

Programmierung_English

COLLABORATORS

	<i>TITLE :</i> Programmierung_English	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		February 9, 2022
<i>SIGNATURE</i>		

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Programmierung_English	1
1.1	Eagleplayer 2.00 Developer-Dokumentation	1
1.2	Danksagungen	1
1.3	The Players	1
1.4	Introduction to the external players	2
1.5	the external player concept	2
1.6	The structure of external players	2
1.7	The structure of custom modules	3
1.8	adapting players	3
1.9	playerheader	3
1.10	Module recognition	4
1.11	interrupts	5
1.12	Meaning of the Tags	6
1.13	Eagleplayer support functions	12
1.14	Hints and Tips	19
1.15	moduleinfo	20
1.16	analyzersteuerung	21
1.17	die neuen eagleplayer-globals	24
1.18	Einbindung des Eagleplayers in andere Programme	24
1.19	The Engines	27

Chapter 1

Programmierung_English

1.1 Eagleplayer 2.00 Developer-Documentation

Developer-Documentation
for
Eagleplayer

Version 2.00
documentation \$VER: V2.00 (01-August 97)
© 1993-97 Defect Software Productions

Contents:

Introduction
Programming of Players
Programming of Engines
supplying moduledata from other programs
Danksagungen

1.2 Danksagungen

Thanks to all who helped us with tips, wishes, bugreports etc.
We hope you are enjoying the result of more than 4 years of programming.

1.3 The Players

Programming of external players:

Introduction to the external players some first words
The external Player concept
The structure of external players structure and basic Tags
The structure of custom modules what is different to normal replayers

- How to build external players
- Playerheader
- Modulerecognition
- Interrupts
- Meanings of the Tags
- Eagleplayer support functions
- Moduleinfo
- Scopes control
- Hints and Tips

1.4 Introduction to the external players

programming of external replayers

Eagleplayer is built up in a modular way by supporting external replayers and engines as plugins. Those are executables which are loaded on demand.

The Player API is largely backwards compatible to Delitracker 1.3 so if you are familiar with Delitracker it shouldn't be a problem to add Eagleplayer support into your replayers.

1.5 the external player concept

"the external Player concept"

External replayers are executables which will be loaded using the AmigaDOS LoadSeg() function to get relocated. They will be recognized by a specific structure at the beginning of the file generated by the "PLAYERHEADER" macro from the include file.

The functions of the players are defined by a list of TagItems whose address is given as an argument to the PLAYERHEADER macro. A replayer or custom module may contain more than just one hunk, it is only important that the first hunk starts with the player header.

A special kind of Players are the custom modules who contain both, replay and module meant for modules hard to relocate in memory by using hardcoded addresses or special formats where just a single module exists for.

1.6 The structure of external players

The structure of external players

```
{ Playerheader      } to make the object identified as a replayer
{ TagArray          } description of abilities of the replayer
{ Interfacecode     } checkroutine,names,checkroutines etc.
{ Replaycode/datas } the replay itself
```

```
minimal required Tags:
DTP_PlayerName
DTP_Creator
DTP_Checkx or EP_Checkx
DTP_InitPlayer
DTP_InitSound
DTP_Interrupt or DTP_Startint/DTP_Stopint
```

1.7 The structure of custom modules

The structure of custom modules

Custom modules are more than simply songs+sampledata. Quite comparable to music formats like Hippel or David Whittaker they consist of moduledata with embedded replay. The replay in custom modules is built up the same way normal external replayers are with some extra Tags.

The custom port is meant for formats where just a single module exists OR the module can't be relocated in memory due to hardwired addresses. If there exist more than just a single module for a format it may be wiser writing a normal replayer instead of adapting each module as a custom module.

