Libraries.doc

Thomas Neumann

COLLABORATORS			
	TITLE : Libraries.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Neumann	January 19, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries.doc

1.1 main

```
*****************************************************************************
*                                                                           *
*           How to make External Libraries to AccessiblePlayer              *
*                                                                           *
*                           Update 29-05-1995                               *
*                                                                           *
*****************************************************************************

-----------------------------------------------------------------------------

                                INTRODUCTION
                                ------------
```

All the external players and noteplayers are built like a library. The players should be stored in the LIBS:APlayer/ directory and the noteplayers in the LIBS:APlayer/NotePlayer/ directory. There are only one function in the library, and this is a very simple one. The only thing it should do, is to return a pointer in A0 to a taglist.

A taglist is a list which contains some parameters, that will indicate what this player supports. There are a lots of tags, where the data field should point to a function, which has to do something e.g. a test function. All your functions will be called with a pointer to the AccessiblePlayer global data area in A5 (see below).

Remember when you code the different functions, you have to save all registers, also D0/D1/A0/A1.

Note also that the library name (without the ap-/an- and the -.library extension) must have a maximum length of 26 characters!!!!

If you want to load a config file or do something else, the first time the library is opened, you can make your code in the library init routine, just remember to free all allocations in the expunge routine.

```
                                    TAGS
                                    ----
```

Your taglist can contain the following tags. Note that you may NOT change the taglist, except the normal tags (TAG_SKIP, TAG_END etc.). If you want some changes, do it in another way. Because of this, I have made some tags pointing to a function instead of a pointer to some data. A good thing is to make your load, test and free memory routines independent of your other routines. If the user has double buffering turned on, your test, load and free code will be called while your play function still plays the previous module.

GLOBAL TAGS

APT_RequestVersion (UWORD)

This tag can be used, if the library uses some global functions which are implemented in a later version of AccessiblePlayer. The ti_Data field should contain the first version number of AccessiblePlayer where the new functions are implemented. The library will not be used, if it needs a newer version of AccessiblePlayer than the one which is currently in use.

APT_PlayerName/APT_NotePlayerName (APTR)

ti_Data should contain a pointer to the player/noteplayer name, like 'Protracker'. The string can max be 30 characters long. This tag must exist.

APT_Description (APTR)

ti_Data should contain a pointer to a description of the player/noteplayer. You can separate a new line with the ASCII code 10 (CR). The following rule should be used when you make the description:

1. The first line should contain the name of the programmer of the original player/noteplayer.

2. The second line should contain the name of the person who adapted this player. If you have made the player, skip this line.

3. The third line should be empty (It looks nicer that way)

4. Line 4-10 should contain a description of what the player/noteplayer can support and what it does.

Example:
```
                    +---------------------------------+
Line 1              |Original player by Lars Hamre.   |
Line 2              |Adapted & optimized by Tax.      |
Line 3 (Empty)      |                                 |
Line 4              |It can handle modules with       |
.                   |either 64 or 100 patterns.       |
.                   |                                 |
.                   |This player uses a NotePlayer.   |
.                   |                                 |
.                   |                                 |
Line 10             |                                 |
                    +--------------------------------+

(BOOL) APT_StartIRQ (FPTR)
--------------------------
You should only use this tag if you want to start your own IRQ. If you want
this  you  should  not  use the APT_Interrupt tag. If you use this tag in a
player,  you  will  in  A1  get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module.  You  have to return a boolean value in D1 that indicates a success
or  failure.  True(1)  means  success and false(0) means failure. You don't
have to return a value if you use this in a NotePlayer.

APT_StopIRQ (FPTR)
------------------
In  this function you have to stop your IRQ routine you have set up in your
APT_StartIRQ  function. If you use this tag in a player, you will in A1 get
the   address   returned   by   your   APT_LoadModule   function   if   supported,
otherwise it will be the start address of the module.

APT_Volume (BOOL)
-----------------
This  boolean  tag indicates that your player/noteplayer can support volume
changing.

APT_VolumeFunc (FPTR)
---------------------
In  some players/noteplayers you need to change the volume with a function,
because you can't get the global volume value within the interrupt routine.
You  can  then use this function to set the volume. It will be called every
time  the  user  change the volume slider or a new module is loaded. If you
use  this tag, you will not be able to support fade. If you use this tag in
a  player,  you  will in A1 get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module.

APT_ChangeChannel (FPTR)
------------------------
This  function  will  be  called  when  the user selects one of the channel
on/off  switches.  It  should  turn the channel on or off, depending on the
given  state.  In D1 (UBYTE) is the channel you have to change (0-3) and D2
(BOOL8)  the  state. True means on and false means off. If you use this tag
in a player, you will in A1 get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module.

APT_RealtimePlay (BOOL)
-----------------------
Use  this  tag  if you also support that the user can play a sampling while
your  player  plays  the  module. If you set this to true, AccessiblePlayer
will  call  your  APT_NewPlaySample  function when one or more channels are
turned off.

>>>>>>>>>> Tags in release 3 or higher (released as version 1.21) <<<<<<<<<

APT_CfgWindow (APTR)
--------------------
This tag should  point  to  a  pointer  that  points  to  a  config  window
```

structure. This will be used when the user presses the config button and your window is already open. APlayer needs the window handler from the structure to put your window to the front.

