)
Shutdown	- bei Programmende

```
        Für genauere Ausführungen der Nachrichten des Systems,
        konsultieren Sie bitte die AutoDocs der AmigaDeveloperCD!

Wird nun einer dieser Einträge angewählt, erscheinen die dazugehörigen
Auswahlmöglichkeiten zur Spezifizierung des gewünschten
Gadgets/Menü/Fensters-Objekts in der Liste zur Rechten.
Ein Doppelklick auf den entsprechenden Eintrag in der rechten Liste, generiert
eine neue Funktion (Prozedur) im Texteditor, in der dann eigener Quelltext
hinzugefügt werden kann. Das können z.B. die vorgefertigten Funktionsmakros
sein, oder die Systemfunktionen des AmigaOS.
```

1.22 vordefinierte Variablen

```
Variablenname   Art                     enthält

os35            (BOOL)                  = TRUE, falls auf dem System OS 3.5  ↩
   installiert ist
terminated      (BOOL)                  = TRUE, wenn das Programm verlassen werden ↩
    soll
vinfo           (void *)                = GetVisualInfo()
Catalog         (struct Catalog *)      = OpenCatalogA()
Menu            (struct Menu *)         = CreateMenus()
AppPort         (struct MsgPort *)      MsgPort für die Fenster und deren  ↩
   Auswertung
Screen1         (struct Screen *)       = LockPubScreen()

Der Variablenname "Screen1" wurde für Version 4.0 geändert. Sie entspricht der  ↩
   damaligen Variable "Screen".
```

1.23 Die Funktionen

```
In Emperor gibt es einige Makrofunktionen, die z.T. komplizierte Operationen mit
einem einfachen Aufruf und den entsprechenden Parametern ausführen. Diese Makros
sind aus dem eigenen Quelltext heraus ansteuerbar.

 Fensterhandling
      Emperor_CloseWindow
      Emperor_OpenWindow
      Emperor_IconifyWindow
      Emperor_UniconifyWindow

 Fensterattribute
      Emperor_ChangeWindowPosition
      Emperor_ChangeWindowSize
      Emperor_SetWindowBusyPointer
      Emperor_SetWindowTitle
      Emperor_SetScreenTitle

 Gadgetattribute
      Emperor_GetGadgetAttr
      Emperor_GetGadgetAttrComplex
```

```
     Emperor_GetGadgetDisabledAttr
     Emperor_GetGadgetReadOnlyAttr
     Emperor_SetGadgetAttr
     Emperor_SetGadgetAttrComplex
     Emperor_SetGadgetDisabledAttr
     Emperor_SetGadgetReadOnlyAttr

Menuattribute
     Emperor_GetMenuItemAttr
     Emperor_SetMenuItemAttr

Verschiedene
     Emperor_ActivateGadget
     Emperor_RethinkLayout
     Emperor_RefreshGadget
     Emperor_QuitFunc
     stringlength()
     stringtoint()
     inttostring()
```

1.24 Die Funktion Emperor_CloseWindow

```
Funktion
 Makrofunktion für das Schließen von Fenstern

Synopsis
 Emperor_CloseWindow(WindowObject)

 void Emperor_CloseWindow(Object *);

Aktion
 Diese Funktion erledigt mehrere Aufgaben für den Programmierer.
 1. Zunächst entfernt sie das erstellte Menü (falls vorhanden !)
 2. Schließt sie das Fenster
 3. Setzt die Variable Window zurück.

Parameter
 WindowObject - Hier muß der Name des Fensterobjektes angegeben werden.
          Diesen stellen Sie im Projekt-Manager ein, wenn Sie ein
          neues Fenster erstellen. Vorgabe ist immer eine so auf-
          baute Variable:
     Object *WindowObjX; wobei X das Fenster bestimmt;
     also z.B. Object *WindowObj1, *WindowObj2...;

Beispiel
 /* Das Schließen eines Fensters muss unbedingt manuell erfolgen, */
 /* da diese Aktionen der Speicherfreigabe nicht vom Hauptcode    */
 /* übernommen werden können. Wenn das Applikationsfenster        */
 /* geschlossen wird, entfernt Emperor aber noch lange nicht das  */
 /* Programm. Hierfür sollte die Funktion Emperor_QuitFunc().    */
 /* Dennoch gibt es durchaus vielfältige Anwendungsmöglichkeiten  */
 /* für diese Funktion.                                           */

 void Emperor_Chooser1GadgetUpEvent(void)
 {
```

```
    /*
    ... Abzuarbeitender Quelltext ...
    ...
    ...
    ...
    */

    Emperor_CloseWindow(WindowObj1);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */

    Window1 = Emperor_OpenWindow(WindowObj1);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }

Siehe auch
  Emperor_OpenWindow()
```

1.25 Die Funktion Emperor_OpenWindow

```
Funktion
  Makrofunktion für das Öffnen von Fenstern

Synopsis
  Window = Emperor_OpenWindow(WindowObject)

  struct Window *Emperor_OpenWindow(Object *);

Aktion
  Diese Funktion erledigt mehrere Aufgaben für den Programmierer.
  1. Öffnet sie das Fenster
  2. Fügt sie das erstellte Menü (falls vorhanden !) hinzu

Parameter
  WindowObject - Hier muß der Name des Fensterobjektes angegeben werden.
           Diesen stellen Sie im Projekt-Manager ein, wenn Sie ein
           neues Fenster erstellen. Vorgabe ist immer eine so auf-
           baute Variable:
       Object *WindowObjX; wobei X das Fenster bestimmt;
       also z.B. Object *WindowObj1, *WindowObj2...;

Ergebnis
  ist eine Variable vom Typ "struct Window *", also auf die alt-bekannte
  Intuition-Struktur. Dies sollte (vorteilhafterweise) die Variable sein,
```

```
 deren Namen Sie beim Erstellen des Fensters im Projekt-Manager eingestellt
 haben. Sollten Sie die Vorgabe nicht verändert haben, ist diese immer eine
 so aufbaute Variable:
   Object *WindowX; wobei X das Fenster bestimmt;
   also z.B. struct Window *Window1, *Window2...;

Beispiel
 void Emperor_Button1GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_CloseWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */

   Window1 = Emperor_OpenWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_CloseWindow()
```