Custom modules differ in 3 points from an external replay, they:

- do not need a check routine
- have got the DTP_CustomPlayer link "Bedeutung der Tags"} Tag set
- have got their module to play right within

```
{ Playerheader      }  custom modules will be in the first step recognized as  ↔
  replayers
{ TagArray          }  description of abilities of the replayer
{ Interfacecode     }  checkroutine,names,checkroutines etc.
{ Replaycode/datas }  the replay itself
{ SoundDatas       }  datas for the module
```

1.8 adapting players

Adapting players

It isn't hard at all to adapt a replay routine to Eagleplayer. Everything you have to do is to write a little bit interface code. Don't panic, Eagleplayer offers a lot of useful routines which will make this task really easy.

To adapt a replay you need either the source code of the replay, a linkable object file of the replay or the documentation of a library you want to use. Furthermore you should have got some modules of that format for testing purposes.

1.9 playerheader

6.2.1 Playerheader

The PLAYERHEADER macro generates the header which is needed for the file to be identified as external replay. The macro MUST be the first code line in your file. The needed parameter is a pointer to a Tag Array where Eagleplayer will look for all functions the replay offers to Eagleplayer.

At every function call from Eagleplayer Eagleplayer's data base address can be found in register A5 (except the interrupt calls, namely DTP_Interrupt) to address global variables and settings, see 'Misc/Deliplayer.i & 'Misc/Eagleplayer.i'.

The contents of the CPU integer registers (D0-A6) are preserved automatically by all support functions so that the replay doesn't have to care about them.

```
PLAYERHEADER <Tagarray>
```

Tagarray - Pointer to a TagArray Structure, terminated by TAG_DONE.
 Tags containing Ti_Data==NULL are ignored, Tag_More,
 Tag_Ignore etc. are supported.

1.10 Module recognition

Module recognition

To make Eagleplayer be able to differ between the single module formats every player has got an own code to recognize the according modules. This routine checks relevant areas within the module which are unique for that format. Examples: 'M.K.' at Offset \$438 for Protracker or "MMD" at offset 0 for MED/OctaMED. Some music formats have got their own replay code in the module, e.g. David Whittaker, Hippel, FRED, MarkII or Promizer 1.x. There your work is a little bit more complex since those branches or jumps at the beginning of the module are mostly not significant enough to ensure that the replay recognizes only modules of the right kind. Remember: Eagleplayer already supports more than 150 music formats. A reliable check routine is a MUST to prevent serious crashes.

The Checkroutine returns the result in D0,
 D0.l = 0 for positive check
 D0.l <> 0 if unknown module or unsupported subkind

Eagleplayer supports 2 basic kinds of music players

- 1) first method, the standard case
 using the DTP_Check2, EP_Check3 or EP_Check5 function entries

advantages:

- the replay doesn't have to care about loading
 - the module will be delivered to the checkroutine automatically depacked
-

- memory allocations will be handled by Eagleplayer, depending on the used Check Tag and Eagleplayer's settings Chip- or FastRam, see also here.

2) second method, if you want/need to do everything by yourself using the DTP_Check1 upcall

advantage: it is possible to create replayers loading the module by themselves

disadvantage: packed files aren't supported

This kind of players are meant only for special cases, such as replayers like WAVE, AIFF, 8SVX loading the sample whilst playing.

1.11 interrupts

Interrupts

Two different types of interrupts are offered:

1) Player, using Eagleplayer's interrupt (DTP_Interrupt)

Advantages:

- player runs independent from video mode
- automatically supports speed control bar
- doesn't have to care about the interrupt initialization code
- compatible to serial.device
- player supports user's choice of timing mode, timer.device, VBlank or CIA-B

Disadvantage:

- only one interrupt source offered which should be sufficient for most cases anyway

b) Players using their own interrupt

Advantage:

- other interrupt sources can be used, e.g. AudioIRQ(*)

Disadvantage:

- more code needed, when hacking onto CIA Hardware registers not future compatible

When using an own timer interrupt you should use the CIA B, Timer A. Do not execute your replay code directly in CIA Hardware interrupt. To prevent problems with serial connects you should use a Softint, called by the Exec function "Cause()" instead. Stay away from direct hacks to the 680x0 interrupt registers and vectors. Use the OS functions AddIntServer() and SetIntVector() instead.

