# Libraries.doc

Thomas Neumann

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Libraries.doc | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Thomas Neumann | January 18, 2023 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# Libraries.doc

## 1.1 main

```
***************************************************************************
*                                                                         *
*           How to make External Libraries to AccessiblePlayer           *
*                                                                         *
*                            Update 27-12-1995                            *
*                                                                         *
***************************************************************************

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

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

All the external players, noteplayers and agents are built like a library. The players should be stored in the LIBS:APlayer/ directory, the noteplayers in the LIBS:APlayer/NotePlayer/ directory and the agents in the LIBS:APlayer/Agents/ 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-/aa- 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.

Players:
--------

A player is a program who can determine a certain module format and play it. It can use a NotePlayer to play the sounds.

NotePlayers:
------------
A noteplayer is a program which feed the hardware with the sounds. It can mix the samples to get more channels, but it’s not necessary. If a player uses a noteplayer, it will fill out a noteplayer structure (See the include file for more information). The noteplayer will then read these information and feed the hardware. Here is a little example: If the player want to change the volume, it sets the volume in the volume field in the noteplayer structure. The player will also set the volume bit in the flag field. The noteplayer will then know it has to change the volume. Remember that the noteplayer is responsible to clear the flag field after reading.

Agents:
-------
Agents is a program which can be started when the user select different things. All agents will get a pointer to some data. The data is different for each agent type.

## 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/APT_AgentName (APTR)
------------------------------------------------------
ti_Data should contain a pointer to the Player/NotePlayer/Agent 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/Agent. 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/Agent.

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/Agent 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             |                                 |
                    +---------------------------------+
```

>>>>>>>>>> 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.