1.26 Die Funktion Emperor_IconifyWindow

```
Funktion
 Makrofunktion für das Ikonifizieren von Fenstern

Synopsis
 Emperor_IconifyWindow(WindowObject)

 void Emperor_IconifyWindow(Object *);

Aktion
 Diese Funktion erledigt mehrere Aufgaben für den Programmierer.
 1. Zunächst entfernt sie das erstellte Menü (falls vorhanden !)
 2. Schließt sie das Fenster
 3. Setzt die Variable Window zurück.

Parameter
```

```
 WindowObject - Hier muß der Name des Fensterobjektes angegeben werden.
          Diesen stellen Sie im Projekt-Manager ein, wenn Sie ein
          neues Fenster erstellen. Vorgabe ist immer eine so auf-
          baute Variable:
      Object *WindowObjX; wobei X das Fenster bestimmt;
      also z.B. Object *WindowObj1, *WindowObj2...;

Beispiel
 /* Das Ikonifizieren eines Fensters muss unbedingt manuell */
 /* erfolgen, da diese Aktionen der Speicherfreigabe nicht  */
 /* vom Hauptcode übernommen werden können.                 */

 void Emperor_Checkbox1GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_IconifyWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */

   Window = Emperor_UniconifyWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_UniconifyWindow()
```

1.27 Die Funktion Emperor_UniconifyWindow

```
Funktion
 Makrofunktion für das Rückikonifizieren von Fenstern

Synopsis
 Window = Emperor_UniconifyWindow(WindowObject)

 struct Window *Emperor_UniconifyWindow(Object *);

Aktion
 Diese Funktion erledigt mehrere Aufgaben für den Programmierer.
```

```
 1. Öffnet sie das Fenster
 2. Fügt sie das erstellte Menü (falls vorhanden !) hinzu

Parameter
 WindowObject - Hier muß der Name des Fensterobjektes angegeben werden.
          Diesen stellen Sie im Projekt-Manager ein, wenn Sie ein
          neues Fenster erstellen. Vorgabe ist immer eine so auf-
          baute Variable:
      Object *WindowObjX; wobei X das Fenster bestimmt;
      also z.B. Object *WindowObj1, *WindowObj2...;

Ergebnis
 ist eine Variable vom Typ "struct Window *", also auf die alt-bekannte
 Intuition-Struktur. Dies sollte (vorteilhafterweise) die Variable sein,
 deren Namen Sie beim Erstellen des Fensters im Projekt-Manager eingestellt
 haben. Sollten Sie die Vorgabe nicht verändert haben, ist diese immer eine
 so aufbaute Variable:
   Object *WindowX; wobei X das Fenster bestimmt;
   also z.B. struct Window *Window1, *Window2...;

Beispiel
 void Emperor_Checkbox1GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_IconifyWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */

   Window = Emperor_UniconifyWindow(WindowObj1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }


Siehe auch
 Emperor_IconifyWindow()
```

1.28 Die Funktion Emperor_ChangeWindowPosition

```
Funktion
 Makrofunktion für das Setzen der Window-Attribute.

Synopsis
 Emperor_ChangeWindowPosition(Window, Left, Top)

 void Emperor_ChangeWindowPosition(struct Window *, WORD, WORD);

Aktion
 Dieses Makro benutzt die ChangeWindowBox()-Funktion der Intuition-Library.
 Es wird die Position des Fensters modifiziert.

Parameter
 Window - das zu verändernde Fenster. Beispiel: Window1
 Left   - Linke Ecke des Fensters relativ zum linken Bildschirmrand.
 Top    - Obere Ecke des Fensters relativ zum oberen Bildschirmrand

Beispiel
 void Emperor_WindowPositionMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_ChangeWindowPosition(Window, 100, 200);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_ChangeWindowSize()
```

1.29 Die Funktion Emperor_ChangeWindowSize

```
Funktion
 Makrofunktion für das Setzen der Window-Attribute.

Synopsis
 Emperor_ChangeWindowSize(Window, Width, Height)

 void Emperor_ChangeWindowSize(struct Window *, WORD, WORD);

Aktion
 Dieses Makro benutzt die ChangeWindowBox()-Funktion der Intuition-Library.
 Es wird die Größe des Fensters modifiziert.
```

```
Parameter
 Window - das zu verändernde Fenster. Beispiel: Window1
 Width  - Breite des Fensters
 Height - Höhe des Fensters

Beispiel
 void Emperor_WindowSizeMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_ChangeWindowSize(Window, 300, 400);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_ChangeWindowPosition()
```

1.30 Die Funktion Emperor_SetWindowBusyPointer

```
Funktion
 Makrofunktion für das Setzen der Window-Attribute.

Synopsis
 Emperor_SetWindowBusyPointer(Window, on/off)

 void Emperor_Window(struct Window *, BOOL);

Aktion
 Dieses Makro benutzt die SetWindowPointer()-Funktion der Intuition-Library.
 Es wird der Mauszeiger beim Aktivieren des Fensters in den "Busy"-Modus
 gesetzt.

Parameter
 Window - das zu verändernde Fenster. Beispiel: Window1
 on/off - schaltet bei "TRUE" den BusyPointer an und bei "FALSE" ab.

Beispiel
 void Emperor_WindowBusyPointerMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
```

```
    */

    Emperor_SetWindowBusyPointer(Window);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }
```