>>>>>>>>>> Tags in release 4 or higher (released as version 1.30) <<<<<<<<<

APT_NewConfig (APTR)

You should only support this tag if you have a config window in your library. The ti_Data field should point to two longwords. In the first longword you should store a pointer to your function. In the second longword there will be stored the global data pointer before your function will be started. Your function will be called when the user selects Config in the player preference window. You have to use the global data function to make your window, so it will get a standard. See in a later section about the standard and how to make your window.

APT_NewPlaySample (FPTR)

This function will be called when the user plays on the keyboard. You should play the selected sample. In D1 (WORD) you will get the period to play. In D2 (UBYTE) you will get the channel you have to play in (0-31). In A2 you will get a pointer to a Sample Info structure. Note that there are a global function in AccessiblePlayer that can help you to play the sample. If you use this tag in a player, you will in A1 get the address returned by your APT_LoadModule function if supported, otherwise it will be the start address of the module.

PLAYER TAGS

(ULONG) APT_EarlyCheck (FPTR)

If you use this tag, AccessiblePlayer will call the function via ti_Data before it has loaded the module. You can use this, if you don't need the whole module in memory before testing. Notice that this tag are mutual excluded with APT_Check. Your testing routine has to return a success flag in D0. 0 means that it can't recognise, 1 if everything went ok or 2 if there was an error. This tag or APT_Check must exist. This tag will also allow crunched files.

(ULONG) APT_Check (FPTR)

If you use this tag, AccessiblePlayer will call the function via ti_Data after it has loaded the module into memory. Only use this tag if you can't test before the whole file is in memory. Notice that this tag is mutual excluded with APT_EarlyCheck. You will get the start address in A1. Your testing routine has to return a success flag in D0. 0 means it can't recognise, 1 if everything went ok or 2 if there was an error. This tag or APT_EarlyCheck must exist.

(APTR) APT_LoadModule (FPTR)

You should only use this tag if you want to make your own loader routine. You can only use this tag if you have the APT_EarlyCheck tag. If you don't have this tag, AccessiblePlayer will load the whole module into memory. The

fileposition will always be zero when your function is called. Your function has to return an address in D0 if everything went ok, otherwise return 0 if some kind of DOS error occured, 1 for out of memory or 2 if another error occured. If you supply the return value 2, you must have the APT_GetError tag. You must have the APT_FreeModule tag if you use this tag.

APT_FreeModule (FPTR)

You must only use this tag if you use the APT_LoadModule tag. In this function you should free all memory you have allocated in the APT_LoadModule function. You will get the address returned by your APT_LoadModule function in A1. Note that this function should support a null pointer, which means do nothing.

(BOOL) APT_ExtLoad (FPTR)

Use this tag if you want to load more files than the current selected module. In A1 you will get the address returned by your APT_LoadModule function if supported, otherwise it will be the start address of the module. You have to return a success boolean in D1, true means that everything went ok and false means an error. You must have the APT_ExtFree and the APT_GetError tags if you use this tag.

APT_ExtFree (FPTR)

In this function you have to free all files loaded with the APT_ExtLoad function. In A1 you will get the address returned by your APT_LoadModule function if supported, otherwise it will be the start address of the module.

(APTR) APT_GetError (FPTR)

You only need this tag, if you supply an error number of 2 (another error) in your APT_LoadModule function or you have the APT_ExtLoad tag. You have to return a pointer in D0 to a null terminated error text.

(BOOL) APT_InitPlayer (FPTR)

This function should initialize your player routine. You have to allocate the audio channels in this function. It will only be called when a new module has been loaded into memory. In A1 you will get the address returned by your APT_LoadModule function if supported, otherwise it will be the start address of the module. You have to return a boolean value in D1 that indicates a success or failure. True means success and false means failure. This tag must exist.

APT_EndPlayer (FPTR)

This function will be called when a module is freed from memory. You should do some cleanup here, like free the audio channels. You will in A1 get the address returned by your APT_LoadModule function if supported, otherwise it will be the start address of the module. This tag must exist.