>>>>>>>>>> Tags in release 5 or higher (released as version 1.40) <<<<<<<<<

APT_Show (APTR)
---------------
Use only this tag if you have a window that is NOT a config window. This tag should point to two longwords. In the first longword you should have 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 Show in the player preference window.

```
You can use the global data function to make your window if you want.

APT_ShowWindow (APTR)
---------------------
This  tag  should  point  to  a  pointer  that  points  to  a  show  window
structure.  This  will  be used when the user presses the show  button  and
your window is already open. APlayer needs  the  window  handler  from  the
structure to put your window to the front.


                          PLAYER/NOTEPLAYER TAGS
                          ----------------------

(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 4 or higher (released as version 1.30) <<<<<<<<<

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.

>>>>>>>>>> Tags in release 6 or higher (released as version 2.00) <<<<<<<<<

APT_ChanSignal (APTR)
---------------------
If  you  make a player and want to be signaled when a sample loops, you can
use  this  tag. It is the same as using an audio interrupt. It has to point
to  2 longwords. The first longword is a pointer to your player task (use a
FindTask()  call).  The second longword is the signal bit number (if bit 2,
store 1<<2 here). If you make a noteplayer, you have to check to see if the
task  longword  is  null and if it's, don't do anything. If it's non- null,
you  have to send the signal to the task when you looping the sampling. You
also  have to send a signal when you start a new sampling. Notice that, you
should only send a signal in your channel 0 routine.


                                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.

>>>>>>>>> Flags in release 6 or higher (released as version 2.00) <<<<<<<<<

AF_Sample                      = If  your  player  is  a sample player, you
                                 have  to set this flag. That means, if the
                                 user  don't  have  the  loop gadget on and
                                 when  APlayer  receives  a NextMod message
                                 from your player, it will stop your player
                                 before it loads the next module.

>>>>>>>>>> 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.

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

APT_TempoFunc (FPTR)
--------------------

If you want to have your own tempo change routine, you can use this tag. Your function will be called every time the user change the tempo. 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. In D1 (WORD) you will get the value the tempo slider is on. This value can be between -111 to 112.

## 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) (NOT USED YET!!!!)
--------------------------------------------------
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.


                                AGENT TAGS
                                ----------

APT_AgentType (UWORD)
---------------------
This  tag   will  tell   APlayer  which  type  of  Agent   this  agent  is.
Currently there is these types:

AGNT_SampleSaver: These  type  will  be  called  when  the user will save a
                  sample in the sample window.

AGNT_Scope      : If you want to make a scope, use this type.  You have  to
                  make  your  own  process which will allocate some signals
                  (see  functions  description  below).  It's important you
                  only  run  at  priority  -25.  The  sample start address,
                  length  etc.  can  you  read  from  the same structure as
                  noteplayers  read  from. Notice that the noteplayer flags
                  is  in  another structure. This is because the noteplayer
                  will  erase  the  flags before your scope routine will be
                  called.

AGNT_FSS        : This agent type is private. Do not use it.

APT_AgentHandler (FPTR)
-----------------------
This  tag  should  point  to  a function which will be called when the user
activate  it.  How the user activate it is different from type to type. You
will  in  A1  get  a  pointer  to  a data structure. This structure is also
different  from  agent type to type. See the include file for the different
structures.

>>>>>>>>>> Tags in release 6 or higher (released as version 2.00) <<<<<<<<<

APT_OSVersion (UWORD)
---------------------
If  your  agent need an OS version bigger than version 37, you can use this
tag. The ti_data field is the OS version number you need.


                             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.

>>>>>>>>>> Data in release 6 or higher (released as version 2.00) <<<<<<<<<

APG_GadBase (APTR)
------------------
This is the Gadtools.library base address.

APG_ChannelFlags (APTR)
-----------------------
This  is  a  pointer to a structure where all the flags from the noteplayer
structure are stored. This will probally only be used in scope agents.

APG_ListFont (APTR)
-------------------
This  is  a  pointer  to  a  TextAttr  structure for the font to be used in
listviews. If you have any listviews, you should use the NLV_Font tag which
points to the same pointer as this does.

APG_GeneralFont (APTR)
----------------------
This  is  also a pointer to a TextAttr structure, but it points to the font
the user has selected as general font.

APG_LoopFlag (BOOL)
-------------------
In this 16 bit boolean there will be stored the state of the loop gadget in
the main window. True means loop is on, false is off.


                                 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  structure  returned  by  the
                 APG_OpenWindow or APG_OpenShowWindow 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 or APG_OpenShowWindow 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 or APG_OpenShowWindow 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, title)
          D0                         A0     A1

        ULONG APG_FileRequester (APTR, 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.

        title  - is a pointer to a NULL terminated string.

OUTPUTS
        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.

OUTPUTS
        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 or APG_OpenShowWindow 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.

>>>>>>>>>>>>>>> Functions From Version 5 (Released as 1.40) <<<<<<<<<<<<<<<

APG_OpenShowWindow
------------------

SYNOPSIS
        window = APG_OpenShowWindow (struct)
         D0                           A0

        APTR APG_OpenShowWindow (APTR);

FUNCTION
        This  function  should only be used in your show window 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.
```

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

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


APG_CloseWindow
---------------

SYNOPSIS
        APG_CloseWindow (window)
                         A0

        void APG_CloseWindow (APTR);

FUNCTION
        Do only use this function if you want to close the window opened by
        the APG_OpenShowWindow or APG_OpenWindow functions by yourself.

INPUTS
        window - the  window  handler returned by the APG_OpenShowWindow or
                 APG_OpenWindow.


APG_Sleep
---------

SYNOPSIS
        APG_Sleep (window)
                   A0

        void APG_Sleep (APTR);

FUNCTION
        If  you want to make a sleep pointer to your window you have opened
        with  the  APG_OpenShowWindow  or APG_OpenWindow functions, you can
        use this function. It will keep track of how many "sleeps" you have
        made,  but  remember  for  each  APG_Sleep call, you have to make a
        APG_Unsleep call.

INPUTS
        window - the  window  handler returned by the APG_OpenShowWindow or
                 APG_OpenWindow.