Think twice if you really need own interrupt routines. Even if you need to use AudioInts there is a more elegant

solution. Please contact us if you need to use AudioInts, our Amplifier System already supports them in a more elegant way.

1.12 Meaning of the Tags

Meaning of the Tags

Except from the default system tag (TAG_DONE, TAG_IGNORE, TAG_MORE, TAG_SKIP) Eagleplayer offers the following tags to be used

DTP_CustomPlayer (BOOL) - this declares the replayer to a Customplayer
When using this Tag DTP_Check1/2 and EP_Check3/5 will be ignored

EP_PlayerVersion (LONG) - using this function you may ensure that at least the specified version of Eagleplayer is available. It is always a good idea to set ti_Data to "EAGLEVERSION" (defined in Eagleplayer.i) for use with the current Eagleplayer version and better.

EP_KickVersion (LONG) - this Tag specifies the minimum OS version it can be used with (37=OS 2.0, 39=OS 3.0, 40=OS 3.1)

EP_AttnFlags (LONG) - this Tag specifies the minimal CPU/FPU config this replayer requires. It is defined in the same way like Execbase's attnflags.
Example: EP_AttnFlags,AFF_68020

EP_EagleBase (APTR) - ti_Data points to a LONG variable to store Eagleplayer's data base address in. This variable is set immediately upon loading the replay to memory and always valid.
Example: EP_EagleBase,&MyEPBase

DTP_PlayerVersion (LONG) - ti_Data contains the version/revision of the replayer. The fixed part of the version is stored 16 bits left-shifted and the revision in the lower 16 bit.
Example: DTP_PlayerVersion,1<<16||2 (for V1.2)

EP_Date (LONG) - ti_Data contains the last modification date of the replayer.
Format: day<<24||month<<16||year
Example: EP_Date,01<<24||08<<16||1997

DTP_PlayerName (STRPTR) - ti_Data points to the name of the player. The name should be the same as the used filename to avoid trouble with playerloader. If you want to specify different names in case the player supports more than just one sound format, just have a look at the Moduleinfo section. required

DTP_Creator (STRPTR) - pointer to the name of the author, shown in Eagleplayer's manager. The string's length is not restricted, may contain linefeeds although manager handles line breaks automatically and needs to be NULL-terminated.

required

EP_EjectPlayer (FPTR) - If the replayer obtains own memory areas, filelocks etc. these can be freed at this point, after that upcall replayer will be removed.

usw. besorgt, kann der an dieser Stelle freigegeben werden, danach wird der Player durch den Eagleplayer entfernt.

DTP_Check1 (FPTR) - pointer to a recognition routine for the current module after 2200 bytes have been read into memory. Returns D0=0 if successful, D0<>0 if not. Address and size of the file can be found in DTG_ChkData and DTG_ChkSize.

Example:

```
dc.l DTP_Check1,check
dc.l TAG_Done,0
```

check:

```
move.l DTG_ChkData(a5),a0
move.l 8(a0),d0
sub.l #'8SVX',d0
rts
```

DTP_Check2 (FPTR) - pointer to a recognition routine for the current module after the whole module has been read (and depacked) into CHIP memory. Returns D0=0 if successful, D0<>0 if not. Address and size of the file can be found in DTG_ChkData and DTG_ChkSize, see above.

Warning The address in DTG_ChkData is only valid for the time the module is checked. The address of the module may and mostly will change until DTP_InitPlayer is called! It is not allowed to store the address from DTG_ChkData as the base pointer of your module. Use DTG_GetListData() instead.

EP_Check3 (FPTR) - pointer to a recognition routine for the current module after 2200 bytes have been read (AND depacked) into ANY public memory. Returns D0=0 if successful, D0<>0 if not. Address and size of the file can be found in DTG_ChkData and DTG_ChkSize, see above.