APT_InitSound (FPTR)

Here in this function you should initialize the module so it will start over with the tune number stored in APG_Tune in the AccessblePlayers global

```
data  area.  In A1 you will get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module. This tag must exist.

APT_EndSound (FPTR)
-------------------
This  function   should  only  clear  the  audio  channels  (if  not  using
noteplayer)  and  reset  variables if you have some. In A1 you will get the
address returned by your APT_LoadModule function if supported, otherwise it
will be the start address of the module. This tag must exist.

APT_Interrupt (FPTR)
--------------------
This function should be your interrupt routine. AccessiblePlayers interrupt
routine will generate a software interrupt pointing to your routine. If you
do  not  support  this  tag,  you  must  have  APT_StartIRQ and APT_StopIRQ
instead.  In  A1  you  will get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module. D1 (BOOL) will indicate that your routine was called from VBlank or
CIA. True means VBlank and false means CIA.

(APTR) APT_ModuleName (FPTR)
----------------------------
This  function  should return a pointer to the name of the module in A0. Do
only  support  this  tag  if  you can find the name. In A1 you will get the
address returned by your APT_LoadModule function if supported, otherwise it
will be the start address of the module.

(APTR) APT_Author (FPTR)
------------------------
This  function  should  return a pointer to the name of the author in D0 or
NULL  if you can’t find it. In A1 you will get the address returned by your
APT_LoadModule  function  if  supported,  otherwise  it  will  be the start
address of the module.

(APTR) APT_SubSong (FPTR)
-------------------------
This  function  should  return a pointer to two words in A0. The first word
should  be  the max number of tunes in the module. The second should be the
default  start tune number to play at start, where the first is 0. You will
in  A1  get  the  address  returned  by  your  APT_LoadModule  function  if
supported, otherwise it will be the start address of the module.

APT_Pause (BOOL)
----------------
This boolean tag indicates that your player can support pause.

(WORD) APT_GetMaxPattern (FPTR)
-------------------------------
This  function  should  return the max number of patterns which are used in
the  current  module. The result should be stored in D1. In A1 you will get
the   address   returned   by   your   APT_LoadModule   function   if   supported,
otherwise it will be the start address of the module.

(WORD) APT_GetMaxSample (FPTR)
------------------------------
This  function  should return the max number of samples used in the current
```

```
module  or  the  supported  number  which the player can handle. The result
should  be  stored  in  D1. In A1 you will get the address returned by your
APT_LoadModule  function  if  supported,  otherwise  it  will  be the start
address of the module.

(WORD) APT_GetSongLength (FPTR)
-------------------------------
You should return the length of the current tune in D1 in this function. In
A1  you  will  get  the address returned by your APT_LoadModule function if
supported, otherwise it will be the start address of the module.

(WORD) APT_GetSongPos (FPTR)
----------------------------
This  function  should  return  the current song position in D1. The result
should  be  between  0  and  the max length-1 (0-x). In A1 you will get the
address returned by your APT_LoadModule function if supported, otherwise it
will be the start address of the module.

(WORD) APT_Rewind (FPTR)
------------------------
If  you  support  that the user can rewind the actual tune, you have to use
this tag. The ti_Data field should point to a function that rewind the tune
one  "pattern".  Note that you should not rewind if the postion is zero. In
A1  you  will  get  the address returned by your APT_LoadModule function if
supported, otherwise it will be the start address of the module. As result,
you have to return the new position in D1.

(WORD) APT_Forward (FPTR)
-------------------------
If  you  support that the user can forward the actual tune, you have to use
this  tag. The ti_Data field should point to a function that count the tune
one  "pattern" forward. You have to make a wrap around, that means when you
get  to  the  end, you have to start over again with the counter. In A1 you
will get the address returned by your APT_LoadModule function if supported,
otherwise  it  will be the start address of the module. As result, you have
to return the new position in D1.

(BOOL) APT_TestNextLine (FPTR)
------------------------------
This  function has to test if the player has moved to the next pattern line
and return true or false in D1 depending if it has or not. This function is
only used in the fade routine in AccessiblePlayer, so if you do not support
volume,  you  should  not  support  this.  In  A1  you will get the address
returned by your APT_LoadModule function if supported, otherwise it will be
the start address of the module.

APT_GetSampleInfo (FPTR)
------------------------
This  function  should  fill out the a SampleInfo structure. In A1 you will
get  the  address  returned  by  your APT_LoadModule function if supported,
otherwise  it  will  be the start address of the module. In A2 you will get
the  start address of the structure you have to fill. In D1 (WORD) you will
get  the  sample number AccessiblePlayer want information about. The number
is  between  0  and the max number of samples-1 (0-x). See the include file
for more information about the structure.

APT_CallBack (FPTR)
```

```
-------------------
This  function  will  only  be  called  if  you  send a CallBack message to
AccessiblePlayer.  This  can  be  used  if  you want the main program to do
something  you  can't  do  in  an interrupt. Note that, if the user is in a
filerequester  or  the  program is about to load, this function will not be
called  before  the program is finished with the job. If you want a task to
run on its own, you have to make a new task. In A1 you will get the address
returned by your APT_LoadModule function if supported, otherwise it will be
the start address of the module.

>>>>>>>>>> Tags in release 2 or higher (released as version 1.1) <<<<<<<<<<

APT_Flags (LONG)
----------------
This tag is used if you want to say some special things to APlayer. You can
use the flags defined below:

AF_AnyMem                      = Set this bit if the module  can  be loaded
                                 into  any  memory.  This  bit will only be
                                 used  if  you  not  have  your  own loader
                                 routine.  Do  not  set this bit if you use
                                 noteplayers.

AF_UseAudio                    = If you  set  this bit, then the user can't
                                 override  the  allocation   of  the  audio
                                 channels.

>>>>>>>>> Flags in release 3 or higher (released as version 1.21) <<<<<<<<<

AF_SongEnd                     = Do  only  set  this  bit  if  you  support
                                 SongEnd and NOT position in your player.

>>>>>>>>>> Tags in release 4 or higher (released as version 1.30) <<<<<<<<<

(UBYTE) APT_UsedChannels (FPTR)
-------------------------------
This  function should return the number of channels used in D1. You will in
A1 get the address  returned by  your APT_LoadModule function if supported,
otherwise it will be the start address of the module.

(UBYTE) APT_SamplesType (FPTR)
------------------------------
This function should return the type of samples in the module in D1. If you
don't  have  this  tag, signed will be taken as default. You will in A1 get
the   address   returned   by   your   APT_LoadModule   function   if   supported,
otherwise it will be the start address of the module.

(APTR) APT_NotePlayer (FPTR)
----------------------------
If your player uses a noteplayer, you have to use this tag. It should point
to  a function that returns a pointer in A0 to a table with a length of max
12  bytes.  In  A1 you will get the address returned by your APT_LoadModule
function  if  supported,  otherwise  it  will  be  the start address of the
module.  The  first  word in the table is a flag word. See below for futher
description. The second byte indicates how many channels you have to use to
play  the  current module. The rest of the table is a little table of which
sample bit length this player can give the noteplayer. It ends with a zero.
```