APG_Unsleep
-----------

SYNOPSIS
        APG_Unsleep (window)
                     A0

        void APG_Unsleep (APTR);

FUNCTION
        This function will unsleep your window again.
```

```
INPUTS
        window - the  window  handler returned by the APG_OpenShowWindow or
                 APG_OpenWindow.


APG_LVSetAttrs
--------------

SYNOPSIS
        APG_LVSetAttrs (window, id, tags)
                          A0    D0   A1

        void APG_LVSetAttrs (APTR, UWORD, APTR);

FUNCTION
        If  you  want to set some attributes in a NEW listviews, you has to
        use  this  function. See the include file to see which tags you can
        use.

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

        id     - this is the gadget id number.

        tags   - this is a pointer to a taglist.


APG_LVGetAttrs
--------------

SYNOPSIS
        APG_LVGetAttrs (window, id, tags)
                          A0    D0   A1

        void APG_LVGetAttrs (APTR, UWORD, APTR);

FUNCTION
        This function will get some attributes from a NEW listview. See the
        include file to see which tags you can use.

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

        id     - this is the gadget id number.

        tags   - this is a pointer to a taglist.


APG_LVChangeColor
-----------------

SYNOPSIS
        APG_LVChangeColor (list, number, color)
                            A0     D0      D1
        void APG_LVChangeColor (APTR, UWORD, UBYTE);

FUNCTION
        If  you  want  to  change the color of some of the elements in your
        listview, you has to use this  function.  You  can  select  between
        dark, highlighted or no select.  No  select means that the user can
        select the node in the listview.

INPUTS
        list   - this is a pointer to your list.

        number - this is the element number in your list starting with 0.

        color  - this byte indicate which color you want to use.  0 is dark
                 (black), -1  is  highlighted (white)  and 1 is  no  select
                 (blue).


APG_CreateList
--------------

SYNOPSIS
        list = APG_CreateList ()
         D0

        APTR APG_CreateList (void);

FUNCTION
        This  function  will  create  a new empty exec list to use with the
        listviews.  You has to use the returned pointer in every other list
        or listview functions.

OUTPUTS
        list - is a pointer to the new created list or NULL for an error.


APG_RemoveList
--------------

SYNOPSIS
        APG_RemoveList (list)
                         A0

        void APG_RemoveList (APTR);

FUNCTION
        This  will  remove  the  list  from memory. Notice that it will NOT
        delete  any  elements  in  the list, you have to do that before you
        call this function.

INPUTS
        list - is  a  pointer  to  a  list returned by the APG_CreateList()
               function.


APG_AddNode
```

```
-----------

SYNOPSIS
        success, number = APG_AddNode (list, text, number, size)
          D0       D1                   A0    A1     D2     D4

        ULONG, ULONG APG_AddNode (APTR, APTR, ULONG, ULONG);

FUNCTION
        This  function  will  add  a node to the list. You can select which
        number  in  the  list  you will insert the new node at. If you just
        want to insert the new node at the bottom in the list, just give -1
        as  the  number.  The size is how many bytes you will use after the
        exec Node structure. This can be the length of the text or whatever
        you  want.  Remember  that  the  text will be copied after the Node
        structure, so the size should be minimum the length of the text.

INPUTS
        list   - is  a  pointer  to a list returned by the APG_CreateList()
                 function.

        text   - is a pointer to a null terminated string.

        number - the number to insert the new node at or -1 for the bottom.

        size   - number of bytes to use as extra bytes in the list.

OUTPUTS
        success - 0 if an error occured.

        number  - the number in the list the new node was inserted at.


APG_DeleteNode
--------------

SYNOPSIS
        APG_DeleteNode (list, number)
                         A0     D2

        void APG_DeleteNode (APTR, ULONG);

FUNCTION
        This function will delete one node in the list. The "number" is the
        number in the list starting with 0.