1.31 Die Funktion Emperor_SetWindowTitle

```
Funktion
 Makrofunktion für das Setzen der Window-Attribute.

Synopsis
 Emperor_SetWindowTitle(Window, windowtitle)

 void Emperor_SetWindowTitle(struct Window *, STRPTR);

Aktion
 Dieses Makro benutzt die SetWindowTitles()-Funktion der Intuition-Library.
 Es wird der Titel des Fensters modifiziert.

Parameter
 Window - das zu verändernde Fenster. Beispiel: Window1
 windowtitle - der neue Titel für das Fenster

Beispiel
 void Emperor_WindowTitleMenuEvent(void)
 {
    /*
    ... Abzuarbeitender Quelltext ...
    ...
    ...
    ...
    */

    Emperor_SetWindowTitle(Window, "new title");

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
 }

Siehe auch
 Emperor_SetScreenTitle()
```

1.32 Die Funktion Emperor_SetScreenTitle

```
Funktion
 Makrofunktion für das Setzen der Window-Attribute.

Synopsis
 Emperor_SetScreenTitle(Window, screentitle)

 void Emperor_SetScreenTitle(struct Window *, STRPTR);

Aktion
 Dieses Makro benutzt die SetWindowTitles()-Funktion der Intuition-Library.
 Es wird der Titel des Fensters modifiziert.

Parameter
 Window - das zu verändernde Fenster. Beispiel: Window1
 screentitle - der neue Titel für den Screen

Beispiel
 void Emperor_WindowTitleMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_SetScreenTitle(Window, "new title");

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetWindowTitle()
```

1.33 Die Funktion Emperor_GetMenuItemAttr

```
Funktion
 Makrofunktion für das Erhalten eines Menüeintrag-Attributes.

Synopsis
 result = Emperor_GetMenuItemAttr(MenuObject, MenuattrID)

 BOOL Emperor_GetMenuItemAttr(UBYTE, UWORD);

Aktion
 Die Funktion erfragt den aktuellen Zustand der Attribute von Menüeinträgen.
 Es kann z.B. erfragt werden, ob ein Eintrag mit Checkit-Eigenschaften
```

```
 (angewählt, wenn das Check-Image (links ein kleiner Haken) zu sehen ist)
 angewählt ist oder nicht.

Parameter
 MenuObject   - Dieser Parameter gibt den Namen des Menü-Objektes an,
          der bei der Erstellung des Menüs angegeben werden kann.
          Diese Lösung ist sehr vorteilhaft, denn die Verwaltung
          von Menüs und deren ID-Nummern ist recht kompliziert
          für Einsteiger. Auch ist es bei falschen Parametern durchaus
          möglich, daß der Rechner abstürzt.
 MenuFlagmask - Dieser Parameter erwartet eine von den folgenden drei Masken,
          die angeben sollen, welches Attribut erfragt werden soll:
     * ITEMENABLED (ob der Eintrag anwählbar ist)
     * MENUTOGGLED (für MutualExclude oder Checkit; erfragt,
               ob der Eintrag gewechselt wurde)
     * CHECKED     (für MutualExclude oder Checkit; erfragt,
               ob der Eintrag angewählt ist)

Ergebnis
 Das Ergebnis ist eine Variable vom Typ BOOL (wahr/falsch-Aussage).
 result = TRUE (wahr), wenn das ausgewählte Attribut gesetzt ist; ansonsten
 ist das Ergebnis result = FALSE (falsch).

Beispiel
 void Emperor_ItemCheckitMenuEvent(void)
 {
   BOOL boolean;

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   boolean = Emperor_GetMenuItemAttr(Menu_ItemCheckit, CHECKED);
   if(boolean) printf("Der Menü-Eintrag ist mit einem Haken angewählt");
   else printf("Der Menü-Eintrag hat keinen Haken");

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetMenuItemAttr()
```

1.34 Die Funktion Emperor_SetMenuItemAttr

```
Funktion
 Makrofunktion für das Setzen eines Menüeintrag-Attributes.
```

```
Synopsis
 Emperor_SetMenuItemAttr(MenuObject, MenuattrID, Set/Reset)

 void Emperor_SetMenuItemAttr(UBYTE, UWORD, BOOL);

Aktion
 Die Funktion kann den aktuellen den Zustand der Attribute von Menüeinträgen
 verändern. Es kann z.B. ein Eintrag mit Checkit-Eigenschaften
 (angewählt, wenn das Check-Image (links ein kleiner Haken) zu sehen ist)
 angewählt werden oder einen Menüeintrag unanwählbar (disabled) machen.

Parameter
 MenuObject   - Dieser Parameter gibt den Namen des Menü-Objektes an,
          der bei der Erstellung des Menüs angegeben werden kann.
 MenuFlagmask - Dieser Parameter erwartet eine von den folgenden zwei Masken,
          die angeben sollen, welches Attribut gesetzt werden soll:
     * ITEMENABLED (macht den Eintrag nicht anwählbar)
     * CHECKED (für MutualExclude oder Checkit; wählt einen Eintrag an)
 Set/Reset    - Gibt an, ob die Maske gesetzt oder gelöscht werden soll.

Beispiel
 void Emperor_ItemCheckitMenuEvent(void)
 {
   BOOL boolean;

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_SetMenuItemAttr(Menu_ItemCheckit, ITEMENABLED, TRUE);
   Emperor_SetMenuItemAttr(Menu_ItemCheckit, CHECKED, FALSE);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_GetMenuItemAttr()
```