A little ex. of a table: 0,4,8,0. It only uses 4 channels and there is only 8 bit samples. You can set these flags:

ANF_HardwareVolume	= Set this bit if your player only have 4 volumes and more channels. This is like Oktalyzer, Octamed etc.
ANF_Signed	= Set this bit if the samples can be signed.
ANF_Unsigned	= Set this bit if the samples can be unsigned.
ANF_Clock	= If your player have another clock frequency to the periods, you have to set this bit.

(APTR) APT_DefaultPlayerInfo (FPTR)

If your player uses a noteplayer, you have to use this tag. It is the same as the APT_NotePlayer tag except that the APT_NotePlayer will be used when APlayer tries to find a noteplayer to use after it has loaded a module. This tag will be used to get the default information. It will probaly be the same information except that the maximum channels in this function should be the max number this player can use and in the APT_NotePlayer tag the maximum channels should be the number of channels the current module use.

NOTEPLAYER TAGS

APT_NotePlayerInfo (APTR)

This tag should point to a little table with a length of max 12 bytes. The first word is a flag word. See below for a description of the bits. The next byte is the max number of channels this noteplayer supports. The rest is a little table of which sample bit length it supports. It should end with a zero. Currently there are these bits you can set in the flag word:

ANF_ChipMem	= If this bit is set, the noteplayer can play samples from chip memory.
ANF_FastMem	= If this bit is set, the noteplayer can play samples from fast memory.
ANF_HardwareVolume	= Set this bit if your noteplayer only have 4 volumes and you support more channels. This is like Oktalyzer, Octamed etc.
ANF_Signed	= Set this bit if you support signed samples.
ANF_Unsigned	= Set this bit if you support unsigned samples.
ANF_Clock	= If your NotePlayer can handle different clock frequencies for the periods, set

this bit.

(BOOL) APT_InitNotePlayer (FPTR)

This function should initialize your noteplayer routine. It will only be called when a new module has been loaded into memory. In D1 (UWORD) you will get the number of channels the player want. In D2 (UBYTE) you will get the samples type. You have to return a boolean value in D1 that indicates a success or failure. True means success and false means failure.

APT_EndNotePlayer (FPTR)

This function will be called when a module is freed from memory. You should do some cleanup here.

APT_InitNotePlayerSound (FPTR)

In this function you have to initialize the audio hardware. It will be called the first time a module is loaded or every time the user starts the module over again. This tag must exist.

APT_EndNotePlayerSound (FPTR)

This function should only clear the audio channels and reset variables if you have some. This tag must exist.

APT_PlayNote (FPTR)

This function should be your routine that will setup the audio hardware. AccessiblePlayer will generate a software interrupt pointing to your routine. It should get the channel information from the global channel tables and feed the hardware with the information. If it's a noteplayer that support more than 4 channels, it also have to do the mixing here.

APT_VirtualChangeChannel (FPTR)

Use this if your noteplayer have more than 4 channels and you can turn them off impendently of each other. In D1 (UBYTE) is the channel you have to change (0-31) and D2 (BOOL8) the state. True means on and false means off.

