Programmierung_Deutsch

COLLABORATORS

	TITLE : Programmierung_Deutsch	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 24, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Programmierung_Deutsch

1.1 Eagleplayer 2.00 Developer-Dokumentation

```
                    Developer-Anleitung
                            zum
                        Eagleplayer



                     Version  2.00
                Anleitung $VER: V2.00 (30.Juni 96)
              © 1993-96 Defect Software Productions


  Inhaltsverzeichnis:


    Einleitung
    Programmierung der Player
    Programmierung der Engines
          Einbindung des Eagleplayers in andere Programme
    Danksagungen
```

1.2 Danksagungen

```
Danke an alle !
```

1.3 Die Player

```
 Die Programmierung der externen Player:
   Einleitung zu den externer Playern
    Das externe Player Konzept
    Schematischer Aufbau von externen Playern
    Schematischer Aufbau von Custom Modulen
   Anpassung von Playern
```

1.4 einleitung zu den externer playern

```
 Die Programierung externer Player
-----------------------------------
```

Der Eagleplayer unterstützt sog. externe Player. Das sind Executables, die in einem speziellen Format vorliegen und vom EaglePlayer nachgeladen werden können.

Es handelt sich bei den Kapiteln 6.1-6.5 um die originale Anleitung von Delirium (leicht verändert). Auf die Besonderheiten des Eagleplayers wird in Kapitel 6.6 eingegangen.

1.5 das externe player konzept

```
 6.1.1 Das externe Player Konzept
----------------------------------
```

Externe Player sind Executables (Objekt Files), die den Replaycode eines Soundsystems enthalten. Am Anfang besitzen sie eine charakteristische Playerstruktur. Diese Struktur wird mit dem Macro aus dem Includefile 'misc/deliplayer.i' erzeugt. Es gibt zwei Arten von externen Playern: normale Player und Custom Module.

1.6 schematischer aufbau von externen playern

```
 6.1.2 Schematischer Aufbau von externen Playern
-------------------------------------------------
```

Normale Player unterscheiden sich von Custom Modulen durch das Vorhandensein einer Check Routine und das Fehlen des DTP_CustomPlayer Tags. Desweiteren enthalten sie natürlich keine Musikdaten :-)

```
{ Playerheader  } Kennzeichnet das Object als Player.
{ TagArray  } Beschreibung, was der Player kann.
{ Interfacecode } Playername/Registerumsetzung/Checkcode u.ä.
```

```
  { Replaycode  } Eigentlicher Musikabspielcode.
  { evtl. Daten } und Daten
```

1.7 schematischer aufbau von custom modulen

```
 6.1.3 Schematischer Aufbau von Custom Modulen
-----------------------------------------------
```

Custom Module sind keine Module im herkömmlichen Sinn, sondern im wesentlichen Player, die ein Spezialmodul beinhalten. Mit diesem Format können Sie auch die ausgefallensten Moduleformate leicht an den EaglePlayer anpassen. Sollten für den Player jedoch mehrere Module existieren, ist es in jedem Fall ratsam, einen richtigen Player zu schreiben.

```
  { Playerheader  } Kennzeichnet das Object als Player.
  { TagArray  } Beschreibung, was der Player kann.
  { Interfacecode } Routinen zur Registerumsetzung u.ä.
  { Replaycode  } Eigentlicher Musikabspielcode.
  { evtl. Daten } und Daten
  { SOUND DATEN } Das eigentliche Modul
```

1.8 anpassung von playern

```
6.2 Anpassung von Playern
--------------------------
```

Es ist relativ einfach, eine Replayroutine an den EaglePlayer anzupassen. Alles was Sie tun müssen, ist ein wenig Interfacecode zu schreiben. Dies ist aber halb so wild, denn der EaglePlayer stellt ihnen viele hilfreiche Routinen zur Verfügung. Um ein neues Soundsystem anzupassen, benötigen Sie zum einen den Quellcode oder ein linkbares Objektfile der Replayroutine, zum anderen sollten mindestens ca. 5 Module zum Testen der Funktionstüchtigkeit des Players vorhanden sein.

1.9 playerheader

```
 6.2.1 Playerheader
--------------------
```

Das PLAYERHEADER Macro generiert den Header, der das File als externen Player identifiziert. Dieses Macro muß angegeben werden und ganz am Anfang des Players stehen. Als einziger Parameter ist ein Pointer auf ein Tag Array einzutragen, in dem alle Funktionen stehen, die der Player dem EaglePlayer zur Verfügung stellt. Bei allen Funktionsaufrufen außer der Interruptroutine (DTP_Interrupt) enthält a5 die Base. Globale Variablen können dadurch adressiert werden (siehe 'Misc/Deliplayer.i & 'Misc/Eagleplayer.i').
Da der Eagleplayer vor Aufruf einer Playerroutine (außer bei

```
    DTP_Interrupt) alle Register sichert, dürfen diese verändert werden.

    PLAYERHEADER <Tagarray>

  Tagarray  Pointer auf ein Tag Array, das mit TAG_DONE
      abgeschlossen sein muß. Tags, bei denen Ti_Data NULL
      ist, werden ignoriert.
```

1.10 modulerkennung

```
 6.2.2 Modulerkennung
----------------------
```

Damit EaglePlayer die einzelnen Module unterscheiden kann, befindet sich in jedem Player eine Erkennungsroutine, die auf ein zugehöriges Modul testet. Diese Routine prüft auf Stellen im Modul, die bei allen Modulen eines Players gleich sind. So z.B. auf 'M.K.' bei Offset $438 im ProTracker-Player. Bei Sound- und NoiseTracker ist nur das Modul abgespeichert, z.B. bei MarkII hängt vor jedem Modul noch die Replay-routine. Bei so einem Modul müssen Sie auf signifikante Assembler-befehle testen. Ein Vergleich auf ein oder zwei Sprungbefehle, egal ob bra oder jmp, ist hier nicht ausreichend, da bei vielen Modulen dieses Typs Sprungtabellen am Anfang stehen und der Player möglicher-weise das falsche Modul erkennen könnte, was in den meisten Fällen zum Absturz führt. Der Player muß genau eine Checkroutine verwenden. Daraus ergeben sich zwei Grundtypen von Playern.

a) Typ eins Player - in der Regel etwas komplexer
Bei ihnen ist die Check1 Funktion implementiert.

Vorteil: Es lassen sich auch Player realisieren, die das Modul selbst nachladen.
Nachteil: Gepackte Files werden nicht unterstützt.

Dieser Player sollte nur für Härtefälle verwendet werden.
(z.B: IFF-8SVX Player, der das Sample während dem Spielen nachlädt, ...)

b) Typ zwei Player - die einfachere Variante
Bei ihnen ist die Check2 Funktion implementiert

Vorteil: Das Modul kann gepackt sein, der Player bemerkt davon nichts.
Nachteil: Das Modul ist in jedem Fall im Chip-Memory.

Es sollte im Normalfall dieser Playertyp verwendet werden.

Egal ob Typ eins oder zwei, wenn der Player das Modul erkannt hat, muß er in d0.l=0 zurückliefern, wenn nicht d0.l<>0.