1.35 Die Funktion Emperor_ActivateGadget

```
Funktion
 Makrofunktion für das Aktivieren eines String- oder CustomGadgets.

Synopsis
 Emperor_ActivateGadget(GadgetObject)

 void Emperor_ActivateGadget(struct Gadget *);
```

```
Aktion
 Beim Aufruf dieser Funktion wird ein StringGadget aktiviert und der Cursor
 in das Gadget gesetzt, so das der User sofort, ohne zu klicken, Eingaben
 machen kann. Dieses Makro nutzt die Intuition-Funktion ActivateGadget()

Parameter
 GadgetObject - Name des Gadgets, das aktiviert werden soll.

Beispiel
 void Emperor_ActivateMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_ActivateGadget(String3);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }
```

1.36 Die Funktion Emperor_QuitFunc

```
Funktion
 Makrofunktion für das Verlassen des Programms.

Synopsis
 Emperor_QuitFunc()

 void Emperor_QuitFunc(void);

Aktion
 Beim Aufruf dieser Funktion werden alle zuvor geöffneten Libraries etc.
 wieder geschlossen. Das bedeutet, geschlossen werden:
   * Libs
   * Catalog
   * geöffnete Fenster
   * allokierte Speicherbereiche für ASL-Requester, Menüs und sonstige.
 Vor Aufruf dieser Funktion müssen die Fenster nicht geschlossen werden !

Beispiel
 void Emperor_ProjectQuitMenuEvent(void)
 {
   BOOL error;

   /*
   ... Abzuarbeitender Quelltext ...
```

```
    ...
    ...
    ...
    */

    if(error) Emperor_QuitFunc();

    /*

    Ende

    */
  }
```

1.37 Die Funktion Emperor_RethinkLayout

```
Funktion
 Makrofunktion für das Neuzeichnen des gesamten Layouts.

Synopsis
 Emperor_RethinkLayout(Window, GadgetObject)

 void Emperor_RethinkLayout(struct Window *, struct Gadget *);

Aktion
 Dieses Makro nutzt die LayoutGadget-Funktion RethinkLayout().
 Beim Aufruf dieser Funktion wird das gesamte Fenster neugezeichnet.
 Dabei muß beim "Refreshing" jedoch nicht die gesamte hierarchische
 Liste der Gadgets neugezeichnet werden, sondern ist als Parameter
 mit definierbar. Wird ein Name eines Gadgets angegeben, wird ab
 dieser Position in der Liste der Gadgets alle nachfolgenden Objekte
 refresht.
 Wenn alles neugezeichnet werden soll, muß GadgetObject der Wert
 "GadgetX[0]" übergeben werden, oder je nach dem, wie Sie Ihre Gadgetliste
 im Projekt-Manager genannt haben. X repräsentiert hier die Nummer des
 Fensters in der Liste im Projekt-Manager. Wichtig beim vollständigen
 Refresh ist das "[0]" hinter dem Namen der Liste. Dieser Ausdruck
 steht für das Root-Gadget, also für das erste Gadget in der Liste.

Parameter
 Window       - Name des Fenster, auf welchem das Gadget liegt.
          Üblicherweise ist das "Window1", "Window2"... .
 GadgetObject - Name des Gadgets, ab dessen Position in der Liste der Gadgets
          alle nachfolgenden Objekte refresht werden sollen.

Beispiel
 void Emperor_RethinkMenuEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */
```

```
    Emperor_RethinkLayout(Window1, Button1);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */

    Emperor_RethinkLayout(Window1, Gadget1[0]);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }

siehe auch
  Emperor_RefreshGadget()
```

1.38 Die Funktion Emperor_RefreshGadget

```
Funktion
  Makrofunktion für das Neuzeichnen eines einzelnen Gadgets.

Synopsis
  Emperor_RefreshGaget(Window, GadgetObject)

  void Emperor_RefreshGadget(struct Window *, struct Gadget *);

Aktion
  Dieses Makro nutzt die Intuition-Funktion RefreshGList().
  Beim Aufruf dieser Funktion wird das angegebene Gadget neugezeichnet.

Parameter
  Window       - Name des Fenster, auf welchem das Gadget liegt.
           Üblicherweise ist das "Window1", "Window2"... .
  GadgetObject - Name des Gadgets, das refresht werden soll.

Beispiel
  void Emperor_RefreshMenuEvent(void)
  {
    /*
    ... Abzuarbeitender Quelltext ...
    ...
    ...
    ...
    */

    Emperor_RefreshGadget(Window1, Button2);

    /*
    ...
```

```
    ...
    ...
    ... weiterführender Quelltext ...
    */
 }

siehe auch
 Emperor_RethinkLayout()
```

1.39 Die Funktion Emperor_GetGadgetAttr

```
Funktion
 Makrofunktion für das Erhalten eines Gadget-Wertes.

Synopsis
 result = Emperor_GetGadgetAttr(GadgetObject)

 STRPTR Emperor_GetGadgetAttr(struct Gadget *);

Aktion
 Dieses Makro benutzt die GetAttr()-Funktion der Intuition-Library.
 Je nachdem, welche Art Gadget abgefragt werden soll, steuert diese
       Funktion, ohne zusätzliche Parameterabfrage, die Flag-Auswahl
       eigenständig. Diese Flags (und deren zugehörige Gadgets) werden
       derzeit unterstützt:

       GA_Selected             (Button, Checkbox)
       CHOOSER_Selected
       CLICKTAB_Current
       FUELGAUGE_Level
       GRAD_CurVal
       INTEGER_Number
       LISTBROWSER_Selected
       PALETTE_Colour
       RADIOBUTTON_Selected
       SCROLLER_Top
       SLIDER_Level
       STRINGA_TextVal
       TEXTEDITOR_Contents - Flag.

 Sollen andere Flags aufgerufen werden, so nutzen Sie bitte die
       Intuition-Funktion. (GetAttr()). Die entsprechenden Werte werden  ↩
          ausgelesen und
       als Rückgabewert ausgegeben.

Parameter
 Gadget - welches Gadget ausgelesen werden soll.
    Beispiel: Checkbox3

Ergebnis
 Die Funktion gibt das o.a. Flag des entsprechenden Gadget aus.

 ACHTUNG!! Der Rückgabewert ist vom Typ "STRPTR" (Zeichenkette).
       Die Wandlung des Wertes kann mit den Funktionen:
```

```
       stringtoint()
       inttostring()

       erfolgen.

Fehler
       Funktion gibt in Version 2.00 noch einen unbrauchbaren Wert zurück.
       Dieser Fehler ist seit Version 2.20 kompensiert.

Beispiel
 void Emperor_Checkbox3GadgetUpEvent(void)
 {
   LONG number;
   STRPTR string;

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   number = stringtoint(Emperor_GetGadgetAttr(Fuelgauge2));

   strcpy(string, Emperor_GetGadgetAttr(String3));

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetGadgetAttr()
```