Global Data Area

All of your functions will be called with a pointer to AccessiblePlayers global data area in A5. In this area there is a lot of internal functions and data that will make it easier for you to implement a new player. In this section I will describe the functions and data in the AccessiblePlayer and which parameters they uses. The normal procedure on how to call an external function, is to use the following code segment:

```
move.l  APG_xxxxx(a5),a4
jsr     (a4)
```

Data

```
APG_FileSize (ULONG)
--------------------
In this longword the length of the module which is being loaded is stored.

APG_Tune (UWORD)
----------------
In this word the current tune number starting with 0 is stored.

APG_MaxVolume (UBYTE)
---------------------
Right  here  the maximum volume which your player may use (the volumeslider
position), if you support volume changing, is stored.

APG_Tempo (UBYTE)
-----------------
The  current  CIA  tempo  is  stored  here.  The  tempo  is  the same as in
Protracker, that means it can be between 32 and 255.

>>>>>>>>>> Data in release 4 or higher (released as version 1.30) <<<<<<<<<

APG_IntBase (APTR)
------------------
This is the Intuition.library base address.

APG_GfxBase (APTR)
------------------
This is the Graphics.library base address.

APG_UtiBase (APTR)
------------------
This is the Utility.library base address.

APG_ReqBase (APTR)
------------------
This is the Reqtools.library base address.

APG_Clock (ULONG)
-----------------
In  this  field the clock will be stored. If a player doesn't change it, it
will be 3546895 as default.

APG_MixingRate (ULONG)
----------------------
This  is the value that will be printed in the sample info window under the
"used mixing rate" line.  You can change this if you do  some mixing to the
mixing rate you use. This is probaly done in the NotePlayer.

APG_SampleInfo (APTR)
---------------------
This is a pointer to a linked SampleInfo structure list. If this pointer is
NULL,  there  isn't  any  sample  list. The first longword in the list is a
pointer  to  the next sample info structure. If this pointer is NULL, there
isn't  more  structures.  The rest is the structure itself. See the include
file for more information.

APG_NullSample (APTR)
```

```
---------------------
This is a pointer to a null word in chip memory.

APG_ChannelInfo (APTR)
----------------------
This will point to 32 channel info structures. These structures are used in
noteplayers  to get the informations about the sample it has to play. There
are one structure for each channel (max 32). See the include file for which
information there is in the structures.

APG_MaxChannels (UWORD)
-----------------------
This will indicate the max number of channels there is in each speaker.  If
this is zero,  the  NotePlayer  should calculate this value by itself if it
has to use it,  else it just use this value.  The most times,  this will be
zero, which means  the  half  number  of used channels is the max number of
channels in each speaker.


                                 Functions
                                 ---------

APG_AllocMem
------------

SYNOPSIS
        adr = APG_AllocMem (len, requirements)
        D0                  D0        D1

        APTR APG_AllocMem (ULONG, ULONG);

FUNCTION
        This  function  will  allocate  some  memory with the Len number of
        bytes.  If  you  use  this function, you have to use APG_FreeMem to
        free the memory again.

INPUTS
        len - number of bytes to allocate.

        requirements - the same as with exec's AllocMem() function.

OUTPUTS
        adr - the allocated address or null if the allocation failed.


APG_FreeMem
-----------

SYNOPSIS
        APG_FreeMem (adr)
                     A1

        void APG_FreeMem (APTR);

FUNCTION
        This  function  will  free  the  memory you have allocated with the
        APG_AllocMem function. Do not use this function to free some memory
        you haven't allocated with the above functions. You can pass a NULL
        to this function.

INPUTS
        adr - the address returned from APG_AllocMem.


APG_GetFilename
---------------

SYNOPSIS
        APG_GetFilename (buffer)
                         A0

        void APG_GetFilename (APTR);

FUNCTION
        This  function will copy the filename with path of the module which
        are  being loaded to the buffer given. This buffer must be at least
        be 2*108 bytes long.

INPUTS
        buffer - is  a  pointer  to  the buffer where you want the filename
                 with path to be placed. The name will be NULL terminated.


APG_FindName
------------

SYNOPSIS
        name = APG_FindName (path)
         A0                   A0

        APTR APG_FindName (APTR);

FUNCTION
        This  function  will scan the string Path after a filename and then
        return a new pointer in the string where the filename start.

INPUTS
        path - a  pointer  to  a  NULL  terminated  string  with  a  path &
               filename.

OUTPUTS
        name - a new pointer in the string where the filename starts.


APG_CheckLoad
-------------

SYNOPSIS
        success = APG_CheckLoad (start, len, adr)
           D0                     D1    D2   A0

         LONG APG_CheckLoad (LONG, LONG, APTR);

FUNCTION
```

```
        You  can  use  this function in your EarlyCheck function. This will
        load  Len bytes from the Start into your buffer starting at address
        Adr. Note that this function will NOT decrunch.

INPUTS
        start - this is the start in bytes, where you want to check from.

        len   - this is the length in bytes you want to read.

        adr   - this  is a pointer to your buffer where you want the readed
                data to be stored.

OUTPUTS
        success - if  this  is  zero,  it  means that an error has occured,
                  otherwise it will contain a nonzero value.


APG_PartialLoad
---------------

SYNOPSIS
        success = APG_PartialLoad (len, adr)
           D0                      D1   A0

         LONG APG_PartialLoad (LONG, APTR);

FUNCTION
        You  can  use  this function in your LoadModule function. This will
        load  Len  bytes  from  the  current  filepostion  into your buffer
        starting at address Adr. Note that this function will NOT decrunch.

INPUTS
        len   - this is the length in bytes you want to read.

        adr   - this is a pointer  to  your buffer where you want the read
                data to be stored.

OUTPUTS
        success - if  this  is  zero,  it  means that an error has occured,
                  otherwise it will contain a nonzero value.