1.11 interrupts

```
 6.2.3 Interrupts
------------------

    Hier gibt es auch eine Einteilung in zwei verschiedene Typen.

    a) Player, die den Eagleplayer Interrupt verwenden
    Vorteil:  Der Player ist vom Videomodus unabhängig.
        Besitzt automatisch die Faster/Slower/Speedregler Funktion.
        Kein Aufwand für den Interrupt (Code + Interruptstruktur).
        Ist kompatibel zum serial.device
    Nachteil: Der Interrupt kommt nicht synchron zum VBlank. (Dies führt
        nur in den seltensten Fällen zu Problemen.)

    b) Player die ihren eigenen Interrupt erzeugen.
    Vorteil:  Es können andere Interruptquellen benutzt werden
        (z.B. AudioIRQ).
    Nachteil: Erhöhter Aufwand, bei VBlank nicht mehr unabhängig vom
        Videomodus.

    Wenn ein eigener Timerinterrupt verwendet wird, sollte die CIAB
    (wg. OS2.0) und die entsprechenden Resourcefunktionen verwendet werden.
    Außerdem ist es sehr sinnvoll die Replayroutine nicht direkt im CIA-B
    TimerInterrupt abspielen zu lassen, sondern im Timerinterrupt einen
    Soft-Interrupt mittels Cause() zu generieren. In diesem SoftInt kann
    man dann die eigentliche Replayroutine ablaufen lassen. So hat man den
    Vorteil, daß man wesentlich kompatibler zum serial.device ist. Dies
    liegt daran, daß SoftInt eine niedrigere Priorität als der der RBF
    (Read Buffer Full) Interrupt hat, d.h. erst wird der serielle Port
    bedient, dann erst die Replayroutine. Es wird davor gewarnt, direkt in
    die Interruptvektoren zu schreiben! Zur Erinnerung: vom Betriebssystem
    werden die Funktionen AddIntServer() und SetIntVector() zur Verfügung
    gestellt!
```

1.12 bedeutung der tags

```
 6.3 Bedeutung der Tags
------------------------

  Außer den SystemTags (TAG_DONE, TAG_IGNORE, TAG_MORE, TAG_SKIP) dürfen
  folgende Tags verwendet werden:

  DTP_CustomPlayer (BOOL) - dieser Tag deklariert einen Player als
    Customplayer.
    Bei Verwendung dieses Tags sind folgende Tags dann
    bedeutungslos:  DTP_PlayerVersion
        DTP_Check1
        DTP_Check2
        DTP_Config
        DTP_UserConfig

  DTP_RequestDTVersion (WORD) - damit kann man sicherstellen, daß
    mindestens eine bestimmte Version von Eagleplayer
    vorhanden ist. Dieser Tag muß angegeben werden, wenn
```

bei den EagleplayerGlobals neue Funktionspointer
hinzugekommen sind und diese vom Player benutzt werden.
ti_Data ist dabei die Playerrevision.

DTP_RequestV37 (BOOL) - wenn dieser Tag vorhanden ist, wird der Player
nur noch von der Kick 2.0 Version von Eagleplayer geladen
(d.h. dtg_GadToolsBase ist gültig).

DTP_PlayerVersion (WORD) - Tag, der die Revisionsnummer des Players
enthält. Bei zwei Playern mit dem gleichen Playernamen wird
derjenige mit der größeren Revisionsnummer geladen.

DTP_PlayerName (STRPTR) - ti_Data enthält den Pointer auf den Namen des
Players. Dieser String kann zwar beliebig lang sein aber
zur Zeit werden nur die ersten 24 Zeichen angezeigt. Dieser
Tag muß existieren!

DTP_Creator (STRPTR) - Pointer auf den Namen des Autors. Dieser wird im
Setupfenster im Playerinfofeld angezeigt. Dieser String
kann $A als Zeilenumbruch enthalten.

DTP_Check1 (FPTR) - Pointer auf eine Modulerkennungsroutine, die
aufgerufen wird, wenn 1024 Bytes des Moduls geladen sind.
Wird das Modul erkannt, liefert sie d0=0, ansonsten d0<>0.

DTP_Check2 (FPTR) - Pointer auf eine Modulerkennungsroutine, die
aufgerufen wird, wenn das komplette Modul geladen und evtl.
entpackt ist. Wird das Modul erkannt, liefert sie d0=0,
ansonsten d0<>0.

DTP_ExtLoad (FPTR) - Pointer auf eine optionale Laderoutine für Module.
Ist kein Fehler aufgetreten, wird d0=0 zurückgegeben, sonst
d0<>0. Hinweis: Achten Sie darauf, daß im Fehlerfall alle
allocierten Ressourcen (Memory, Locks, ...) wieder
freigegeben werden, da dann keine weiteren Playerfunktionen
mehr angesprungen werden.

DTP_Interrupt (FPTR) - Pointer auf eine Interruptroutine, die mittels
eines Timerinterrupts standardmäßig alle 1/50 sec
aufgerufen wird. Dies ist die Standardmethode, um mit der
richtigen Abspielgeschwindigkeit im PAL/NTSC/Productivity
Videomodus zu spielen. Wenn keine DTP_Faster/DTP_Slower
Tags angegeben sind, übernimmt Eagleplayer dies durch
Verändern der Interruptfrequenz. Dieser Tag kann auch nicht
existieren, dann müssen aber DTP_StartInt/DTP_StopInt
vorhanden sein !

DTP_Stop (FPTR) - Pointer auf eine optionale Stoproutine. Wenn dieser Tag
nicht vorhanden ist, verfährt Eagleplayer folgendermaßen:
  Interrupt stoppen (DTP_StopInt)
  Sound Cleanup (DTP_EndSnd)
  Song initialisieren (DTP_InitSnd)
Ansonsten hat diese Routine die Aufgabe, einen evtl.
spielenden Song anzuhalten und so zu initialisieren, daß
dieser beim nächsten Starten des Interrupts von Anfang an
zu spielen beginnt.

DTP_Config (FPTR) - Pointer auf eine optionale Initialisierungsroutine.
Diese wird nur einmal unmittelbar nach dem Laden des
Players aufgerufen. Mögliche Anwendungen: Laden einer
playerspezifischen Konfigurationsdatei.

DTP_UserConfig (FPTR) - Pointer auf eine optionale Initialisierungs-
routine. Diese wird nur dann aufgerufen, wenn der User den
Player im Setupfenster anwählt und das 'Config' Gadget
drückt. Mögliche Anwendungen: Öffnen eines Fensters zum
Setzen weiterer Optionen wie z.B. Instrumentenpfad und
Abspeichern einer playerspezifischen Konfigurationsdatei.

DTP_SubSongRange (FPTR) - Dieser Tag sollte angegeben werden, wenn der
Player MultiModule unterstützt. ti_Data zeigt dabei auf
eine Routine, die als Returnwert in d0 die minimale und
in d1 die maximale Subsongnummer zurückgeben muß.
Hinweis: Wenn möglich sollte dieser Tag (evtl. zusammen
mit DTP_SubSongTest) anstelle von DTP_NextSong/DTP_PrevSong
verwendet werden.

DTP_InitPlayer (FPTR) - Pointer auf eine optionale Initialsierungs-
routine, die aufgerufen wird, wenn ein Modul erfolgreich
geladen wurde. Tritt kein Fehler auf, liefert sie d0=0,
ansonsten d0<>0. Hier muß die Allocation der Audiokanäle
stattfinden! (Eagleplayer stellt dafür eine eigene Routine
zur Verfügung) Falls der Player Multi-Module unterstützt,
muß hier dtg_SndNum(a5) auf die erste Subsongnummer gesetzt
werden. Falls eine Routine für DTP_SubSongRange existiert,
macht Eagleplayer das automatisch (d.h. die Initialisierung
von dtg_SndNum(a5) kann weggelassen werden).

DTP_EndPlayer (FPTR) - Pointer auf eine optionale Cleanuproutine, die
aufgerufen wird, wenn das Modul aus dem Speicher entfernt
wird. Hier muß die Freigabe der Audiokanäle stattfinden!
(Eagleplayer stellt dafür eine eigene Routine zur
Verfügung)

DTP_InitSound (FPTR) - Pointer auf eine optionale Initialsierungsroutine.
Diese muß das Modul initialisieren, so daß beim Starten des
Interrupts der Song von Anfang an zu spielen beginnt.

DTP_EndSound (FPTR) - Pointer auf eine optionale Cleanuproutine. Diese
kann z.B. die Lautstärkeregister und die Audio-DMA
rücksetzen.

DTP_StartInt (FPTR) - Pointer auf eine Initialsierungsroutine, die
existieren muß, wenn DTP_Interrupt nicht vorhanden ist.
In diesem Fall muß hier der Sound gestartet werden.

DTP_StopInt (FPTR) - Pointer auf eine Cleanuproutine, die existieren muß,
wenn DTP_Interrupt nicht vorhanden ist. In diesem Fall muß
hier der Sound gestoppt werden.