1.40 Die Funktion Emperor_GetGadgetAttrComplex

```
Funktion
 Makrofunktion für das Erhalten eines Gadget-Wertes.

Synopsis
 result = Emperor_GetGadgetAttrComplex(GadgetObject, GadgetattrID)

 STRPTR Emperor_GetGadgetAttrComplex(struct Gadget *, ULONG);

Aktion
 Dieses Makro benutzt die GetAttr()-Funktion der Intuition-Library.
 Im Gegensatz zu Emperor_GetGadgetAttr() muß hier
 eine konkrete Angabe eines Tags erfolgen. Dafür können jedoch alle nur
 denkbaren Werte eingesetzt werden.

 ACHTUNG !! Bitte achten Sie darauf, daß Sie die zum Gadget gehörenden Tags  ↩
    nutzen !
```

```
Parameter
 Gadget          - welches Gadget ausgelesen werden soll.
             Beispiel: Checkbox3
       GadgetattrID    - welche GadgetattrID dafür verwendet werden soll.
             Beispiel: CHECKBOX_TextPlace

Ergebnis
 Die Funktion gibt den Wert des Tags abhängig vom Gadget aus.

 ACHTUNG!! Der Rückgabewert ist vom Typ "STRPTR" (Zeichenkette).
       Die Wandlung des Wertes kann mit den Funktionen:

       stringtoint()
       inttostring()

       erfolgen.

Beispiel
 void Emperor_Checkbox3GadgetUpEvent(void)
 {
   LONG number;
   STRPTR string;

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   number = stringtoint(Emperor_GetGadgetAttrComplex(Fuelgauge2, FUELGAUGE_Min));

   strcpy(string, Emperor_GetGadgetAttrComplex(Button3, GA_Text));

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetGadgetAttrComplex()
```

1.41 Die Funktion Emperor_GetGadgetDisabledAttr

```
Funktion
 Makrofunktion für das Erhalten des Gadget_Disabled-Attributes.

Synopsis
 result = Emperor_GetGadgetDisabledAttr(GadgetObject)

 BOOL Emperor_GetGadgetDisabledAttr(struct Gadget *);
```

```
Aktion
 Dieses Makro benutzt die GetAttr()-Funktion der Intuition-Library.
 Es wird das GA_Disabled-Flag (ob das Gadget anwählbar ist)
 des Gadgets ausgelesen und als Rückgabewert ausgegeben.

Parameter
 Gadget - welches Gadget ausgelesen werden soll.
    Beispiel: Button1
        Integer3

Ergebnis
 Die Funktion gibt das GA_Disabled-Flag des entsprechenden Gadgets wieder.

Beispiel
 void Emperor_Integer1GadgetUpEvent(void)
 {
   BOOL result;
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   result = Emperor_GetGadgetDisabledAttr(Button1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetGadgetDisabledAttr()
```

1.42 Die Funktion Emperor_GetGadgetReadOnlyAttr

```
Funktion
 Makrofunktion für das Erhalten des Gadget_ReadOnly-Attributes.

Synopsis
 result = Emperor_GetGadgetReadOnlyAttr(GadgetObject)

 BOOL Emperor_GetGadgetReadOnlyAttr(struct Gadget *);

Aktion
 Dieses Makro benutzt die GetAttr()-Funktion der Intuition-Library.
 Es wird das GA_ReadOnly-Flag (ob das Gadget ist)
 des Gadgets ausgelesen und als Rückgabewert ausgegeben.

Parameter
 Gadget - welches Gadget ausgelesen werden soll.
```

```
    Beispiel: Button1
        Integer3

Ergebnis
 Die Funktion gibt das GA_ReadOnly-Flag des entsprechenden Gadgets wieder.

Beispiel
 void Emperor_Integer1GadgetUpEvent(void)
 {
   BOOL result;
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   result = Emperor_GetGadgetReadOnlyAttr(Button1);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_SetGadgetReadOnlyAttr()
```