INPUTS
        list   - is  a  pointer  to a list returned by the APG_CreateList()
                 function.

        number - the number in the list you want to delete.


APG_DeleteList
--------------

SYNOPSIS
```

```
        APG_DeleteList (list)
                        A0

        void APG_DeleteList (APTR);

FUNCTION
        This  will delete all the nodes in the list, but it will not remove
        the  list  from memory. If you want that, call the APG_RemoveList()
        function after this function.

INPUTS
        list - is  a  pointer  to  a  list returned by the APG_CreateList()
               function.


APG_CopyList
------------

SYNOPSIS
        newlist = APG_CopyList (list, size)
          D0                    A0    D0

        APTR APG_CopyList (APTR, ULONG);

FUNCTION
        If  you  want  a  copy of your list, you can use this function. The
        size is the same as in the APG_AddNode() function.

INPUTS
        list - is  a  pointer  to  a  list returned by the APG_CreateList()
               function.

        size - number of bytes to use as extra bytes in the list.

OUTPUTS
        newlist - is a pointer to the new list or null for an error.


APG_ExchangeNodes
-----------------

SYNOPSIS
        APG_ExchangeNodes (list, number1, number2)
                            A0     D0       D1

        void APG_ExchangeNodes (APTR, ULONG, ULONG);

FUNCTION
        This  function  will exchange two nodes in your list. If one of the
        numbers you have given can't be found, nothing will happend.

INPUTS
        list    - is  a  pointer to a list returned by the APG_CreateList()
                  function.

        number1 - a node number starting with 0.
        number2 - a node number starting with 0.


APG_MoveToTop
-------------

SYNOPSIS
        APG_MoveToTop (list, number)
                       A0     D0

        void APG_MoveToTop (APTR, ULONG);

FUNCTION
        This function will move a node to the top of the list.  If it can't
        find the node, nothing will happend.

INPUTS
        list   - is  a  pointer  to a list returned by the APG_CreateList()
                 function.

        number - is the node number starting with 0.


APG_MoveToBottom
----------------

SYNOPSIS
        APG_MoveToBottom (list, number)

        void APG_MoveToBottom (APTR, ULONG);

FUNCTION
        This  function  will  move  a node to the bottom of the list. If it
        can't find the node, nothing will happend.

INPUTS
        list   - is  a  pointer  to a list returned by the APG_CreateList()
                 function.

        number - is the node number starting with 0.


APG_FindNode
------------

SYNOPSIS
        success, node = APG_FindNode (list, number)
          D0      A0                   A2     D0

        ULONG, APTR APG_FindNode (APTR, ULONG);

FUNCTION
        This  function  will search the list after the node number you have
        given and if found, return a pointer to the node structure.

INPUTS
        list   - is  a  pointer  to a list returned by the APG_CreateList()
```

```
                  function.

        number - is the node number starting with 0.

OUTPUTS
        success - 0 if it couldn't find the node.

        node    - is a pointer to the node if it could find it.


APG_FindNodeNumber
------------------

SYNOPSIS
        number = APG_FindNodeNumber (list, text)
          D0                          A0    A1

        ULONG APG_FindNodeNumber (APTR, APTR);

FUNCTION
        This  function  will  return  the node number in the list where the
        "text" string is. If it can't find it, -1 will be returned.

INPUTS
        list - is  a  pointer  to  a  list returned by the APG_CreateList()
               function.

        text - is a pointer to a null terminated string.

OUTPUTS
        number - is the node number starting with 0.


APG_StringCompare
-----------------

SYNOPSIS
        result = APG_StringCompare (string1, string2)
          D0                          A0       A1

        UWORD APG_StringCompare (APTR, APTR);

FUNCTION
        If  you  want to compare two strings, this is the function for you.
        It  will compare them and return a value to tell if the strings are
        equal, bigger or less.

INPUTS
        string1 - is a pointer to a null terminated string.

        string2 - is a pointer to a null terminated string.