DTP_Volume (FPTR) - Pointer auf eine Funktion, welche die Lautstärke neu
setzt. Die Funktion wird aufgerufen, wenn die Volume neu
gesetzt wird (Slider, ARexx) und beim Initialisieren des
Moduls vor DTP_InitSnd. Die Mastervolume steht in

```
    dtg_SndVol(a5). Die Mastervolume ist dabei der maximale
    Volumewert. Die effektive Volume errechnet sich also
    durch: VOL_eff = (( dtg_Volume(a5) * modulevolume )>>6)
    Näheres siehe Beispielsourcen.

  DTP_Balance (FPTR) - Pointer auf eine Funktion, welche die Balance neu
    setzt. Die Funktion wird aufgerufen, wenn die Balance neu
    gesetzt wird (Slider, ARexx) und beim Initialisieren des
    Moduls vor DTP_InitSnd. Die Balance für die linken Kanäle
    steht in dtg_SndLBal(a5), für die rechten Kanäle in
    dtg_SndRBal(a5). Hinweis: Alle Player die Balance unterstützen
    können auch Volume! Man verwendet dann die gleiche Routine
    zum Setzen der Volume&Balance. Die linke Volume errechnet
    sich wie folgt: (( dtg_Volume(a5) * dtg_SndLBal(a5) )>>6)
    Entsprechendes gilt für rechts.

  DTP_Faster (FPTR) - Pointer auf eine Funktion, die den Abspielvorgang
    beschleunigt.

  DTP_Slower (FPTR) - Pointer auf eine Funktion, die den Abspielvorgang
    verlangsamt.

  DTP_NextPatt (FPTR) - Pointer auf eine Funktion, die den Patternzeiger
    um eins erhöht.

  DTP_PrevPatt (FPTR) - Pointer auf eine Funktion, die den Patternzeiger
    um eins erniedrigt.

  DTP_NextSong (FPTR) - Pointer auf eine Funktion, die Subsongnummer auf
    den nächsten Subsong setzt und diesen spielt.
    (Falls vorhanden)

  DTP_PrevSong (FPTR) - Pointer auf eine Funktion, die Subsongnummer auf
    den vorhergehenden Subsong setzt und diesen spielt.
    (Falls vorhanden)

  DTP_SubSongTest (FPTR) - (ab Version 1.35) Dieser Tag wird nur ausgewertet,
    wenn schon der Tag DTP_SubSongRange angegeben wurde. ti_Data
    zeigt dabei auf eine Routine, die als Returnwert in d0 einen
    Boolschen Wert zurückliefert. Dieser gibt an, ob der SubSong
    mit Nummer dtg_SndNum(a5) gültig ist (d0=0) oder nicht
    (d0<>0). Dieser Tag ist nur bei den Playern nötig, bei denen
    nicht jeder SubSong (innerhalb der durch DTP_SubSongRange
    festgelegten Grenzen) benutzt ist.
```

1.13 eagleplayer support funktionen

```
 6.4 Eagleplayer support Funktionen
-----------------------------------

  Eagleplayer stellt zur Erleichterung der Playeranpassung einige
  Funktionen zur Verfügung. Eine Funktion wird wie folgt aufgerufen:

  move.l  dtg_XXX(a5),a0
  jsr (a0)
```

```
Alle folgenden Funktionen außer dtg_SongEnd und dtg_SetTimer verwenden
d0/d1/a0/a1 als Scratchregister. In a5 muß bei allen Aufrufen (außer bei
dtg_SongEnd und dtg_SetTimer) die Base stehen. Derzeit existieren
folgende Funktionen:

dtg_GetListData

SYNOPSIS
  memory size = dtg_GetListData(number)
  a0     d0          d0.l

FUNCTION
  Liefert Adresse und Länge eines mit LoadFile() geladenen
  Files.

INPUTS
  number - Nummer eines Files beginnend mit 0 für das vom
     User selektierte File.

RESULT
  memory - ein Pointer auf die Startadresse des Files im
     Speicher oder NULL im Fehlerfall.
  size - Länge des Files in Bytes bzw. 0 im Fehlerfall.


dtg_LoadFile

SYNOPSIS
  success = dtg_LoadFile(name)

FUNCTION
  Lädt und entpackt ggf. das angegebene File ins Chip-
  Memory. (Hinweis: diese Funktion ergänzt automatisch,
  falls das File mit dem angegebenen Namen nicht geöffnet
  werden konnte '.pp','.im' und '.xpk')

INPUTS
  name - der Filename steht in einem internen Buffer (seine
         Adresse steht in dtg_PathArray)

RESULT
  success - alles ok d0.l=0, sonst d0.l<>0.


dtg_CopyDir

SYNOPSIS
  dtg_CopyDir()

FUNCTION
  Kopiert das Directory des von User angewählten Files an das
  Ende des Strings, auf den dtg_PathArray(a5) zeigt.


dtg_CopyFile
```

```
SYNOPSIS
  dtg_CopyFile()

FUNCTION
  Kopiert den Filenamen des von User angewählten Files an das
  Ende des Strings, auf den dtg_PathArray(a5) zeigt.


dtg_CopyString

SYNOPSIS
  dtg_CopyString(string)
           a0

FUNCTION
  Kopiert den String, auf den das Register a0 zeigt, an das
  Ende des Strings, auf den dtg_PathArray(a5) zeigt.

INPUTS
  string - der Pointer auf den anzuhängenden String steht
     in a0


dtg_AudioAlloc

SYNOPSIS
  success = dtg_AudioAlloc()

FUNCTION
  Belegt alle Audiokanäle.

RESULT
  success - alles ok d0.l=0, sonst d0.l<>0.


dtg_AudioFree

SYNOPSIS
  dtg_AudioFree()

FUNCTION
  Gibt die mit dtg_AudioAlloc belegten Audiokanäle wieder
  frei.


dtg_StartInt

SYNOPSIS
  dtg_StartInt()

FUNCTION
  Startet den Soundinterrupt. (Falls er nicht schon läuft.)
  Falls DTP_Interrupt existiert, startet Eagleplayer einen
  Timerinterrupt, ansonsten wird DTP_StartInt aufgerufen.


dtg_StopInt
```

```
  SYNOPSIS
    dtg_StopInt()

  FUNCTION
    Stoppt den Soundinterrupt. (Falls er nicht schon angehalten
    ist.) Falls DTP_Interrupt existiert, stoppt Eagleplayer
    seinen Timerinterrupt, ansonsten wird DTP_StopInt
    aufgerufen.


  dtg_SongEnd

  SYNOPSIS
    dtg_SongEnd()

  FUNCTION
    Signalisiert Eagleplayer, daß das Modul einmal komplett
    gespielt wurde. Diese Funktion verändert keine Register
    und darf auch von Interrupts aufgerufen werden.


  dtg_CutSuffix

  SYNOPSIS
    dtg_CutSuffix()

  FUNCTION
    Entfernt am Ende des Strings, auf den dtg_PathArray(a5)
    zeigt ggf. die Endung '.pp', '.im' oder '.xpk'


  dtg_SetTimer

  SYNOPSIS
    dtg_SetTimer()

  FUNCTION
    Programmiert den CIA-Timer mit dem Wert, der sich in
    dtg_Timer(a5) befindet. Diese Funktion verändert keine
    Register und darf auch von Interrupts aufgerufen werden.
```