This check is intended to be used in case the replay supports Eagleplayer's Amplifier system or doesn't need Chipram for replaying in other cases. It should be used if the 2200 bytes are sufficient for recognizing the file type.

EP_Check5 (FPTR) - same as DTP_Check2 but module is loaded into ANY public memory.

EP_Check6 (FPTR) - same as DTP_Check1 but called after the EP_Check3 loop, used by the Datatype player. (please don't use this Tag except you have got a good reason)

Note: one of DTP_CheckX or EP_CheckX is required unless you're on writing a Custom module.

EP_CheckModule (FPTR) - optional additional check of the module after it was recognized by DTP_CheckX or EP_CheckX for any corrupt data or other faults. Returns D0=0 in case of no error, D0<>0 otherwise.

DTP_ExtLoad (FPTR) - pointer to an optional routine to load your module (e.g. DTP_Check1 based players) or additional files (e.g. TFMX sample files, Startrekker ".nt" files). In case of no error D0=0 has to be returned, D0<>0 otherwise

Note: Please don't forget to free all resources allocated during the Check and Extload process (Memory, Files, Locks, ...) before returning an error since no more player functions will be called then.

EP_FreeExtLoad (FPTR) - pointer to an optional routine to free data loaded by custom routines during the DTP_Extload routine. Doesn't need to be called if Eagleplayer's dtg_LoadFile or EPG_NewLoadFile functions have been used to load data.

DTP_Interrupt (FPTR) - pointer to an interrupt compliant routine called in standard case every 1/50 sec. Depending on the user's settings this is done by a Timer.device using routine, a CIA-B Interrupt or a VBlank interrupt. The interrupt frequency may be altered using the dtg_SetTimer function. Either this tag or a DTP_StartInt/DTP_StopInt combination are required

DTP_StartInt (FPTR) - pointer to an initialization routine which has to exist in case DTP_Interrupt is not available. When calling DTP_StartInt the sound has to be started by the replayer.

Notes:

- DTP_StartInt has got priority over DTP_Interrupt.
- If DTP_Startint and DTP_Interrupt are specified in the Taglist the function pointer of DTP_Interrupt's ti_Data will only be used for the "Fast Forward" feature.

DTP_StopInt (FPTR) - pointer to a cleanup routine called to stop the interrupt. This upcall is required in case that a DTP_StartInt Tag exists, else don't specify it.

DTP_InitPlayer (FPTR) - pointer to an initializing routine called after a module was successfully loaded. In case of no error D0=0 has to be returned, D0<>0 otherwise. At this point you may obtain the address(es) and sizes of file(s) loaded for DTP_Check2, EP_Check3 and EP_Check5 based players + files read by dtg_LoadFile or EPG_NewLoadFile using the dtg_GetListData function. Furthermore you have to allocate here the audio channels for replayers directly using Amiga's native audio hardware. (Amplifier players may not allocate the Audio channels, that is a task of the used Amplifier) If the replayer supports multiple songs within a module

dtg_SndNum(a5) has to be set to the number of the first subsong. If a routine for DTP_SubSongRange or a DTP_NewSubSongrange entry exists Eagleplayer does this step automatically and dtg_SndNum doesn't need to be initialized.

DTP_EndPlayer (FPTR) - pointer to an optional cleanup routine when removing the module from memory. At this point you have to free the audio channels. (Eagleplayer offers an easy to handle routine for that task)

DTP_InitSound (FPTR) - pointer to an optional initialisation routine. This one has got to initialize the module in a way to ensure that the module is playing from beginning when the interrupt is started. (e.g. set songposition to 0, reset speed & timer variables...)

DTP_EndSound (FPTR) - pointer to an optional cleanup routine. This one can e.g. set the volume registers to zero and disable DMA activity. (for Amplifier based players not needed)

DTP_Stop (FPTR) - pointer to an optional stoproutine. If this Tag doesn't exist Eagleplayer works in the following way:
stop Interrupt (DTP_StopInt)
sound cleanup (DTP_EndSnd)
song initialisation (DTP_InitSnd)
Furthermore this routine has got the task to stop an eventually playing song and initialize it in a way to play it at next interrupt start from beginning on.