APG_Load
--------

SYNOPSIS
        adr = APG_Load (name, type)
        D0              A0    D1

        APTR APG_Load (APTR, BOOL);

FUNCTION
        This  function  will  decrunch  the  file  and  load  it  into some
        allocated  memory.  When  you  want to free the memory allocated by
        this function, you must use the APG_FreeMem function.

INPUTS
```

```
        name - a pointer to the filename you want to load.

        type - which  memory type you want to allocate. True means chip and
               false means public.

OUTPUTS
        adr - is the address where the file is loaded or zero for an error.
              The allocated memory will automatically be freed if the error
              is a load error.


APG_DupOpen
-----------

SYNOPSIS
        fh = APG_DupOpen ()
        D0

        BPTR APG_DupOpen (void);

FUNCTION
        If  you  want  to use the file AFTER the load function, you have to
        call this function. It will open the file again, which will prevent
        a deletion of the temp file, if the original file was crunched. You
        must call DupClose to close the file again.

OUTPUTS
        fh - a new filehandler to the file or null for an error.


APG_DupClose
------------

SYNOPSIS
        APG_DupClose (fh)
                      D0

        void APG_DupClose (BPTR);

FUNCTION
        Use this function to close a file opened with the DupOpen function.
        It  will  close  the  file and delete the temp file. You can pass a
        null to this function.

INPUTS
        fh - the filehandler from the DupOpen function.


APG_Seek
--------

SYNOPSIS
        APG_Seek (pos)
                  D2

        void APG_Seek (ULONG);
```

```
FUNCTION
        This function will change the fileposition to the position Pos from
        the beginning of the file which is about to be loaded.

INPUTS
        pos - the new fileposition.


APG_CalcVolume
--------------

SYNOPSIS
        newvol = APG_CalcVolume (vol)
          D0                     D0

        UWORD APG_CalcVolume (UBYTE);

FUNCTION
        You  can use  this function  if you want to calculate a new volume.
        This  is  very  useful,  because if you support volume changing you
        just have to call this function before you store the  volume in the
        hardware  register  and  then  you  will  get a new volume which is
        calculated relatively to the volume which the user has chosen. This
        function is safe to call from interrupts.

INPUTS
        vol - the volume you want.

OUTPUTS
        newvol - the new volume you have to use.


APG_WaitDMA
-----------

SYNOPSIS
        APG_WaitDMA ()

        void APG_WaitDMA (void);

FUNCTION
        This function will wait enough time for the audio DMA to set up the
        hardware.  Use  this  instead  of  using raster wait or DBRAs. This
        function is safe to call from interrupts.


APG_SendMsg
-----------

SYNOPSIS
        APG_SendMsg (msg)
                     D2

        void APG_SendMsg (UWORD);

FUNCTION
        You  have  to  use  this  function if you want to send a message to
        AccessiblePlayer.  Such  a  message  could  be  a  NextModule  or a
        NextPosition  message.  See  the include file for a list of all the
        messages and the values you can send. This function is safe to call
        from interrupts.

INPUTS
        msg - the message you want to send.


APG_SetTimer
------------

SYNOPSIS
        APG_SetTimer ()

        void APG_SetTimer (void);

FUNCTION
        This  function  will  set  the  CIA  timer  to  the tempo stored in
        APG_Tempo  field in the global data area. This is safe to call from
        interrupts.


APG_NewProcess
--------------

SYNOPSIS
        process=APG_NewProcess (tags)
          D0                    A0

        APTR APG_NewProcess (APTR);

FUNCTION
        This   function   will  make  a  new  process.  It  will  call  the
        CreateNewProcess() function in the dos.library. See docs about this
        function for understanding.

INPUTS
        tags - a pointer to a tag list.

OUTPUTS
        process - the created process or null for an error.


APG_OpenWindow
--------------

SYNOPSIS
        window=APG_OpenWindow (struct)
         D0                     A0

        APTR APG_OpenWindow (APTR);

FUNCTION
        This function should only be used in your configuration routine. It
        will  open a window descriped in the structure given. See below and
        in the include file for more information. When you make your gadget
        structure,  you  should  always  count  the  gadget  ID  from 1 and
        upwards. Do never use gadget IDs 997-999, because they are reserved
        numbers.

INPUTS
        struct - a pointer to a structure describing the window.

OUTPUTS
        window - a private window handler structure or zero for an error.


APG_WaitMsg
-----------

SYNOPSIS
        msg=APG_WaitMsg (window)
        D0                A0

        APTR APG_WaitMsg (APTR);

FUNCTION
        This function will get your configuration task to  sleep  if  there
        aren't  any  message in the queue, else it will get the message and
        handle it if it's one of the  private  messages.  If  not  it  will
        return with a pointer to the message.

INPUTS
        window - a   pointer   to   a   window   handler  returned  by  the
                 APG_OpenWindow function.

OUTPUTS
        msg - a  pointer  to  the next message. This is a standard gadtools
              message.


APG_Reply
---------

SYNOPSIS
        APG_Reply (msg)
                   A0

        void APG_Reply (APTR);

FUNCTION
        This will reply the message returned by the APG_WaitMsg function.

INPUTS
        msg - a pointer to the message.


APG_ActivateGadget
------------------

SYNOPSIS
        APG_ActivateGadget (window, id)
                            A0      D0
        void APG_ActivateGadget (APTR, UWORD);

FUNCTION
        This will activate the gadget with the ID number. You  should  only
        call this function with a string or integer gadget.

INPUTS
        window - a   pointer   to   a  window  structure  returned  by  the
                 APG_OpenWindow function.

        id     - the gadget ID number.


APG_GetGadAdr
-------------

SYNOPSIS
        adr=APG_GetGadAdr (window, id)
        A0                   A0    D0

        APTR APG_GetGadAdr (APTR, UWORD);

FUNCTION
        This function  will  return  a  pointer  to  the  intuition  gadget
        structure  with  the gadget ID number. You can use this function if
        you want to use the structure by yourself, like when you should get