OUTPUTS
        result - is the result from the test. 0=S1=S2, 1=S1>S2, -1=S1<S2.

>>>>>>>>>>>>>>> Functions From Version 6 (Released as 2.00) <<<<<<<<<<<<<<<
```

```
APG_GetScreenHd
---------------

SYNOPSIS
        scrhd = APG_GetScreenHd ()
          D0

        APTR APG_GetScreenHd (void);

FUNCTION
        This  function  will  return  the screen handler where APlayer will
        open all the windows. This can be used if you want to open your own
        window.

OUTPUTS
        scrhd - is the screen handler.


APG_GetMemType
--------------

SYNOPSIS
        memtype = APG_GetMemType ()
          D0

        ULONG APG_GetMemType (void);

FUNCTION
        If your player uses a noteplayer, you can call this function to see
        what type of memory the found noteplayer can use.  The return value
        is the same as the "requirement" values to the AllocMem function.

OUTPUTS
        memtype - the memory type.


APG_AllocScopeSignal
--------------------

SYNOPSIS
        scope = APG_AllocScopeSignal ()
          D0

        APTR APG_AllocScopeSignal (void);

FUNCTION
        When  you  making  a scope agent,  you has to call this function to
        allocate  some signals for your task. This function will allocate a
        little  structure  and link it together with other structures. This
        is  done,  so APlayer knows which scope agents has the windows open
        and  send  a  signal  to  them. Your task just have to wait for the
        signals  bit.  When  your task  wake up from sleep, you can get the
        pointer from the APG_ChannelFlags pointer to get the flags for each
        channels.  This  table is 32 bytes long, one byte for each channel.
        These  bytes  are  the  same as the NPC_Flags byte in the NPChannel
        structure, which you can get from the APG_ChannelInfo pointer.
```

```
OUTPUTS
        scope - a pointer to a structure you have to use in other calls.


APG_FreeScopeSignal
-------------------

SYNOPSIS
        APG_FreeScopeSignal (scope)
                              A0

        void APG_FreeScopeSignal (APTR);

FUNCTION
        This  function  will  free  the  node  and  the  signals  which was
        allocated by the APG_AllocScopeSignal function.

INPUTS
        scope - a   pointer   to   the   structure   you   got   from   the
                APG_AllocScopeSignal.


APG_GetScopeSignal
------------------

SYNOPSIS
        signal = APG_GetScopeSignal (scope)
          D0                           A0

        ULONG APG_GetScopeSignal (APTR);

FUNCTION
        This  function  will return the signals to wait for. These bits can
        just  or'es  together  with  the  other  bits you want to wait for.
        Remember  you  also  have to wait for a CTRL-C signal, and when you
        get it, you has to do the same  as when the user closes the window.
        This is  because  if the user  has your  window open  and then quit
        APlayer, APlayer will then send a CTRL-C signal to all the tasks.

INPUTS
        scope - a   pointer   to   the   structure   you   got   from   the
                APG_AllocScopeSignal.

OUTPUTS
        signal - the signal bits you have to wait for.


APG_TestScopeSignal
-------------------

SYNOPSIS
        result = APG_TestScopeSignal (scope, signals)
          D0                            A0      D0

        UWORD APG_TestScopeSignal (APTR, ULONG);

FUNCTION
```

```
        After  the  wait you has to call this function with the result from
        execs Wait() function. This function will then test the signal bits
        to  see if it's a scope signal. It can return 3 different values. 0
        means  it  not  a scope signal, which means it can be a signal from
        your  window  or  a  CTRL-C  signal. If bit 0 is set, it's a CHANGE
        signal.  When you get this signal you have to call a routine, which
        get the values from the NPChannel structure and act on them. If bit
        1  is  set,  it's a VBLANK signal. When you get this signal you can
        call  a  routine, which do anything you want in a VBlank interrupt.
        This it's probally an update of the window. Notice that you can get
        both signals at the same time.