DTP_Config (FPTR) - pointer to an optional initialisation routine. This one will be called only once after the player was loaded. possible use: loading a player specific configuration file or opening some libraries. (libs could be closed then upon the EP_EjectPlayer call)

DTP_UserConfig (FPTR) - Pointer to an optional configuration routine called when the user presses the "config" gadget for this player in manager.
possible use: opening a window to set options like instruments path or any other options and saving own configuration files to EP's config directory. (path can be found in EPG_ConfigDirArrayPtr)

DTP_SubSongRange (FPTR) - This Tag should be supplied if the player supports multimodules. ti_Data points to a function that returns in d0 the minimum and in d1 the maximum subsong number.

Note: if possible this Tag should be used instead of a DTP_NextSong/DTP_PrevSong pair. The number of the subsong to play can be obtained from dtg_SndNum(a5) in your DTP_InitSound-routine.

DTP_Volume (FPTR) - (for non Amplifier players only)
pointer to function that sets the volume. This function

is called every time the volume is changed (via Arexx or slider) and once at the initialising phase of the module (before DTP_InitSnd is called). The mastervolume can be found in dtg_SndVol(a5). The mastervolume is the highest volume allowed. The effective volume can be calculated using the following formula:

$$\text{VOL_eff} = ((\text{MASTERVOLUME} * \text{modulevolume}) \gg 6).$$

See also the example sources.

DTP_Balance (FPTR) - (for non Amplifier players only)
 pointer to a function setting the balance. This function is called every time the balance is changed by the user and once at the initialising phase of the module (before dtp_InitSnd is called). The balance for the left channel can be found in dtg_SndLbal(a5), for the right channel in dtg_SndRbal(a5). Note: All players that support balance are capable of volume too! Then you must use the same routine for both operations. The mastervolume for the left channels can be calculated with this formula:

$$\text{LeftMaster} = ((\text{dtg_Volume}(a5) * \text{dtg_SndLbal}(a5)) \gg 6).$$

 For the right channels the formula is similar.

EP_Voices (FPTR) - using this function the informations about the 4 voices will be submitted by Eagleplayer in D0. If the bit for the voice set, this channel is enabled.
 Bit 0 = Voice 0; Bit 1 = Voice1; Bit 2 = Voice 2;
 Bit 3 = Voice3

EP_SetSpeed (FPTR) - If your replayer uses own interrupt routines (DTP_StartInt/DTP_StopInt) this function call can be used to change the replaying speed of your module. The speed range is from -25 to +25 and will be delivered to the function in register D0. Alternatively it can be found in EPG_Speed(a5). Speed values <0 are meant for slower replaying, speed values >0 for faster playing.

DTP_NextPatt (FPTR) - pointer to a function that increases the song position by 1.

DTP_PrevPatt (FPTR) - pointer to a function that decreases the song position by 1.

EP_GetPositionNr (FPTR) - pointer to a function returning the current song position in register D0 for displaying purposes (Moduleinfo, BifatGUI, EMPYGui ...).

DTP_NextSong (FPTR) - pointer to a function that increases the subsongcounter (only if the next subsong exists).

DTP_PrevSong (FPTR) - pointer to a function that decreases the subsongcounter (only if the previous subsong exists).

EP_Get_ModuleInfo (FPTR) - This function of the replayer has to return a completely initialized Moduleinfotaglist or zero for no taglist in register A0. see also here

NOTE: it's still possible that this function is called when no module was loaded, please check if the replay has got a valid module address !

EP_Free_ModuleInfo (FPTR) - This optional function is for example to free previously allocated memory for the Moduleinfo-taglist. No return parameters expected.