        the string from a string gadget.

INPUTS
        window - a   pointer   to   a  window  structure  returned  by  the
                 APG_OpenWindow function.

        id     - the gadget ID number.

OUTPUTS
        adr    - the start address to the gadget structure.


APG_Flash
---------

SYNOPSIS
        APG_Flash ()

        void APG_Flash (void);

FUNCTION
        This function will flash the screen.


APG_AllocChannels
-----------------

SYNOPSIS
        request=APG_AllocChannels ()
          D0
        APTR APG_AllocChannels (void);

FUNCTION
        You have to call this function in your APT_InitPlayer  function  to
        allocate  the  audio  channels.  It  will  try to allocate all four
        channels with priority 127, and if  it  succeeds  you  will  get  a
        pointer  to an IOAudio structure or a null for failure. You may not
        use this structure, you have to make a copy of it. Remember to call
        the  APG_FreeChannels  in  your APT_EndPlayer function when you are
        finished.

OUTPUTS
        request - a pointer to a IOAudio request or null for an error.


APG_FreeChannels
----------------

SYNOPSIS
        APG_FreeChannels ()

        void APG_FreeChannels (void);

FUNCTION
        This function will free the channels and  close  the  audio.device.
        You have to call this function in your APT_EndPlayer function.

>>>>>>>>>>>>>>>> Functions From Version 2 (Released as 1.1) <<<<<<<<<<<<<<<<

APG_CutSuffix
-------------

SYNOPSIS
        APG_CutSuffix (buffer)
                         A0

        void APG_CutSuffix (APTR);

FUNCTION
        This function will cut off a file extension from the file given  in
        buffer.

INPUTS
        buffer - a pointer to a buffer with the filename. This  buffer  has
                 to be at least 2*108 bytes long.

>>>>>>>>>>>>>>>> Functions From Version 3 (Released as 1.21) <<<<<<<<<<<<<<<

APG_OpenFile
------------

SYNOPSIS
        fh = APG_OpenFile (name)
        D0                  A0

        BPTR APG_OpenFile (APTR);
```

```
FUNCTION
        This will open the file with  the  name  "name".  If  the  file  is
        packed,  it  will  unpack it to a temp file and then open this file
        instead. Therefore you have to use the APG_CloseFile function.

INPUTS
        name - the name of the file you want to open (with path).

OUTPUTS
        fh   - the filehandler.


APG_CloseFile
-------------

SYNOPSIS
        APG_CloseFile (fh)
                       D0

        void APG_CloseFile (BPTR);

FUNCTION
        This will close the file you  have  opened  with  the  APG_OpenFile
        function.  If the opened file was packed, this function will delete
        the temp file again.

INPUTS
        fh - the filehandler returned by the APG_OpenFile function.


APG_FileRequester
-----------------

SYNOPSIS
        return = APG_FileRequester (file)
          D0                          A0

        ULONG APG_FileRequester (APTR);

FUNCTION
        This function will popup a filerequester where  the user can select
        one file.

INPUTS
        file   - a pointer  to a buffer  where you  want the  filename with
                 path  to be stored. This  buffer has  to be at least 2*108
                 bytes long.

        return - is the return value, where 0 means cancel and 1 means ok.


APG_DirRequester
----------------

SYNOPSIS
        return = APG_DirRequester (path)
```

```
          D0                        A0

        ULONG APG_DirRequester (APTR);

FUNCTION
        This function will popup a filerequester where  the user can select
        a path.

INPUTS
        path   - a  pointer  to a  buffer  where you  want  the path  to be
                 stored. This buffer has to be at least 2*108 bytes long.

        return - is the return value, where 0 means cancel and 1 means ok.


APG_UpdateGadgets
-----------------

SYNOPSIS
        APG_UpdateGadgets (window)
                           A0

        void APG_UpdateGadgets (APTR);

FUNCTION
        This function will activate the player configuration gadget  update
        routine.  You  can  use  this, if you for ex. have changed a string
        gadget and then want it updated.

INPUTS
        window - a  pointer  to  a   window  structure   returned  by   the
                 APG_OpenWindow function.


APG_CalcTempo
-------------

SYNOPSIS
        tempo = APG_CalcTempo (clock)
         D0                     D0

        UBYTE APG_CalcTempo (UWORD);

FUNCTION
        If you have a timer clock, like 14187 (PAL), and you want a normal
        BPM tempo, you  can  use this function to calculate it. If we take
        the above example, it  will return 125 (1773447/14187 (PAL)). This
        function is safe to call from interrupt.

INPUTS
        clock - the CIA timer clock.

OUTPUTS
        tempo - the calculated tempo.

>>>>>>>>>>>>>>>> Functions From Version 4 (Released as 1.30) <<<<<<<<<<<<<<<
```

```
APG_NewPlaySample
-----------------

SYNOPSIS
        APG_NewPlaySample (period, channel, si)
                           D1       D2     A2

        void APG_NewPlaySample (UWORD, UBYTE, APTR);

FUNCTION
        This  function is very useful. It will play the sample which is set
        up  in  the SampleInfo structure. It will setup the volume, looping
        etc.  See  include  file  for more information about the SampleInfo
        structure.

INPUTS
        period  - this is the period the sample has to be played with.

        channel - this  is the channel the sample should be played in. This
                  can be a number between 0 and 3.

        si      - this is a pointer to a Sample Info structure.


APG_NotePlayer
--------------

SYNOPSIS
        APG_NotePlayer ()

        void APG_NotePlayer (void);

FUNCTION
        You  should  only use this function in your player routine when you
        using  noteplayers.  This  call  should  be done at the end of your
        player  routine.  It  will  call  the  noteplayer in a new software
        interrupt.


APG_GetMaxVolume
----------------

SYNOPSIS
        vol = APG_GetMaxVolume ()
         D0

        UWORD APG_GetMaxVolume (void);

FUNCTION
        This function will return the maximum volume you have to play with.
        For the most times,  it  will  just  return  the  same  value as in
        APG_MaxVolume global data register, but if the module is fading, it
        will return the current  fading  volume.  This function is probally
        used in NotePlayers. This function is safe to call from interrupts.

OUTPUTS
        vol - the maximum volume.
```