1.14 tips zur anpassung

```
 6.5 Tips zur Anpassung
------------------------

  In der Checkroutine und in den Playern überhaupt darf nur auf Speicher
  zugegriffen werden, der auch im Bereich des geladenen Moduls liegt !!
  Wir haben eine riesige Menge an Enforcer-Hits bemerkt, die im wesentlichen
  auf illegale Speicherzugriffe durch die Checkroutinen zurückzuführen sind,
  wenn das Modul zu kurz ist.

  Wenn die Möglichkeit besteht, sollte der neue Player unbedingt mit Enforcer
```

```
  getestet werden. Dabei sollten alle anderen externen Player gelöscht werden.

  Diese kleine Checkliste für sollte für die Player/Custommodul Anpassung
  erfüllt werden, damit der Player systemkonform ist und einwandfrei
  funktioniert:

  [ ] ist die Checkroutine präzise genug ?
  [ ] werden die Audiokanäle richtig belegt und freigegeben ?
  [ ] wird aller allocierter Speicher freigegeben ?
  [ ] werden alle Locks wieder freigegeben ?
  [ ] enforcer und mungwall - Test bestanden ?
  [ ] werden alle denkbaren Fehler korrekt abgefangen ?
  [ ] wurde der Player mit 1.3, 2.0 und 3.0 getestet ?
  [ ] spielt der Player in allen Videomodi korrekt ?
  [ ] greift der Player nur auf zugewiesenen Speicher zu
  [ ] werden korrekte Fehlermeldungen zurückgegeben
  [ ] Werden überall mögliche Enforcer-Hits abgefangen ?
  [ ] Arbeitet die Checkroutine bei fremdem Module sicher ?
      Enforcergefahr !!!
  [ ] Wird bei GetInfos nur die Adr der Tagliste übergeben ?


  Problemecke:
---------------

  Symptom mögliche Ursache      Beseitigung
-------------------------------------------------------------------------------

  Absturz a5 überschrieben
    Register nicht gesichert
    falsche oder zu späte Initialisierung
    Modul zu kurz
    Modul zu neu für den Player   präziserer Check

  klingt schräg Audiodaten nicht im Chipmem   Player ins Chipmem
    falsche Initialisierung
    Modul zu neu für den Player   präziserer Check
    Player benötigt bestimmte Werte in  zusätzliche
    einigen Registern     Initialisierung
    unverträglicher Videomodus

  kein Ton  Audio-DMA abgeschaltet
   bei >68000 Player schreibt direkt auf    Betriebssystem
    Interruptvektoren     benutzen

  fast keine  fehlerhaftes Interrupt-Handling   bei VBlank-IRQ:
  freie CPU-Zeit          Z-Flag setzen
```

1.15 die besonderheiten und neuerungen des eagleplayers

```
 6.6 Die Besonderheiten und Neuerungen des Eagleplayers
-------------------------------------------------------

Der Eagleplayer bietet eine Fülle neuer Möglichkeiten, die im folgenden
```

erläutert werden. Folgende Tags sind für Custom-Module im Eagleplayer wieder relevant, z.B. falls es irgendjemand packen sollte, Jason Page-Module als Custom zu erschaffen:

```
DTP_PlayerName
DTP_Creator
DTP_ExtLoad
```

Zu den besonderen Neuerungen gehören vor allem die neuen Eagleplayer-Tags, die ModuleInfo-Funktion, Analyzerunterstützung sowie neuen Globals.

1.16 die neuen tags

```
 6.6.1 Die neuen Tags
----------------------
```

EP_Get_ModuleInfo (FPTR) - Diese Funktion hat in A0 den Zeiger auf eine fertig initialisierte Module-Info Tagliste zurückzugeben oder Null für keine TagListe. Weitere Informationen finden Sie im Kapitel 6.6.2 Moduleinfo

EP_Free_ModuleInfo (FPTR) - Diese optionale Funktion kann zum Beispiel dazu dienen, Speicher, der für die Moduleinfotagliste allokiert wurde, freizugeben. Es werden keinerlei Returnwerte erwartet.

EP_Voices (FPTR) - Über diese Funktion werden die Werte für die einzelnen Stimmen in den Bits 0-3 des Datenregisters D0 übergeben. Ist das jeweilige Bit gesetzt, so ist die entsprechende Stimme eingeschaltet. Bit 0 entspricht Kanal 0 , Bit 1 = Kanal 1 usw.

EP_Structinit (FPTR) - übergibt im Adressregister A0 einen Pointer auf eine UPS_USER - Struktur, in der der jeweilige Player die Informationen für die Analyzerprogramme (Engines) ablegt oder Null. Diese Strukturadresse wird dann vom Eagleplayer an alle Engines weitergegeben, die sie dann auswerten.

EP_StructEnd (FPTR) - Optional,die Struktur wird freigegeben, der Speicher,der eventuell allokiert wurde, darf freigegeben werden.

EP_LoadPlConfig (FPTR) - ruft Routine auf, die die Konfiguration des jeweiligen Players lädt (vorzugsweise aus 'ENV:Eagleplayer/x.config')

EP_SavePlConfig (FPTR) - Routine zum Sichern der Playerkonfiguration nach 'Envarc:Eagleplayer/x.config' (Vorschlag der Autoren)

EP_GetPositionNr (FPTR) - gibt in D0.l die aktuelle Patternnummer zurück, die gerade gespielt wird, benötigt für Patternumschaltung (dabei Scrollt der Eagleplayer die jeweilige Position ins Fenster

EP_SetSpeed (FPTR) - Für Player mit eigenem Timer wird hier die Möglichkeit gegeben, den Speedregler zu nutzen. Dabei wird in D0 ein Wert von -25 bis +25 übergeben, der den aktuellen Stand des Reglers repräsentiert. 0 Stellt dabei den Standardwert dar.

EP_Flags (LONG) - Die Flags geben an, welche Funktionen der Player grundsätzlich unterstützt. Dieser Tag wurde eingeführt, seitdem wir "modifizierende" Player erschufen, die versuchen, Routinen in Modulen mit Playroutine darinnen so zu modifizieren, daß z.B. Analyzerunterstützung möglich ist. Da sich solche Replays voneinander unterscheiden, kann es sein, daß die eine oder andere Funktion von Modul zu Modul nicht möglich ist. Daher gibt es folgende Flagbits:

EPF_Songend - Der Player unterstützt Songend

```
    EPF_Restart   - Modul läßt sich mehrfach abspielen (teilweise
        nicht möglich !)
    EPF_Disable   - Player ist nicht erlaubt
    EPF_NextSong  - kann nächsten Untersong anwählen
    EPF_PrevSong  - kann vorherigen Untersong anwählen
    EPF_NextPatt  - kann nächstes Pattern anwählen
    EPF_PrevPatt  - kann vorheriges Pattern anwählen
    EPF_Volume    - Lautstärkeregelung möglich
    EPF_Balance   - Balance möglich
    EPF_Voices    - Stimmenbeeinflussung möglich
    EPF_Save      - Modul kann gesichert werden
    EPF_Analyzer  - Analyzerunterstützung
    EPF_ModuleInfo- Informationen über Modul erhältlich
    EPF_SampleInfo- Informationen über verwendete Sample erhältlich
    EPF_Packable  - Module darf gepackt werden
    EPF_VolVoices - Die Lautstärken der Stimmen können unter
        schiedlich sein (EPG_VolVoice1...)

    Dieser Tag dient vor allem Informationszwecken im PlayerWindow.


  EP_KickVersion (WORD)  - Mindeste Kickstartversion (37 für Kick 2.0)
  EP_PlayerVersion (LONG)- Mindeste EaglePlayer-Version
    Wenn der Player nur vom EaglePlayer geladen werden soll, kann
    bei DTP_RequestDTVersion eine -1 eingetragen werden.
  EP_EjectPlayer (FPTR)  - Falls der Player irgendwelchen Speicher, Filelocks
    usw. besorgt, kann der an dieser Stelle freigegeben werden,
    danach wird der Player durch den Eagleplayer entfernt.

  EP_Date (LONG) Datum, an dem der Replayer geschireben wurde.

  EP_Check3 (FPTR) - Checktroutine, die nach dem Laden der ersten 1000 Byte
         aufgerufen wird. Sollte das Module erkannt werden, wird
         es ins Fastram geladen.
```