EP_NewModuleinfo (APTR) - ti_Data points to a Moduleinfo taglist which has to get initialized during the DTP_Initplayer callup. It is an easier alternative to the old EP_Get_Moduleinfo method.

EP_Structinit (FPTR) - returns in A0 a pointer to a UPS_USER structure where the replayer puts the informations for the Analyzerprograms (Engines) about the currently played sampledata. This structure will then be given to all Engines who need those infos to work. See also here

EP_StructEnd (FPTR) - optional, e.g. to free the memory used for the structure

EP_LoadPlConfig (FPTR) - calls the routine loading (or reloading) the configuration of this replayer (EP's current configuration path can be obtained from EPG_ConfigDirArrayPtr(a5))

EP_SavePlConfig (FPTR) - calls the routine for saving the configuration of this replayer (EP's current configuration path can be obtained from EPG_ConfigDirArrayPtr(a5))

EP_Flags (LONG) - These flags within "TI_Data" of this tag show which functions the player basically supports. required This Tag was included because of some "self modifying" players which change the module to support Analyzer,Voices, Volume, Amplifiers...

When the internal replays of this module differ to another then it may happen that the replayer can't change the module in this way we need it so that one or the other function won't be available from module to module.

Therefore we defined the following flagbits for showing the basic abilities of the replayer in the Manager window:

EPF_Songend - the replayer supports Songend
 EPF_Restart - the module is restartable (should always be the case)
 EPF_Disable - The player is disabled (don't set by yourself!)
 EPF_NextSong - supports next subsong
 EPF_PrevSong - supports previous subsong
 EPF_NextPatt - supports jump to next songposition
 EPF_PrevPatt - supports jump to previous position
 EPF_Volume - volumecontrol possible *1
 EPF_Balance - balance possible *1
 EPF_Voices - can change the state of the voices
 EPF_Save - module can be saved
 EPF_Analyzer - Scope-support *1
 EPF_ModuleInfo- can tell infos to the current module
 EPF_SampleInfo- infos to the samples of the current mod

```

EPF_Packable - module may be packed (DTP_Check2, EP_Check3,
              EP_Check5 based players: set always)
EPF_VolVoices - volume of each voice can be different to the
              next (EPG_VolVoicel,...)

```

*1) these flags can always be set for players using the Amplifier system

undocumented Tags: (ask us for descriptions)

EP_InitAmplifier

```

EP_PatternInit ;Init Patterninfostruct
EP_PatternEnd ;Free Patterninfostruct - optional

```

```

EP_SampleInit ;create Sampleinfostructure
EP_SampleEnd ;ejected !!!

```

```

EP_Save
EP_ModuleChange ;Change Module
EP_ModuleRestore ;Restore Module
EP_SaveAsPT ;Save Module as Protrackermodule
EP_PlaySample ;Play Sample d0=SampleNr

```

```

EP_CreatorLNr ;for locale purposes
EP_PlayerNameLNr ;for locale purposes
EP_PlayerInfo
EP_CheckSegment
EP_Show
EP_Hide
EP_LocaleTable
EP_Helpnodename

```

```

EP_PlaySampleInit
EP_PlaySampleEnd
EP_Check7 ;for Formatloader (DTP_Check1 alike) a0=Formattags
EP_Check8 ;for Formatloader (DTP_Check2 alike) a0=Formattags
EP_SetPlayFrequency
EP_SamplePlayer

```

1.13 Eagleplayer support functions

Eagleplayer support Functions

Eagleplayer offers many functions to make adapting of players easier. A function callup is done in the following way:

```

move.l dtg_XXX(a5),a0 ;or respectively EPG_XXX(a5),a0
jsr (a0)

```

All following functions (except dtg_SongEnd and dtg_SetTimer) use d0/d1/a0/a1 as scratch registers. In a5 you should place the base address for all calls. (außer except from dtg_SongEnd and dtg_SetTimer)