Configuration of libraries

In this section I will explain how to make your configuration window and how to handle messages etc. First you have to make your own loader routine in the library INIT function. This loader should just load the configuration file from the "ENV:APlayer/Players/" or the "ENV:APlayer/NotePlayers/" directory. The filename should be the players/noteplayers name with a ".cfg" extension. Then you make the player/noteplayer as always, but you should also implement the APT_NewConfig and APT_CfgWindow tags in your tag list. See above for further explanation of these tags.

When the user selects the config gadget in the player window, your config routine will be started as a new process with the players/noteplayers name (starting with an "ap" or "an" prefix). Therefore you have to exit with a zero in D0 and a RTS command. After some initializing which may not take too long, you have to call the global function APG_OpenWindow. This will open a window centered on the screen with the size etc. you have given. It will also make a default menu which the user can use. This menu will be handled by AccessiblePlayer, so you don’t have to worry about that. The only thing you should handle, is the gadgets you have set as extra gadgets. The default gadgets (Save, Use & Cancel) will also be handled by AccessiblePlayer. It will save the configuration as raw data.

After you have called the APG_OpenWindow function, you have to start a loop where you call APG_WaitMsg. This function will get the task to sleep if there aren’t any messages. If there is a message, it will test to see it’s one of the private messages, like a menu selection. If so, they will be handled and your task will go to sleep again. If it isn’t one of the private messages, it will return a pointer to the message. After you have got the values you need, you have to reply the message with the APG_Reply function. If the user have selected the save, use or cancel gadget, you will get a zero as message pointer. Then you have to exit your task with a moveq #0,d0 and a RTS. You don’t have to close your window, this will be done by AccessiblePlayer. If you use the Exit pointer in the structure, AccessiblePlayer will call this function before it will save the configuration. In this function you have to get the values from your string or integer gadgets.