1.17 moduleinfo

```
 6.6.2 Moduleinfo
------------------

FÜr die Moduleinfofunktion stellt der Eagleplayer eine Reihe von Tags zur
Verfügung, die Auskunft über das aktuelle Modul ermöglichen. Die Tagliste
wird bei Aufruf von "EP_Moduleinfo" (siehe auch dort im Kapitel 6.6.1).
in A0 übergeben. Bitte beachten Sie, daß bisher noch nicht alle Informationen
auch angezeigt werden. In der registrierten Version wird es ein Window geben,
in dem hoffentlich alle Informationen ausgewertet werden.

Ab Eagleplayer V1.50 gibt es einen Tag, der gleich auf die Tagliste der
ModuleInfotags zeigt.

MI_SongName (STRPTR) - Songname, der mitunter im Modul zu finden ist. Wird eine
    Null in TI_Data übergeben, so erscheint bei Moduleinfo ein
    "Unknown" als Songname. Sehr komfortable Möglichkeit, den
```

```
    richtigen Namen gerippter Module zu erhalten.

MI_AuthorName (STRPTR) - Name dessen, der den Song schrieb, bei Rückgabe von
     Null in TI_Data gibt der Eagleplayer ein "Unknown" aus.

MI_SubSongs (LONG) - Anzahl der Untersongs im Modul

MI_Pattern (LONG) - Anzahl der Patterns im Modul

MI_MaxPattern (LONG) - Maximale Anzahl der Patterns (z.B. Soundtracker: 64)

MI_Length (LONG) - Länge des Songs (z.B. in Patterns)

MI_MaxLength (LONG) - Maximale Länge des Songs (z.B. Soundtracker 127)

MI_Steps (LONG) - Anzahl der Steps (BP Soundmon)

MI_MaxSteps (LONG) - Max. Anzahl der Steps

MI_Samples (LONG) - Anzahl der benutzten Samples

MI_MaxSamples (LONG) - Max. Anzahl der Samples (z.B. Protracker: 31)

MI_SynthSamples (LONG) - Anzahl der benutzten synthetischen Samples

MI_MaxSynthSamples (LONG) - Maximale Anzahl der synthetischen Samples

MI_Songsize (LONG) - Größe des Songs in Bytes

MI_SamplesSize (LONG) - Länge der Samples in Bytes

MI_ChipSize (LONG) - benutzter Chip-Speicher in Bytes

MI_OtherSize (LONG) - benutzter Fast-Speicher in Bytes

MI_Calcsize (LONG) - berechnete Länge des Modules in Bytes

MI_SpecialInfo (STRPTR) - Zeiger auf Sonderinformationen als Text

MI_LoadSize (STRPTR) - Anzahl der geladenen Bytes für SoundSysteme, die externe
          Dateien nachladen
MI_Unpacked (LONG) - Ungepackte Länge in Bytes (z.B. wie lang ein Propacker-
        modul als Protracker wäre)
MI_UnPackedSystem (LONG)(STRPTR) - gibt an, aus was dieses Format entstand,
         entweder eine interne Nummer(siehe unten) oder ein String,
         der den Namen enthält
         Folgende Varianten wurden bisher vorgesehen
         MIUS_OldSoundtracker
         MIUS_Soundtracker
         MIUS_Noisetracker
         MIUS_Protracker
MI_Prefix (STRPTR) - Zeiger auf ein Präfix für den Namen des Modules, so z.B.
         'Mod.' oder 'Mdat.'. So kann man das Modul unter dem
         richtigen Namen mit einer passenden Kennung abspeichern.

MI_About (STRPTR)  - Zeiger auf einen Informationstext zum Player.
```

```
MI_MaxSubSong (LONG)  - Anzahl der maximal möglichen Untersongs bei diesem
      Soundformat.

MI_Voices (LONG)  - Anzahl der benutzen Stimmen bei diesem Soundformat.

MI_MaxVoices (LONG)  - Anzahl der maximal möglichen Stimmen bei diesem
      Soundformat.
```

1.18 analyzeransteuerung

```
 6.6.3 Analyzeransteuerung
----------------------------

Die Analyzeransteuerung erfolgt mit Hilfe der UPS_USER - Struktur , die im
folgenden erläutert wird. (Übergabe dieser siehe Kapitel 6.6.1 "EP_Structinit)

Ab Eagleplayer V1.50 kann eine interne UPS_Struktur verwendet werden. Die
Adresse steht in EPG_UPS_Structure. Sie ist vorinitialisiert und wird bei
jeder Volume/Balance/Voice-Änderung automatisch gefüllt (UPS_DMACon).
Dabei muß unbedingt das Flag EPF_InternalUPSStructure beim Tag EP_Flags
gesetzt werden. Sie muß und darf nicht freigegeben werden.

Achtung. Es kann passieren, daß die Struktur in den nächsten Versionen des
Eagleplayers geändert wird, um auch SoundKarten und A5000 zu unterstützen!


Die Struktur sieht so aus:

 STRUCTURE UPS_USER,0

  APTR  UPS_Voice1Adr
  UWORD UPS_Voice1Len
  UWORD UPS_Voice1Per
  UWORD UPS_Voice1Vol
  UWORD UPS_Voice1Note
  UWORD UPS_Voice1SampleNr
  UWORD UPS_Voice1SampleType
  UWORD UPS_Voice1Repeat

  LABEL UPS_Modulo

  APTR  UPS_Voice2Adr
  UWORD UPS_Voice2Len
  UWORD UPS_Voice2Per
  UWORD UPS_Voice2Vol
  UWORD UPS_Voice2Note
  UWORD UPS_Voice2SampleNr
  UWORD UPS_Voice2SampleType
  UWORD UPS_Voice2Repeat

  APTR  UPS_Voice3Adr
  UWORD UPS_Voice3Len
  UWORD UPS_Voice3Per
  UWORD UPS_Voice3Vol
```

```
  UWORD UPS_Voice3Note
  UWORD UPS_Voice3SampleNr
  UWORD UPS_Voice3SampleType
  UWORD UPS_Voice3Repeat

  APTR  UPS_Voice4Adr
  UWORD UPS_Voice4Len
  UWORD UPS_Voice4Per
  UWORD UPS_Voice4Vol
  UWORD UPS_Voice4Note
  UWORD UPS_Voice4SampleNr
  UWORD UPS_Voice4SampleType
  UWORD UPS_Voice4Repeat

  UWORD UPS_DMACon

  UWORD UPS_Flags
  UWORD UPS_Enabled
  UWORD UPS_Reserved

  LABEL UPS_SizeOF
```

Die Einträge haben folgende Bedeutungen

UPS_Voice?Adr - Adresse des Samples, das gerade auf dieser Stimme gespielt
    wird

UPS_Voice?Len - Länge des Samples, das gerade auf dieser Stimme gespielt wird

UPS_Voice?Per - aktueller Wert der Sampleperiod, spielt eine Schlüsselrolle,
    wird eine Periode<>0 übergeben, so heißt das im allgemeinen,
    daß eine neue Note gespielt wird. Die Sampleperiod ist un-
    abdingbar für den Analyzerbetrieb. Können Sie nicht heraus-
    finden, wann eine Note angespielt wird, so setzen Sie die
    Periode halt immer dann, wenn auf die Audio-Hardware zugegrif-
    fen wird($DFF0A6/B6/C6/D6), gilt auch für Samplelänge und
    Adresse

UPS_Voice?Vol - Lautstärke, die auf die Hardwareregister geschrieben werden
    soll, Lautstärkeregelung wird dabei nicht berücksichtigt, d.h.
    wenn der UrWert z.B. 64 ist, die Lautstärke aber nur 32
    beträgt, will ich nicht 32, sondern 64 sehen, verstanden !
    (Berücksichtigung der Lautstärke ist schon anderweitig vor-
    gesehen, Siehe UVolume)

UPS_Voice?Note- noch nicht unterstützt

UPS_Voice?Samplenr - gibt die aktuelle Samplenummer an , noch von keiner Play-
    routine und keinem Engine unterstützt.