1.43 Die Funktion Emperor_SetGadgetAttr

```
Funktion
 Makrofunktion für das Setzen eines Gadget-Wertes.

Synopsis
 Emperor_SetGadgetAttr(GadgetObject, Value)

 void Emperor_SetGadgetAttr(struct Gadget *, STRPTR);

Aktion
 Dieses Makro benutzt die SetGadgetAttrs()-Funktion der Intuition-Library.
 Je nachdem, welche Art Gadget neu beschrieben werden soll, steuert diese
       Funktion, ohne zusätzliche Parameterabfrage, die Flag-Auswahl
       eigenständig. Diese Flags (und deren zugehörige Gadgets) werden
       derzeit unterstützt:

       GA_Selected
       CHOOSER_Selected
       CLICKTAB_Current
       FUELGAUGE_Level
       GRAD_CurVal
       INTEGER_Number
       LISTBROWSER_Selected
       PALETTE_Colour
```

```
       RADIOBUTTON_Selected
       SCROLLER_Top
       SLIDER_Level
       STRINGA_TextVal
       TEXTEDITOR_Contents - Flag.

 Sollen andere Flags aufgerufen werden, so nutzen Sie bitte die
       Intuition-Funktion (SetGadgetAttrs()). Die entsprechenden Werte werden ↩
          ausgelesen und
       als Rückgabewert ausgegeben.

Parameter
 Gadget - welches Gadget ausgelesen werden soll.
    Beispiel: Checkbox3

 Value  - Wert, wie das Flag verändert werden soll.
          ACHTUNG!! Der Parameter ist vom Typ "STRPTR" (Zeichenkette).
                Die Wandlung des Wertes kann mit den Funktionen:

                stringtoint()
                inttostring()

                erfolgen.

Beispiel
 void Emperor_Checkbox3GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_SetGadgetAttr(Fuelgauge2, "23");

   Emperor_SetGadgetAttr(String3, "String");

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_GetGadgetAttr()
```

1.44 Die Funktion Emperor_SetGadgetAttrComplex

```
Funktion
 Makrofunktion für das Setzen eines Gadget-Wertes.

Synopsis
```

```
 Emperor_SetGadgetAttrComplex(GadgetObject, GadgetattrID, Value)

 void Emperor_SetGadgetAttrComplex(struct Gadget *, ULONG, STRPTR);

Aktion
 Dieses Makro benutzt die SetGadgetAttrs()-Funktion der Intuition-Library.
 Im Gegensatz zu Emperor_SetGadgetAttr() muß hier
 eine konkrete Angabe eines Tags erfolgen. Dafür können jedoch alle nur
 denkbaren Werte eingesetzt werden.

 ACHTUNG !! Bitte achten Sie darauf, daß Sie die zum Gadget gehörenden Tags ↩
    nutzen !

Parameter
 Gadget          - welches Gadget ausgelesen werden soll.
             Beispiel: Checkbox3

       GadgetattrID    - welche GadgetattrID dafür verwendet werden soll.
             Beispiel: CHECKBOX_TextPlace

 Value  - Wert, wie das Flag verändert werden soll.
          ACHTUNG!! Der Parameter ist vom Typ "STRPTR" (Zeichenkette).
                Die Wandlung des Wertes kann mit den Funktionen:

                stringtoint()
                inttostring()

                erfolgen.

Beispiel
 void Emperor_Checkbox3GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_SetGadgetAttrComplex(Fuelgauge2, FUELGAUGE_Min, "23");

   Emperor_SetGadgetAttrComplex(String3, STRINGA_MaxChars, "10");

   Emperor_SetGadgetAttrComplex(Listbrowser1, LISTBROWSER_MinVisible, "20");

   Emperor_SetGadgetAttrComplex(Listbrowser1, LISTBROWSER_Labels, (STRPTR) & ↩
      Listbrowser_NewListArray);
   /*******************************************/
   /* Dies ist kein Fehler im Guide....       */
   /* Wenn Labels geändert werden sollen, dann   */
   /* ist darauf zu achten, daß Sie das Argument */
   /* mit (STRPTR) casten und so auf diesem      */
   /* Wege die Adresse (&)!! übergeben          */
   /*******************************************/


   /*
```

```
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }

Siehe auch
  Emperor_GetGadgetAttrComplex()
```

1.45 Die Funktion Emperor_SetGadgetDisabledAttr

```
Funktion
  Makrofunktion für das Setzen des Gadget_Disabled-Attributes.

Synopsis
  Emperor_SetGadgetDisabledAttr(GadgetObject, disable/enable)

  void Emperor_SetGadgetDisabledAttr(struct Gadget *, BOOL);

Aktion
  Dieses Makro benutzt die SetGadgetAttrs()-Funktion der Intuition-Library.
  Es wird das GA_Disabled-Flag (ob das Gadget nicht anwählbar ist)
  des Gadgets neu beschrieben.

Parameter
  Gadget      - welches Gadget ausgelesen werden soll.
          Beispiel: Button1
        Integer3
  dis/enable  - Wert, ob das Flag gesetzt oder gelöscht werden soll.

Beispiel
  void Emperor_Integer1GadgetUpEvent(void)
  {
    /*
    ... Abzuarbeitender Quelltext ...
    ...
    ...
    ...
    */

    Emperor_SetGadgetDisabledAttr(Button1, TRUE);

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }

Siehe auch
  Emperor_GetGadgetDisabledAttr()
```

1.46 Die Funktion Emperor_SetGadgetReadOnlyAttr

```
Funktion
 Makrofunktion für das Setzen des Gadget_ReadOnly-Attributes.

Synopsis
 Emperor_SetGadgetReadOnlyAttr(GadgetObject, on/off)

 void Emperor_SetGadgetReadOnlyAttr(struct Gadget *, BOOL);

Aktion
 Dieses Makro benutzt die SetGadgetAttrs()-Funktion der Intuition-Library.
 Es wird das GA_ReadOnly-Flag (ob das Gadget ist)
 des Gadgets neu beschrieben.

Parameter
 Gadget - welches Gadget ausgelesen werden soll.
    Beispiel: Button1
         Integer3
 on/off - ob das Flag gesetzt oder gelöscht werden soll.

Beispiel
 void Emperor_Integer1GadgetUpEvent(void)
 {
   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   Emperor_SetGadgetReadOnlyAttr(Button1, FALSE);

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

Siehe auch
 Emperor_GetGadgetReadOnlyAttr()
```