At the moment you may use the following routines:
(API compatible to Delitracker 1.30)

dtg_GetListData

SYNOPSIS

```
memory size = dtg_GetListData(number)
a0      d0      d0.1
```

FUNCTION

Returns address and size of a loaded file

INPUTS

number - number of the file, starting with 0 for the 1st
file (selected by user)

RESULT

memory - a pointer to the start address of the file
in memory or NULL in case of an error
size - size of the file in Bytes or 0 in case of
an error

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 d0.1=0, sonst d0.1<>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.

extended Eagleplayer support functions

----- 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 eingestellten 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 ausgewertet 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 = Einsprungsadresse 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 abgeschlossen, 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
    Verzeichnis 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 Verzeichnis 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 Engines 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 auch auf andere Systeme übertragen werden. Es wird in den Samplennamen nach der Kennung "by" bzw. "#" gesucht. Der nächste String ist dann der Autorname. Der Autorname muß nicht copiert werden. Es reicht aus, wenn die Adresse des Autornamens in den ModuleInfo-Tag eingetragen wird.

Input: Arg1 = Start des 1. Samplennamens
Arg2 = Offset zum nächsten Sample
Arg3 = Länge des Samplennamens
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 Engines 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 Engines 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 systemkonform 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

1.14 Hints and Tips

Hints and Tips

The player should not change the LED condition because Eagleplayer will handle it.

This is a small list that you should match when you create your own player or custom module.

[] checkroutine exact enough and enforcer proof?

```

    (remember: more than 150 supported music file types
yet)
[ ] audiochannels allocated/freed correctly?
[ ] all allocated memory freed after playing?
[ ] all locks unlocked after playing?
[ ] enforcer and mungwall proof?
[ ] viable error handling path taken for all possible errors?
[ ] player tested under 2.0, 3.0, 3.1?
[ ] does the player work correct in all videomodes?
[ ] does the player return correct error codes?
[ ] the player only accesses memory belonging to it?

```

1.15 moduleinfo

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_Noisetrapper
- 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.

1.16 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_Structur 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 herausfinden, wann eine Note angespielt wird, so setzen Sie die Periode halt immer dann, wenn auf die Audio-Hardware zugegriffen 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?Samplnr - gibt die aktuelle Samplenummer an , noch von keiner Play-routine und keinem Engine unterstützt.

UPS_Voice?Sampletyp - 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.

1.17 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 Merzzellen 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 Merzzelle 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 Unterprogramme die Subroutinen nur nach einem USClass_LockEP benutzen.

Im folgenden werden die wichtigsten Eagleplayer-Unterprogramme erklärt:

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, dass 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.

1.18 Einbindung des Eagleplayers in andere Programme

Binding the Eagleplayer into other programs

The Eagleplayer offers the possibility to be called via its Engine port from an external source. To do this, you have to send it a "UM_Message" structure filled with certain values. An example for these opportunities is the Noise-converter which is distributed with this package.

What to do exactly ?

~~~~~

First find the Port:

```

move.l 4.w,a6
lea Portname(pc),a1 ;"EAGLEPLAYERPORT",0
jsr _LVOfindport(a6) ;find Eagleplayer Port
tst.l d0 ;test Result
beq.w .error ;Port not found
move.l d0,a4 ;save it

```

Second Create a Messageport (Under 1.3 you have to do this yourself)

```

move.l 4.w,a6
jsr _LVOcreatmsgport(A6)
move.l d0,d7 ;save it

```

Third you must get your own Task for getting the Message back