UPS_Voice?Sampletype - Sampletyp, noch nicht unterstützt

UPS_Voice?Repeat - gibt an, ob das Sample nur einfach - oder sich wiederholend
    gespielt wird , wenn der Wert 0 ist, heißt das Repeat ein,
    wenn er 1 ist, so ist der Repeat ausgeschaltet

UPS_DMACon   - gibt an, welche Stimmen ein/ausgeschaltet sind, Bit 0 für

```
         Kanal 0 ,Bit 1 = Kanal 1 usw. , ist das Bit gesetzt, so ist
         der Kanal eingeschaltet (Name etwas verwirrend, der Übergabewert
         sollte sich eigentlich auf die "EP_Voices"-Funktion ,Kapitel
         6.6.1 beziehen)

UPS_Flags    - Flagbits, die angeben , welche Möglichkeiten der UPS_USER -
         Struktur der jeweilige Player nutzt.

    UPSFL_Adr - Sampleadresse
    UPSFL_Len - SampleLänge
    UPSFL_Per - Sampleperiod (WICHTIG!)
    UPSFL_Vol - Lautstärke
    UPSFL_Note  - Note, noch nicht unterstützt
    UPSFL_SNr - Samplenummer
    UPSFL_STy - Sampletyp, noch nicht unterstützt
    UPSFL_DMACon  - welche Stimmen an/aus sind

UPS_Enabled  - gibt an, ob Zugriff auf die Struktur erlaubt ist, 0 heißt ja,
         eine Angabe <>0 bedeutet, daß die Struktur zur Auswertung ge-
         sperrt ist.

       Die restlichen Einträge sind für zukünftige Versionen des Eagleplayers
       vorgesehen.

------------------------------ Achtung ---------------------------------------
Für die derzeitigen Engines wird erwartet, daß mindestens UPSF_Adr,
UPSF_Len,UPSF_Per, UPSF_Dmacon und UPSF_Vol gesetzt und unterstützt werden.
Werden noch die anderen Parameter (UPS_Voice?Adr, UPS_Voice?Len, UPS_Voice?Per,
UPS_Voice?Vol) gesetzt und UPS_Enabled nach verlassen der Playroutine "0" ist.
------------------------------------------------------------------------------
```

1.19 die neuen eagleplayer-globals

```
 6.6.4 Die neuen Eagleplayer-Globals (ab Eagleplayer V1.10+)
-------------------------------------------------------------

Nachdem in der Vergangenheit auf Änderungen in den Globals verzichtet wurde,
ist es notwendig geworden, einige Merkzellen bzw. Unterprogramme hinzuzufügen.

Bei den neuen Eagleplayer-Globals werden die Argumente nicht mehr in Registern
sondern in Argumentzellen übergeben. Davon sind 8 Stück vorhanden. In der
Merkzelle EPG_ArgN muß immer die Anzahl der Argumente stehen. Sollte ein
Unterprogramm mehrere Argumente verlangen, müssen diese auch übergeben werden
und vor allem muß EPG_ArgN immer auf den max. gesetzt werden. Sollte die
Parameterübergabe anders von statten gehen, wird darauf hingewiesen.

Die Unterprogramme des Eagleplayers dürfen von jedem Replayer genutzt werden,
außer im Interrupt. Wenn nicht anders darauf hingewiesen wird, dürfen User-
programme die Subroutinen nur nach einen USClass_LockEP benutzen.

Im folgenden werden die wichtigsten Eagleplayer-Unterprogramme erklärt:
```

```
----- EPG_SaveMem -----
    Es wird ein Speicherbereich unter Berücksichtigung des Save-
    Modes gesichert. Diese Funktion ist erst in der registrierten
    Version möglich. Ist der Savemode -1, wird der im Eagleplayer
    eingestellte Save-Mode verwendet.

  Input:  Arg1 = Startadresse
    Arg2 = Länge des Speicherbereiches
    Arg3 = Pathadresse
    Arg4 = SaveMode (-1=Eagleplayereinstellung
          0=nicht gepackt
          1=PP-Crunched
          2=LH-Crunched
          3=XPK-Crunched
    Arg5 = Flags
      Bit 0=0   Anzeige im Playerwindow 0=ja
      Bit 1=1   Zieldatei immer deprotecten
      Bit 2=1   Safe Save
    ArgN = 5

  Output: Arg1 = Ergebnis (0=Alles ok)



----- EPG_FileRequest -----
    Es wird ein Filerequester unter Berücksichtigung des einge-
    stellten Filereq-Mode geöffnet. Es kann zwischen einer
    FileSelektion und einer DirSelektion unterschieden werden.

  Input:  Arg1 = Filerequester Titlename
    Arg2 = Directory Path
    Arg3 = Filename
    Arg4 = Window
    Arg5 = Filerequestertype (1=Fileselekt 0=Dirselekt)
    Arg6 = OutPut-Text für Eagleplayer-Statuswindow
    ArgN = 6

  Output: Arg1 = Ergebnis (0=Cancel oder Systemfehler, sonst 1)



----- EPG_TextRequest -----
    Es wird ein Textrequester geöffnet. Übergeben werden muß ein
    Ascii-Text. Dieser wird dann ausgwertet und das Window wird
    der Größe des Textes angepaßt. In dem Text können Kennungen
    übergeben werden, die auf Argumente zeigen, die aus der
    Argumentenliste entnommen werden. Zudem kann angegeben werden
    wie viele Gadgets man verwenden will und es können eigene
    Image-Daten ins Window mit übernommen werden. Auf Kick2.0
    ist der Textrequest Publicscreen unterstützt.
    Hinweis: Die Routine weißt noch einen bisher nicht gefundenen
    Bug auf. Die Routine sollte aber trotzdem in Playern benutzt
    werden, weil in späteren EPversion der Fehler hoffentlich
    beseitigt ist.

  Input:  Arg1 = TextAdresse
    Arg2 = Pointer to Pubscreenname (nur Kick2.0, sonst 0)
```

```
    Arg3 = Position on Screen (x.w & y.w)
    Arg4 = Pointer to Gadgetnames
    Arg5 = Poniter to Requestername
    Arg6 = Pointer to ArgumentListe
    Arg7 = Pointer to ImageDatas
    ArgN = 7

    Kennungen für Argumente: %s - String
           %d - Zahl in Dezimal angeben

  Output: Arg1 = Ergebnis   0=Fehler (z.B Window zu groß)
          sonst Nummer des Gadgets



----- EPG_LoadExecutable -----
    Es wird ein ausführbares Programm geladen. Es wird entpackt,
    falls dies möglich ist

  Input:  Arg1 = FilePath
    ArgN = 1

  OutPut: Arg1 = Einsprungadresse des Programms
    d0   = Fehler (0=alles ok)



----- EPG_NewLoadFile -----
    Wie DTG_LoadFile, nur das hier die Memeigenschaften mit ange-
    geben werden.
  Input:  Arg1 = Memeigenschaften
    ArgN = 1
    DTG_PathArrayPtr = Path des Files

  Output: d0 = Ergebnis (0=alles ok)



-- EPG_ScrollText --

    Scrollt den angegebenen Text ins Statuswindow des Eagleplayers
    Wird der Text mit Null abgeschlossen, bleibt er stehen, wird er
    mit eins abgeschlossen wird er im Loop gescrollt, wird er mit
    zwei angeschlossen, wird der Text bis an den linken Rand ge-
    scrollt, falls dieser noch nicht erreicht ist.

  Input:  Arg1 = Textadresse
    ArgN = 1

----- EPG_LoadPlConfig -----
    Lädt eine PlayerConfig. Diese Funktion wurde eingeführt um
    beim Laden der Replayer, wenn kein Env-Verzeichnis existiert,
    nicht andauernd Cancel zu drücken. Es wird getestet, ob die
    Config im Env-Verzeichnis liegen soll oder nicht. Ist nun das
    Verzichnis nicht vorhanden, wird keine Config geladen.
    (Funktion noch nicht eingebaut)
```

```
  Input:  Arg1 = ConfigPath
    ArgN = 1

  Output  Arg1 = Ergebnis


----- EPG_SavePlConfig -----
    Speichert eine PlayerConfig ab. Diese Funktion wurde eingeführt
    um beim Saven der ReplayerConfiguration, wenn kein Env-Ver-
    zeichnis existiert, nicht Cancel zu drücken. Es wird getestet,
    ob die Config im Env-Verzeichnis liegen soll oder nicht. Ist
    nun das Verzichnis nicht vorhanden, wird keine Config geladen.
    (Funktion noch nicht eingebaut)

  Input:  Arg1 = ConfigPath
    Arg2 = Startadresse
    Arg3 = Endadresse
    Arg4 = SaveMode (siehe EPG_SaveMem)

  Output: Arg1 = Ergebnis (0=alles ok)


----- EPG_FindTag -----
    Sucht einen Tag in der angegebenen TagListe. Die Funktion ist
    Kickstart unabhängig und darf auch von Enginesn aus
    ohne USClass_LockEP benutzt werden.

  Input:  a0 = Tagliste
    d0 = Tag

  Output: d0 = Wert des Tags
    d1 = Tag gefunden (0=nein,dann ist d0 auch 0, 1=ja)


----- EPG_FindAuthor -----
    Sucht den Autor eines Musicstückes in den angegebenen Grenzen.
    Die Routine wird normalerweise beim Soundtracker und seinen
    Mutanten angewendet, sie kann aber ach auf andere Systeme
    übertragen werden. Es wird in den Samplenamen nach der Kennung
    "by" bzw. "#" gesucht. Der nächste String ist dann der Autor-
    name. Der Autorname muß nicht copiert werden. Es reicht aus,
    wenn die Adresse des Autornamens in den ModuleInfo-Tag einge-
    tragen wird.

  Input:  Arg1 = Start des 1. Samplenamens
    Arg2 = Offset zum nächsten Sample
    Arg3 = Länge des Samplenamens
    Arg4 = Sampleanzahl
    ArgN = 4

  Output  Arg1 = Pointer to Autorstring oder NULL
    Arg2 = Länge des Autorstrings oder NULL

----- EPG_Hexdez -----
    Convertiert die Hexzahl in d0 in eine dezimale Ascii-
    Darstellung. Diese Funktion kann auch von Enginesn
    ohne Probleme genutzt werden.
```

```
  Input:  d0 = Hexzahl
    d1 = Flags  Bit 0=1 Nullen verstecken
        Bit 1=1 Vorzeichen benutzen
    a0 = OutPutpuffer
  OutPut: -


----- EPG_TypeText -----
    Es wird ein Text ins Mainwindow geprintet. Diese Funktion kann
    auch von Enginesn genutzt werden.

  Input:  A0 = Adresse ds Testes
  OutPut: -


----- EPG_ModuleChange -----
    Ein Module wird nach den Vorgaben umgebaut. Diese Funktion wird
    benutzt um Module mit Playroutine analyzerfähig und systemkon-
    form zu machen. Vor- und nach Ablaufen der Hauptfunktion wird
    der Cache, falls vorhanden, gelöscht.

  Input:  Arg1  = Startadresse der umzubauenden Daten
    Arg2  = max Länge
    Arg3  = Umbautabelle
    Arg4  = 1.b=1 Mehrmals eine Routine umbauen
        2.b=1 2. Umbauroutine benutzen
        3.b=1 keine Suche nach Werten
        4.b=1 Keine Suche nach Jmp
    Arg5  = 1.w Kennbyte für Jump
        2.w Kennbyte für Wert
    ArgN  = 5
  OutPut: Arg1  = Fehlernummer oder Null
    Arg2  = Anzahl der Umbauten


----- EPG_ModuleRestore -----
    Ein Module muß vor dem Saven in den Originalzustand zurück
    gesetzt werden. Dies erledigt die Funktion ModuleRestore.
    Vor- und nach Ablaufen der Hauptfunktion wird der Cache,
    falls vorhanden, gelöscht.

  Input:  Arg1  = Startadresse der umzubauenden Daten
    Arg2  = max Länge
    Arg3  = Umbautabelle
    Arg4  = 1.b=1 Mehrmals eine Routine umbauen
        2.b=1 2. Umbauroutine benutzen
        3.b=1 keine Suche nach Werten
        4.b=1 Keine Suche nach Jmp
    Arg5  = 1.w Kennbyte für Jump
        2.w Kennbyte für Wert
    ArgN  = 5
  OutPut: Arg1  = Fehlernummer oder Null
    Arg2  = Anzahl der Umbauten


Ab der Eagleplayerversion 1.50 gibt es einen Eintrag in den Globals namens
```