1.47 Die Funktion stringtoint

```
Funktion
       Diese Funktion wandelt Zeichen vom Typ STRPTR (=Zeichenkette) in
       Integers (LONG = 32-Bitzahl) um.

Synopsis
 result = stringtoint(string)

 LONG stringtoint(STRPTR);
```

```
Aktion
       Diese Funktion konvertiert eine Zeichenkette vom Typ STRPTR
       (=Zeichenkette) in eine Integer (LONG = 32-Bitzahl) um.
       Dabei werden die einzelnen Zeichen der Zeichenkette der
       Reihenfolge nach ausgelesen und in die Integervariable geschrieben.

       Sie ist für die Umwandlung in Funktionen wie

       Emperor_GetGadgetAttr
       Emperor_SetGadgetAttr

       gedacht.

Parameter
 string - umzuwandelnde Zeichenkette

Beispiel
 void Emperor_Cut_MenuEvent(void)
 {
   LONG number;
   STRPTR string = "-123456789";

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   number = stringtoint(string); /* !! number = -123456789 !! */

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

siehe auch
 inttostring()
```

1.48 Die Funktion inttostring

```
Funktion
       Diese Funktion wandelt Zahlen vom Typ Integer (LONG = 32-Bitzahl)
       in Zeichenketten um.

Synopsis
 result = inttostring(number)

 STRPTR inttostring(LONG);

Aktion
```

```
       Diese Funktion konvertiert eine Zahl im Bereich von -999999999
       bis 999999999 in eine Zeichenkette (= STRPTR).
       Sie ist für die Umwandlung in Funktionen wie

       Emperor_GetGadgetAttr
       Emperor_SetGadgetAttr

       gedacht.

Parameter
 number - umzuwandelnde Zahl

Beispiel
 void Emperor_Cut_MenuEvent(void)
 {
   STRPTR string;

   /*
   ... Abzuarbeitender Quelltext ...
   ...
   ...
   ...
   */

   strcpy(string, (char *) inttostring(-123456789));

   /*
   ...
   ...
   ...
   ... weiterführender Quelltext ...
   */
 }

siehe auch
 stringtoint()
```

1.49 Die Funktion stringlength

```
Funktion
       Diese Funktion gibt die Länge einer Zeichenkette aus.

Synopsis
 result = stringlength(string)

 ULONG stringlength(STRPTR);

Parameter
 string - Zeichenkette, deren Länge bestimmt werden soll

Beispiel
 void Emperor_Cut_MenuEvent(void)
 {
         ULONG result;
```

```
    /*
    ... Abzuarbeitender Quelltext ...
    ...
    ...
    ...
    */

          result = stringlength("Hallo"); /* result = 5 */

    /*
    ...
    ...
    ...
    ... weiterführender Quelltext ...
    */
  }
```

1.50 Die Geschichte

```
2.0  - Erste Veröffentlichung

2.1  - AmigaOS 2.0 Gadgets hinzugefügt !!
       (GadTools-Gadgets sind nun verfügbar !)
     - Programm ist durch Code-Umsetzung von ReActor nach eigenem Code geschrumpft ↩
       .

2.2  - Programmabsturz, wenn Clicktab-Gadgets in einem Test-Fenster waren.
     - Geschwindigkeitsoptimierungen (z.B. beim Start, Ende etc.)
     - definierte Catalog-Erstellung
     - Makro "Preferences-Menü" für Einstellungs-Programme
     - C++-Quelltext Unterstützung
     - besserer & einfacherer generierter C-Quelltext
     - minimale Fehler behoben

2.3  - Prefs-Editor überarbeitet
     - wählbarer Prozessortyp für generierten StormC-Projekt
     - modifizierte Programmierhilfe
     - diverse Funktionsmakros überarbeitet
     - minimale Fehler behoben

3.0  - Gadget-Test Routine überarbeitet
     - Fehler beim Anzeigen von Tapedeck & Colorwheel Gadgets behoben
     - Probleme mit Clicktabs & Page Gadgets behoben (siehe "clicktabsexample")
     - Funktion "Shutdown()" hinzugefügt (Aufruf, bei Programmende)
     - Fortschrittsanzeige während des Speicherns
     - Makrofunktion Emperor_Set/GetGadgetAttrComplex() hinzugefügt

3.1  - Fehler beim Quelltextgenerieren mit ColorWheels behoben
     - Quelltext ist nun SAS/C freundlich*er*
     - Fehler beim Kopieren einer Funktion von der Programmierhilfe in den  ↩
       Texteditor behoben
     - Online Hilfe für Makrofunktionen hinzugefügt (siehe Kapitel  ↩
       Programmierhilfe)
     - ARexx-Verbindung zu StormC (Scripts - Verzeichnis)
```

```
     - Argumente beim Starten von StormC hinzugefügt (z.B. GoldED support für ↩
       StormC)
     - Ladebild hinzugefügt

3.2  - diverse Image-Gadgets (Bevel, Bitmap, Glyph) hinzugefügt
       (siehe "ClicktabExample" für Einzelheiten)
     - Fehler mit Slider und Getfile-Gadgets behoben
     - Problem mit dem Requester-Body-Text behoben ["\n" (Newline) wird nun ↩
       berücksichtigt]
     - Routine für hinzufügen, verschieben (hoch und runter) und löschen von ↩
       Objekten
       in den Listen überarbeitet

3.5  - Anzeigen des Scroller & Slider-Gadgets ist überarbeitet worden
     - Layout-Gadget (BevelStyle & -State) springt nicht mehr zu zuvor ↩
       eingestellten
       Werten zurück
     - Finnischen Katalog hinzugefügt
     - Programmumgebung zur Konfiguration von Version, Programmnamen, Stack, ↩
       Speicher
       und vielem, vielem mehr hinzugefügt
     - löschen von Fenstern ist nun möglich
     - wenn Pages hinzugefügt wurden, waren deren Generationen inkorrekt
     - GadgetHelp hinzugefügt - einfach ein Gadget verbinden, um die GadgetHelp zu
       konfigurieren ! (Programmumgebung)
     - Funktion <Menü-Bearbeiten-Liste löschen> für das Zurücksetzen der aktiven ↩
       Liste von
       Objekten hinzugefügt
     - Label's Locale-Verbindung arbeitet nun korrekt
     - GadgetHelp für beinahe jedes Gadget
     - wählbarer HookType für String-Gadgets
     - diverse Initial-konfigurationen für Texteditor-Gadgets hinzugefügt
     - verschieben von Locale-Einträgen ist nun möglich
     - editieren der Includeliste ist nun möglich
     - neues Startupbild hinzugefügt (Dank an Janne Peräaho)

3.6
     - Gradientsliders horiz/vert Ausrichtung arbeitet nun korrekt
     - einige Probleme mit globalen Variablen behoben
     - Probleme mit Node-Arrays behoben
     - spezielle Positionen/Größen für OS 2.0 Fenster arbeiten nun korrekt
     - wählbarer Smallfont für komplexe Listen (Texteditor/Programmierhilfe)
     - Fehler behoben beim verschieben vom Vorschaufenster
     - veränderbare Library-Integration für das eigene Projekt
     - Fehler behoben im Installer-Skript (arbeitet jetzt mit OS-Versionen >3.5 ↩
       zusammen)
     - Interconnection Maps hinzugefügt (Verknüpfung von Gadgets)
     - Quelltextgenerierung auf Basis von AmigaOS 2.0, AmigaOS 3.5+ oder gemischt


4.0  - GESCHAFFT ! der gesamte Quelltext ist neugeschrieben, strukturiert und ↩
   überarbeitet !!
       nur noch 4 MBytes Systemspeicher sind für den Start von Emperor notwendig !
     - der Programmfluß hat sich teilweise geändert, weil auch alle low-level ↩
       Funktionen
       neugeschrieben wurden.
     - neuer Look
```

```
     - komplexeres und professionelleres Menü
     - erweitertes GadTools-handling (mit RechtemMausButton)
```