```

move.l 4.w,a6
suba.l A1,A1 ;Our Own Task
JSR _LVOfindtask(A6) ;find the Task
move.l d0,d6 ;save it

lea mymess(a5),a1 ;Pointer to UM_Message Structure
move.w #UM_sizeof-20,mn_length(a1) ;set the size of this Message struct
move.b #nt_message,ln_type(a1) ;Message-Type
move.l d7,mn_replyport(a1) ;Portadresse,an die
;zurückgesendet wird

move.w #-1,UM_UserNr(a1) ;everytime -1 !!!
move.l #USM_Externalprg,UM_Type(a1) ;Type of Usermessage: External Prg.
move.l d7,UM_Userport(a1) ;for Userprogram compatibility:
;Replyportaddress
move.l d6,UM_TaskAdr(a1) ;Save our Taskaddress

move.l #-1,UM_Signal(a1) ;No Signal
move.w #USClass_Command,UM_class(a1) ;Kind of Message: always a Command
move.l #0,UM_Userwindow(A1) ;Write here your Windows Address
;(for correct Requesterhandling)
;not required

move.l #UCM_Playmem,UM_Command(A1) ;Type of Command: here
;to play a certain Memory range

moveq #EPT_String+30,d0 ;allocate extra command structure
movem.l dl-a6,-(Sp) ;for submitting extra Informations
move.l 4.w,a6 ;like Name, Address, Size or
moveq #1,d1 ;other Infos depending on command
jsr _LVOallocmem(A6) (UCM...)
movem.l (sp)+,dl-a6 ;Memorysize=Structure+Stringsized
tst.l d0 ; =EPT_String+???
beq .Error2

move.l d0,a2

```

```

move.l a2,UM_ArgString(A1) ;Save this Structure into Message
move.l #EPT_String+30,EPT_Stringsize(A2) ;Save the Size, IMPORTANT
clr.l EPT_Next(A2) ;no next Text
move.l Sourceadr(a5),EPT_Result1(A2) ;Arg1, here Startaddress of
;mem to play
move.l SourceLen(a5),EPT_Result2(A2) ;Arg2, here Size of Mem to play

moveq #30,d1 ;copy a Name for this Memory Range
lea.l my_filename(a5),a3 ;e.g. "ripped using Eagleripper"
move.l a2,-(sp) ;not required
lea EPT_string(A2),a2
.copyfilename
move.b (a3)+,(A2)+
dbeq d1,.copyfilename

move.l (sp)+,a2

clr.l UM_Result(A1) ;no result !

move.l 4.w,a6
move.l a4,a0 ;EAGLEPLAYERPORT
jsr _LVOputmsg(a6) ;send the Message

.wait
move.l d7,a0
jsr _LVOwaitport(A6) ;wait for the Port

move.l d7,a0
jsr _LVOgetmsg(A6) ;get it back
tst.l d0
beq.s .wait
move.l d0,a1
cmp.l #USM_Externalprg,um_type(A1) ;our Message ?
beq.s .ok
jsr _LVOreplymsg(a6)
bra.s .wait
.ok
move.l UM_result(A1),d5 ;get Result (Errorcode: EPR_... or 0)

move.l UM_ArgString(A1),d0 ;get Reply String (Our String was freed,
;this is an other one we must free)
beq.s .notanswered
move.l d0,a1
move.l EPT_Result1(a1),temp1 ;Save Results
move.l EPT_Result2(a1),temp2 ;Save Results
moveq #28,d1
lea EPT_String(a1),a3
lea Stringtemp(a5),a2 ;Copy result String
.copy
move.b (a3)+,(a2)+
dbeq d1,.copy

move.l a1,d6 ;Free String-Structure(s)
.clrnext
move.l d6,a3
move.l EPT_Stringsize(A3),d0

```

---

```
move.l a3,a1
move.l EPT_Next(A3),d6
move.l 4.w,a6
jsr _LVOfreemem(A6)
tst.l d6
bne.s .clrnxt
.notanswered
move.l d7,a0
jsr _LVODeleteMsgPort(a6) ;delete our Messageport
rts
.error
```

## 1.19 The Engines

Why are there no Engine programming documentations yet ?

-----

Our big problem is the complexity of our engine-port, even since Version 1.0. Just to mention external user interfaces, the playerloader, manager, formatloader samplesaver...

Who really wants to program an engine for Eagleplayer should contact us and we'll try to find a solution.

---