"EPG_NewJumpTab". Dies ist ein Pointer auf noch mehr Unterprogramme die
genutzt werden können. Der große Unterschied besteht darin, das diese direct
angesprungen werden können, also wie eine Library !
Diese Funktion ist zwar implementiert, allerdings darf sie noch nicht genutzt
werden.

Desweiteren sind in den Globals die benutzten Librarybasen festgehalten. Unterlassen Sie es unbedingt, diese zu verändern. Sämtliche neuen Globals sind nur Lese-Globals, außer den Argumentzellen.

1.20 Einbindung des Eagleplayers in andere Programme

```
 Einbindung des Eagleplayers in andere Programme
```

Der Eagleplayer bietet die Möglichkeit über den eigenen EnginePort
eine Kommonikation mit völlig eigenenständigen Programmen zu vollziehen.
Es werden normale Exec_Messages ausgestauscht, mit der Messagelänge
"UM_Sizeof" und dem Eagleplayerinternen Typ "USM_ExternalPrg".

Ein Beispiel für diese Möglichkeiten bietet der auch auf dem Aminet
verfügbare Noiseconverter. Er gibt an den Eagleplayer das Kommando,
ein bestimmtes Module aus dem Speicher zu laden und abzuspielen
und umgekehrt. Auch der ExoticRipper unterstützt dieses Feature,
indem er gerippte Module direkt an den Eagleplayer weitergeben
kann.

Es ist auch möglich, ein Extloading zu emulieren, so daß ein Abspielen aus
einem Ripper gar kein Problem ist.