INPUTS
        scope - a   pointer   to   the   structure   you   got   from   the
                APG_AllocScopeSignal.

        signals - the result from the Wait() function.

OUTPUTS
        result - A bit result. See above for an description.


APG_DosError
------------

SYNOPSIS
        APG_DosError ()

        void APG_DosError (void);

FUNCTION
        This  function will show a dos error requester. This can be used if
        you got some error from a dos function  and want to show what  went
        wrong.


APG_ShowRequest
---------------

SYNOPSIS
        result = APG_ShowRequest (bodytxt, data, gadtxt)
          D0                        A0      A1     A2

        ULONG APG_ShowRequest (APTR, APTR, APTR);

FUNCTION
        This function will show a reqtools requester.

INPUTS
        bodytxt - this is a pointer to the text you want to show.

        data    - this is  a  pointer to a datastream to  use  with the "%"
                  codes.

        gadtxt  - this is a pointer to the text you want in the gadgets.
```

---

## 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/", "ENV:APlayer/NotePlayers/" or the "ENV:APlayer/Agents/" directory. The filename should be the players/noteplayers/agents name with a ".cfg" extension. Then you make the player/noteplayer/agent 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/agents name (starting with an "apc", "anc" or "aac" 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.

---

## How to make Show windows

In this section I will explain how to make your show window and how to handle messages etc. You should have the APT_Show and the APT_ShowWindow tags in your tag list. See above for further explanation of these tags.

When the user selects the show gadget in the player window, your show window routine will be started as a new process with the players/noteplayers/agents name (starting with an "aps", "ans" or "aas" 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_OpenShowWindow. This will open a window centered on the screen with the size etc. you have given. You have to handle your own gadgets.

After you have called the APG_OpenShowWindow 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 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 closed the window, you will get a zero as message pointer. Then you have to exit your task with a moveq #0,d0 and a RTS. If you use the Exit pointer, AccessiblePlayer will call this function before it closes the window.

Note that you can open more than one window if you like.

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

## NewListViews

In AccessiblePlayer, I have coded my own listview routines, so even on Kickstart 2.0 they will get Kickstart 3.0 look. I have also implemented a lot of extra features and more will come in the future. If you will use these listviews in your own windows, like in the config or the show window, you just set the type to NEWLISTVIEW_KIND, the flags and text to NULL and a pointer to a tag list. Some of the tags can only be used when you create the listview, others can both be used when you create it or when you want to set an attribute. See the include file to see which tags you can do what on. You need the following IDCMP in your window:

GadgetUp!GadgetDown!IntuiTicks!MouseButtons!MouseMove!RawKey

Here is a description of the different tags:

### GA_Disabled (BOOL)

If you want to disable your listview, use this tag. (Create Only)

### NLV_Labels (APTR)

This tag should point to a list with all your elements. The list is the same kind as in the normal gadtools listview, except that the first word right after the Node structure is a color flag. Do NOT change this manualy, use the APG_LVChangeColor() function instead. There is a lot of global functions you can use to generate your list. You can also set this to NULL if you don't want any elements in the list. This is the default value.

### NLV_Top (UWORD)

This will set the top of the list. The first element in the list is zero and the second is one etc. Default is zero.

### NLV_Selected (UWORD)

If you want to select an element, you can use this tag. If you want to unselect an element, set the number to -1. This is default. This tag will be ignored if your listview is read only.

NLV_SelectTwice (BOOL8)
-----------------------
You will only get one message each time an user press an element in the listview. If the user press the selected element you will NOT get a new message. If you want to get a message every time the user select an element, even if it's the selected one, set this to TRUE. This can be used, if you want to check for doubleclick. Default is FALSE. This tag will be ignored if your listview is read only.

NLV_CursorKeys (BOOL8)
----------------------
This tag is one of the most coder friendly ones :) If you set this tag to TRUE, AccessiblePlayer will automatic implement a cursor key routine. That means, the user can use the cursor keys to scroll up and down in the listview, without you do anything. See the guide to see how you can use the cursor keys. Default is FALSE. This tag will be ignored if your listview is read only.

NLV_ScrollWidth (UWORD)
-----------------------
If you want to change the width of the scroller bar, you can use this tag. The default is 16.

NLV_Tabs (APTR)
---------------
This is one of the heavy tags. You can use tabs in your elements to set the text in rows, even if you use a proportional fonts. See for example in the FSS window. Notice that, if you use this tag, the updating of the listview will be slower, because it has to clear small pieces for each element in the visible area in the listview.

This tag has to point to a table with a row description where the elements are pixel positions. You also have to use some special codes in your element text. You can use three codes, binary 1, 2 and 3.

3 means that you will change the style in the text. After the code you have two style numbers. This will be used as parameters to the SetSoftStyle() function in the graphics library. See the docs to this function for more information, but remember that the numbers you have should be one bigger than you will give. This is because you can give zero as parameter and if you don't add 1 to the number, Accessibleplayer will think that it's the end of the string.

1 means that you want the text that comes AFTER this code to be left centred.

2 means that the text which comes BEFORE will be right centred.

Notice that is only the text to the next code or to the end of the string that is influenced by the code. Now in your row destription table you have to tell Accessibleplayer where it has to set the rows. This table is very complex. I haven't made a table that works the first time yet, so read this every time you make a listview with tabs.

For every 1 code (left centre), the listview routines read one number in the table. This number is where to end this part of the text. It will start at the current position. If the text is longer, it will be clipped.

For every 2 code (right centre), the listview routines read two numbers in the table. The first number is where the text has to end and the second is where the next text string has to start. The text will start at the current position. If the text is longer, it will be clipped.

When you are finished with the table, you has to end it with a -1.

Did you get it? Well, we have to make an example:

In my FSS window I have the following element text:

```
dc.b    "%d.",2,"%s",1,"%u",2,0
```

Which will print the text as following:

```
 x. xxxxx          x
xx. xxxxxxxxx     xx
```

My row description table have the following numbers:

```
3*8,4*8,26*8,31*8,31*8,-1
```

Let’s get through it step by step. In my element text I have a number and the binary code 2. This means that I want the number to be right centred. The number has to start at position 0 and end at position 3*8 and the next string has to start at position 4*8.

The next part is a string with the binary code 1. It will start at position 4*8 (the current position) and end at position 26*8.

Now I have another number and the binary code 2. This will start at position 26*8 (the current position) and end at position 31*8. The next string has to start at position 31*8 (therefore the two same numbers).

Now our string is finished, at we have a -1 in our table, so it should working now. :)

NLV_Font (APTR)
---------------
With this tag you can change the font which will be used in the listview. It have to point to a TextAttr structure. If it can’t open the font, the Topaz 8 font will be used, which is also the default.

NLV_Function (FPTR)
-------------------
This tag should point to the function you want to be called when the user presses on an element. In your function you will in D3 get the node number in your list the user has pressed starting with 0. This tag will be ignored if your listview is read only.

NLV_MoveTop (BOOL)
------------------
This tag can only be used when you set the attributes. This is also one of the friendly ones. If you set this to TRUE and you change the selected element, the listview routines will check to see if the selected one is in

```
the  visible area. If it is, nothing will happend, but if it’s not, the top
will  be  set  to  the selected element. With this tag,  you will always be
sure that the user can see the  selected element.  This tag will be ignored
if your listview is read only.

>>>>>>>>>>>>>>>>>>> Tags From Version 6 (Released as 2.00) <<<<<<<<<<<<<<<<<

NLV_ReadOnly (BOOL)
-------------------
If you want your listview to be read only, you can use this tag.  The bevel
box will be recessed and the user can’t select any item in  the  list.  The
default is FALSE.
```