1.51 FAQ

```
Frage:   Wenn ich die Größe von GadTools-Gadgets verändern will, ist die Umrandung ↩
     für das
         aktivierte Gadget nicht oder kaum noch zu erkennen.

Antwort: Höchstwahrscheinlich nutzen Sie ein Programm wie VisualPrefs, welches die ↩
     Darstellung
         von GadTools-Gadgets verändert. Wenn Sie es deaktivieren, sollte es  ↩
            wieder funktionieren.
         Sie können aber auch einfach, die Ränder der "String"- und "Integer"- ↩
            Gadgets wieder
         auf Normalgröße bringen, was denselben Effekt hat.


Frage:   Emperor startet auf meinem System nicht !

Antwort: Es müssen diverse Dinge gewährleistet sein:
         * AmigaOS 3.5 installiert ?
         * vom Systemspeicher müssen mehr als 4 MBytes frei sein
         * Assign auf <PFAD>:Emperor
         * RexxMast gestartet ?
         * Stack auf mehr als 50000 Bytes eingestellt ?
           - unter CLI:  'stack 50000' eintippen
           - unter WB:   in den Piktogramminformationen einen Stack von >50000  ↩
              Bytes einstellen
           - unter DOpus:in Menü 'Einstellungen-Umgebung' den Punkt CLI-Start  ↩
              anwählen und
                        Stack ändern !
         Eigentlich sollte zu jedem dieser Fehler aber ein Fehlerfenster  ↩
            erscheinen
```

1.52 known Bugs & future Plans

```
bekannte Fehler des Programms:

       1. Drucken der Objekttabelle wurde deaktiviert, weil die Zusammenstellung
          im Speicher derart aufwendig war, daß ich ZUNÄCHST darauf verzichtet  ↩
             habe.
          Dieser Bug sollte aber keinen Einfluß auf die Funktionstüchtigkeit des  ↩
             Programms
          haben.

       2. Das Attributfenster "Datebrowser" kann schriftartbedingt viel zu breit  ↩
          sein !
          (nur warum .... ?)

       3. Clips speichern funktioniert noch nicht richtig. (deaktiviert)
```

4. Text bzw. Zeilenformatierung funktioniert nicht.

5. Labels und Bitmaps werden im Vorschaufenster nicht angezeigt.

zukünftige Pläne:

1. Speedbar-Buttons sollen konfigurierbar sein.

2. Screen-Kategorie fehlt noch.

3. Undo/Redo der Listen soll verfügbar sein.

4. Überprüfen der Listen soll verfügbar sein.

5. Insertmode unterstützt das ReAction-Texteditor-Gadget selbst noch nicht.

6. Im Projekt-Verzeichnis sollen dereinst Verzeichnisse für die einzelnen Projekte
   erstellt werden.

1.53 Der Autor

Falls es irgendwelche Verbesserungsvorschläge gibt oder sich Schwierigkeiten
mit dem Programm ergeben sollten, bitte an mich wenden.

Die Adresse des Autors:

Matthias Gietzelt
Ringstraße 41
19069 Hundorf
Germany

EMail:
shamane@exmail.de

1.54 Danksagungen

Ich möchte mich sehr bei Janne Peräaho für sein hübsches Startupbild bedanken.
Ebenso für seine Fehlerberichte um existente Fehler in Emperor zu beseitigen.
- Danke Janne !

Ebenfalls bedanken möchte ich mich bei Olivier Martin, der mich auf diverse Fehler
hingewiesen und ein Beispielprojekt mit Emperor kreiert hat. Das Beispielprojekt
trägt den Namen "automate" und ist im "Project"-Verzeichnis Emperors zu finden.
- Danke Olivier !

Ich möchte mich auch bei allen anderen 'Fehlermeldern' für die gründliche Benutzung

Emperors bedanken !!