Hier nun ein kleiner Sourcetext zur Demostration. Er ist voll lauffähig auf
ASM-One und Devpac. Wer also mal ein bischen testen will, braucht ihn nur
aus der Dok zu pflücken.
(Schauen Sie auch ruhig in die englische Doc. Dort wurde ein anderer
Sourcetext mit eingefügt)

```
*****************************************************************
*      PlayMem, Message an Eagleplayer senden                   *
*   ----------------------------------------                    *
*                                                                *
* Dieses Sourcecode ist ein Beispiel, um Speicher im Eagleplayer abzu-  *
* spielen. Es sollte dabei logischerweise um ein heiles Module handeln. *
* Es werden dabei die Startadressen der jeweiligen Files sowie deren  *
* Größe angegeben. Probleme gibt es bei Replayern, die Check1 benutzen. *
* Diese Funktion ist hervorragend für Ripper (Hi Turbo) geeignet.       *
*                                                                *
* Kick2.0+ ist Vorraussetzung !!!                                *
* Geschrieben mit ASM-One/Devpac.     © DEFECT 1993              *
*****************************************************************
    incdir  "include:"
    include "misc/eagleplayer.i"
    include "exec/exec_lib.i"
```

```
  *-------------- Eagleplayer-TestProgramm ------------*
Prg_Start:  bsr.s PlayEagle
    moveq #0,d0
    rts


  *------------- Eagleplayer gestartet ?? -------------*
PlayEagle:  move.l  4.w,a6
    lea EP_Portname(pc),a1
    jsr _LVOFindport(a6)  ;Eagleplayers Port finden
    tst.l d0
    beq.w .NoEagleplayer    ;Cancel->Ende des Programms
    move.l  d0,EP_Port

  *------ Schnell mal einen Port Createn ---------*
    jsr _LVOCreateMsgPort(a6)
    move.l  d0,TP_Port
    suba.l  a1,a1
    jsr _LVOFindTask(a6)  ;Task finden
    move.l  d0,d6

  *------------ Message vorbereiten --------------*
    lea MyMessage(pc),a1
    clr.l (a1)      ;LN_Succ löschen
    clr.l 4(a1)     ;LN_Pred löschen
    move.w  #$800,8(a1)   ;LN_Type & Pri setzen
    move.l  #PlayMem.MSG,10(a1) ;LN_Name setzen
    move.l  TP_Port(pc),14(a1)  ;Portadresse,an die
            ;zurückgesendet wird
    move.w  #UM_sizeof-20,18(a1)  ;Length festlegen

    move.w  #-1,UM_UserNr(a1)
    move.l  #USM_Externalprg,UM_Type(a1)
    move.l  d7,UM_Userport(a1)
    move.l  d6,UM_TaskAdr(a1)
    move.l  #-1,UM_Signal(a1)
    move.w  #USClass_Command,UM_class(a1)
    move.l  #0,UM_Userwindow(a1)
    move.l  #UCM_PlayMem,UM_Command(a1)

  *------- Argumentstruktur initialisieren -------*
    moveq #EPT_String+30,d0
    movem.l d1-a6,-(Sp)
    move.l  4.w,a6
    move.l  #$10001,d1
    jsr _LVOAllocmem(A6)
    movem.l (sp)+,d1-a6
    tst.l d0
    beq .DelMsgPort

  *--- Hauptfile in spezieller Struktur übergeben ---*
    move.l  d0,a2
    move.l  a2,UM_ArgString(a1)
    move.l  #EPT_String+30,EPT_Stringsize(a2)
    clr.l EPT_Next(a2)
```

```
    move.l  DatenPuffer(pc),EPT_Result1(a2)   ;Arg1 (Adr)
    move.l  DatenPuffer+4(pc),EPT_Result2(a2) ;Arg2 (Size)

  *----------- den Filenamen übergeben --------------*
  * Auf jeden fall sollte eine Kennung übergeben     *
  * werden, durch die erkannt wird, welcher Replayer *
  * genutzt wird. Der entsprechende Player wird dann *
  * geladen !!             *
  ************************************************
    move.l  a2,a4
    moveq #30,d1
    lea.l My_Filename(pc),a3
    lea EPT_String(a2),a2
.Copyfilename:  move.b  (a3)+,(a2)+
    dbeq  d1,.Copyfilename

    clr.l UM_Result(a1)
    move.l  a1,a2


  *-- Nun die Argumentenstrukturen für EXTLoading füllen --*
    lea DatenPuffer(pc),a3
    move.l  DatenPufferAnz(pc),d7
    subq.l  #2,d7
    blo.s .NoExtLoading

  *---- Speicher für nächste Argumentstruktur besorgen ----*
.Fill:    move.l  4.w,a6
    move.l  #EPT_String,d0
    move.l  #$10001,d1
    jsr _LVOAllocMem(a6)
    move.l  d0,a1
    tst.l d0
    beq.w .OutOfMem

  *----------- nächste Argumentstruktur füllen ------------*
    move.l  d0,(a4)
    move.l  d0,a4
    addq.l  #8,a3
    move.l  (a3),EPT_Result1(a1)
    move.l  4(a3),EPT_Result2(a1)
    move.l  #EPT_String,EPT_StringSize(a1)
    dbf d7,.Fill


  *---------------------- Message schicken ---------------------*
  *- Der Eagleplayer kopiert die entsprechenden Daten in selbst -*
  *- Allokierten Speicher, so daß der Speicher hier beim Test   -*
  *------- nach Replyen der MSG weiter genutzt werden kann ------*
.NoExtLoading:  lea MyMessage(pc),a1
    move.l  4.w,a6
    move.l  EP_Port(pc),a0
    jsr _LVOPutmsg(a6)

  *---------- Auf Message warten ------------*
.Wait:    move.l  TP_Port(pc),a0
    move.l  4.w,a6
```

```
    jsr _LVOWaitPort(A6)
    move.l  TP_Port(pc),a0
    jsr _LVOGetMsg(A6)
    tst.l d0
    beq.s .Wait
    move.l  d0,a1
    cmp.l #USM_ExternalPrg,UM_Type(A1)  ;Handelt es sich um
    beq.s .Ok       ;unsere MSG
    jsr _LVOReplyMsg(a6)
    bra.s .Wait

  *--- Zurückgekommende Message auswerten ---*
.Ok:    move.l  UM_ArgString(A1),d0   ;Antwort kopieren und
    beq.s .NotAnswered
    move.l  d0,a4
    move.l  EPT_Stringsize(a4),d0
    move.l  d0,d1
    sub.l #EPT_String-1,d1
    blo.s .FreeNextStr
    lea EPT_String(a4),a3
    lea StringTemp(pc),a2
.Copy:    move.b  (a3)+,(a2)+
    dbf d1,.Copy

  *----------------- Argumentenstrukturen freigeben ---------------*
  *- Achtung: Es können immer mehrere Argumentstrukturen gesendet -*
  *- werden. Immer alle freigeben !!!!!!!!!!!!!!!!!!!!!!!!!!!!!! -*
.FreeNextStr: move.l  a4,a1
    move.l  EPT_StringSize(a1),d0
    move.l  (a4),a4
    move.l  4.w,a6
    jsr _LVOFreeMem(a6)
    moveq #0,d0
    cmp.l a4,d0
    bne.s .FreeNextStr

  *--- Ausschrift im TestProgramm "Playing ..." ---*
    lea StringTemp(pc),a1   ;kann verdammt lang
    bsr ShowStatus      ;sein, Aufpassen !!!!
    bra.s .DelMsgPort

  *-- Ausschrift im TestProgramm "Keine Antwort" --*
.NotAnswered: lea Stat_NoAnswer(pc),a1
    bsr ShowStatus
.DelMsgPort:  move.l  TP_Port(pc),d0
    beq.s .NoPort
    move.l  d0,a0
    move.l  4.w,a6
    jsr _LVODeleteMsgPort(a6)
.NoPort:  clr.l TP_Port
    rts

.NoEagleplayer: lea Stat_NoEagleplayer(pc),a1
    bra ShowStatus


  *-- Anderen Argumentstrukturen freigeben --*
```

```
.OutOfMem:  move.l  MyMessage+UM_ArgString(pc),a4
.FreeStruct:  move.l  EPT_String(a4),d0
    move.l  a4,a1
    move.l  (a4),a4
    move.l  4.w,a6
    jsr _LVOFreeMem(a6)
    moveq #0,d0
    cmp.l d0,a4
    bne.s .FreeStruct
    lea Stat_OutOfMemory(pc),a1
    bsr ShowStatus
    rts

  *----------- Unterprogramm zum printen von Texten ----------*
ShowStatus: rts


*-------------------------- Variablenvereinbarungen ----------------------*
Stat_OutOfMemory:dc.b "Out of Memory !",0
Stat_NoEagleplayer:dc.b "No Eagleplayer found !",0
Stat_NoAnswer:  dc.b  "Keine Antwort vom Eagleplayer !",0
StringTemp: ds.b  100
EP_PortName:  dc.b  "EAGLEPLAYERPORT",0 ;Name des Eagleplayerportes
PlayMem.MSG:  dc.b  "EP_PlayMem",0

    even
MyMessage:  ds.b  UM_SizeOf   ;Messagemem

My_Filename:  dc.b  "MDAT.NoName",0   ;Prefix sollte angegeben sein,
            ;um mögliches Playerloading
            ;when needed (Batch) zu
            ;unterstützen


EP_Port:  dc.l  0         ;PortAdr des EP
TP_Port:  dc.l  0         ;PortAdr des TP

*------- Dies ist nur ein Hilfspuffer zum Initialisieren der UM_Message ------*
*------------ Er muß dann entsprechend vorher gefüllt werden -----------------*
DatenPufferAnz: dc.l  2     ;Anzahl der Files
DatenPuffer:  dc.l  Data1     ;Adr 1 File
    dc.l  Data1Size   ;Size 1 File
    dc.l  Data2     ;Adr 2 File
    dc.l  Data2Size   ;Size 2 File


*------------------------------ Testmodule laden -------------------------*
*-- Kann auch im FastMem liegen, weil der Eagleplayer das Module kopiert. --*
*---------------------------------------------------------------------------*
    incdir  "xModules:TFMX/"  ;entpackt laden !!!
Data1:    incbin  "MDAT.Turrican-1"
Data1Size = *-Data1

Data2:    incbin  "SMPL.Turrican-1"
Data2Size = *-Data2
```

1.21 Die Engines

```
  Warum haben wir noch keine Programmierrichtlinien herausgegeben ?
 ------------------------------------------------------------------
```

Das große Problem besteht darin, daß die Engines schon in der Version
1.0 des Eagleplayers sehr komplex aufgebaut waren und ab der Version 1.10+
ein Haufen neuer Funktionen dazukam. Es sei dazu nur die Möglichkeit erwähnt,
externe, vom User ausgewählte Oberflächen zu entwickeln. Unser zweites großes
Problem ist das Zeitproblem, mal abgesehen davon, daß dann sowieso keiner
durchsehen würde, selbst wenn wir uns große Mühe in der Dok geben würden.
(Wir sehen ja selbst bald keine Sonne mehr :)-=-=)
Wer unbedingt eine Engine programmieren will, der sollte sich bei uns
melden.