Sound

COLLABORATORS			
	TITLE : Sound		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 19, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sound

1.1 Sound V1.00

```
PureBasic - Sound V1.00

  Managing sounds will never be so easy and so fast
  than with this library. Believe us, we push the
  Amiga hardware to the max, to allow system compliant
  but instant sound replay. It simply use the 4 stereo
  audio channels for efficient output. The sound format
  is the IFF/8SVX, standard of the Amiga.

Commands summary:

   ChangeSoundPeriod
   ChangeSoundVolume
   CreateSound
   DecodeSound
   FreeSound
   GetSoundLength
   InitSound
   LoadSound
   PeekSoundData
   PlaySound
   PokeSoundData
   SaveSound
   SetSoundChannels
   SetSoundPeriod
   SetSoundVolume
   SoundFilter
   StopSound
   UseSoundChannels

   Sound Demo
```

1.2 changesoundperiod

```
  SYNTAX
```

```
Result.w = ChangeSoundPeriod(#Obj.w,Period.w)

  FUNCTION
Use this function to change the period when the sound
is actually played.

The Period set in the Sound Object, that is used when
the sound is started, is not affected with this call.

#Obj
The Object to change period for.

Period
This is the new period value.

Result
This mask show which are the affected channels that the period
is changed for.
```

1.3 changesoundvolume

```
  SYNTAX
Result.w = ChangeSoundVolume(#Obj.w,Volume.w)

  FUNCTION
Use this function to change the volume when the sound
is actually played.

The Volume set in the Sound Object, that is used when
the sound is started, is not affected with this call.

#Obj
The Object to change volume for.

Volume
This is the new volume value.

Result
This mask show which are the affected channels that the
volume is changed for.
```

1.4 createsound

```
  SYNTAX
Result.l = CreateSound(#Obj.w,Length.l)

  FUNCTION
This function create a new Sound Object and all of it's
sound data is set to zero. Period, Volume and Channels
are also set to zero.

To read and write the sound data use PeekSoundData()
```

```
and PokeSoundData(). Period, Volume and Channels have
to be set with the appropriate statement.

#Obj
The Object to create.

Length
The length of the sound data.

It's useless to specify any value higher than 128K
here as the Amiga hardware doesn't support it.

Result
If this is TRUE the Object could be created else
it is FALSE.
```

1.5 decodesound

```
  SYNTAX
Result.w = DecodeSound(#Obj.w,Pointer.l)

  FUNCTION
This function initialize a Sound Object from the
IFF sound stored inside the program.

* Period are set to value taken from IFF file.
* Volume is set to 64.
* Channels is set to 15.

If the Sound Object is already initialized it must
first be freed with a call to FreeSound() else it
stay in memory and there will be no possibility to
play it.

#Obj
The Object to use.

Pointer
A pointer to the IFF sound.

Result
This is TRUE if the Sound Object could be initialized
from the included IFF sound else it is FALSE.
```

1.6 freesound

```
  SYNTAX
FreeSound(#Obj.w)

  STATEMENT
This statement frees a Sound Object that is initialized
with LoadSound() or DecodeSound().
```

```
If no fast memory is availible then the program end up in
chip memory and a Sound Object that is initialized with
DecodeSound() is valid until the Object is initialized
with; LoadSound(); or again DecodeSound(), another
included IFF sound is used.
Which mean, such Object could never be freed.

#Obj
The Object to free.
```

1.7 getsoundlength

```
  SYNTAX
Result.l = GetSoundLength(#Obj.w)

  FUNCTION
This function gets the length of a Sound Object.

Use it for calculations so PeekSoundData() and
PokeSoundData() not read and write outside of
the actual sound data.

#Obj
The Object to use.

Result
The length of the sound data.
```

1.8 initsound

```
  SYNTAX
Result.w = InitSound(Objects.l)

  FUNCTION
This function is the initroutine, it set up all needed stuff,
and it must be called befor all other functions.
It could only be called once.

Objects
This is how many Sound Objects that is wanted.

Result
If this is TRUE the call was successful else it is FALSE and
then no other functions could be called.
```

1.9 loadsound

```
  SYNTAX
Result.b = LoadSound(#Obj.w,FileName$)

  FUNCTION
Use this function to initialize a Sound Object from an IFF
sound stored on disk.

* Period are set to value taken from IFF file.
* Volume is set to 64.
* Channels is set to 15.

If the Sound Object is already initialized it must first be
freed with a call to FreeSound() else it stay in memory and
there will be no possibility to play it.

#Obj
The Object to use

FileName
This is the full path to the IFF sound.

Result
It will be TRUE if the Sound Object could be initialized from
the specified IFF sound else it is FALSE.
```

1.10 peeksounddata

```
  SYNTAX
Result.b = PeekSoundData(#Obj.w,Position.l)

  FUNCTION
This function read some sound data from a Sound Object.

#Obj
The Object to use.

Position
This is where in the Sound Object the data should be read.

Result
The data read, it could range between -128 and 127.
```

1.11 playsound

```
  SYNTAX
Result.w = PlaySound(#Obj.w,Repeat.w)

  FUNCTION
This function play the specified Sound Object and it use
the values; Period, Volume and Channels set in the Object.
```

```
#Obj
The Object to play.

Repeat
To have the sound played one or more times and then stoped
set this to a positive none zero value, if the sound should
be repeated forever set it to minus.

Result
This mask show the actual channels that is used for this
sound, zero indicate that the sound is not played.
```

1.12 pokesounddata

```
  SYNTAX
PokeSoundData(#Obj.w,Position.l,Data.b)

  STATEMENT
This statement write some data to a Sound Object.

#Obj
The Object to use.

Position
This specify where in the Sound Object the data
should be written.

Data
The data to write, should range from -128 to 127.
```

1.13 savesound

```
  SYNTAX
Result.b = SaveSound(#Obj.w,FileName$)

  FUNCTION
This function save a Sound Object to disk as a IFF sound.

#Obj
The Object to save.

FileName
This is the full path to where the IFF sound should be saved.

Result
It will be TRUE if the Sound Object could be saved to disk as
the specified IFF sound else it is FALSE.
```

1.14 setsoundchannels

```
  SYNTAX
SetSoundChannels(#Obj.w,Channels.w)

  STATEMENT
This statement set Channels in the Sound Object that
is used when the sound is played.

The Channels are set to 15 when the Sound Object is
initialized with LoadSound() or DecodeSound().

#Obj
The Object to set channels for.

Channels
This is a mask that specify which channels that should
be used for this Object.

1 = use only channel 0
2 = use only channel 1
4 = use only channel 2
8 = use only channel 3

When added together:

 5 = use channel 0 and 2
15 = use all channels
```

1.15 setsoundperiod

```
  SYNTAX
SetSoundPeriod(#Obj.w,Period.w)

  STATEMENT
This statement set Period in the Sound Object that
is used when the sound is played.

The Period will be correctly set to the right value,
taken from IFF file, when Sound Object is initialized
with LoadSound() or DecodeSound().

#Obj
The Object to set period for.

Period
The period, it should be the final value as no
calculations at all is done on this.
```

1.16 setsoundvolume

```
  SYNTAX
SetSoundVolume(#Obj.w,Volume.w)
```

```
  STATEMENT
This statement set Volume in the Sound Object
that is used when the sound is played.

The Volume are set to 64 when the Sound Object is
initialized with LoadSound() or DecodeSound().

#Obj
The Object to set volume for.

Volume
The volume, should range between 0 and 64.
```

1.17 soundfilter

```
  SYNTAX
SoundFilter(ON/OFF)

  STATEMENT
This statement turn the audiofiler on or off.

ON/OFF
Set this to TRUE to turn the audiofilter ON or
set it to FALSE to turn audiofiler OFF.
```

1.18 stopsound

```
  SYNTAX
StopSound(#Obj.w)

  STATEMENT
Use this statement to stop a sound.

#Obj
The Object to stop.
```

1.19 usesoundchannels

```
  SYNTAX
Result.w = UseSoundChannels(Channels.w)

  FUNCTION
This function frees and allocate channels, both that in
the same call.

Lets say:
If all channels are successfully allocated on first call,
Channels set to 15, and then if channel 2 and 3 should
be freed then have Channels set to 3 on second call,
```

```
channel 0 and 1 stays allocated; 2 and 3 are freed.

Channels
This is the wanted channels which also say what
channels to free.

Result
If this mask is TRUE it show which channels that could
be allocated else it is FALSE and the wanted channels
are not allocated.
```