SPObject_Ref_ENG

COLLABORATORS			
	TITLE : SPObject_Ref_ENG		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SPObject_Ref_ENG

1.1 SPObject_Ref_ENG.doc

--background--

SPO_AllocHandle()

SPO_FreeHandle()

SPO_Read()

SPO_StartReplay()

SPO_Write()

SPO_StopReplay()

SPO_FreeResources()

SPO_SetAccessMode()

SPO_SetWriteSubType()

SPO_SetWriteName()

SPO_SetReadName()

SPO_FileInfoRequest()

SPO_CheckFileType()

SPO_SetReqIOWindow()

SPO_ContinueReplay()

SPO_FastForward()

SPO_FastBackward()

SPO_GetSampleBuffer()

```
               SPO_GetSampleInfo()

               SPO_GetSampleList()

               SPO_SetSampleList()
```

1.2 SPObject.library/--background--

```
VERSION
     $VER: SPObject_Ref_ENG.doc V6.3 (3.4.97)

COPYRIGHT
     © 1995-97 by Andreas R. Kleinert. All rights reserved.

     - Feel free to translate this Doc-File into other languages. -

GENERAL
       Andreas R. Kleinert,
       Sandstrasse 1,
       D-57072 Siegen,
       Germany.

     EMail:  Fido             Andreas Kleinert 2:2457/350.18
             Usenet/InterNet  Andreas_Kleinert@superview.ftn.neckar-alb.de

       If nothing else works, try one of these Fido-InterNet gateways:

             Andreas_Kleinert@p10.f435.n2457.z2.fido.sub.org (in Germany)
             Andreas_Kleinert@p10.f435.n2457.z2.fidonet.org  (USA or other)

NOTE
     * DO _NEVER_ ACCESS ANY SPOBJECTS DIRECTLY.
     * DO NEVER BYPASS superplay.library !

     THE FOLLOWING NOTES ARE ONLY FOR PROGRAMMERS OF SPOBJECTS :


HISTORY
     V2.5   Autodoc format
            Change one SVO_ prefix to SPO_
            Add version behind NAME
            SPObject.library/SPO_SetReadeName
                                      ^
              -> SPObject.library/SPO_SetReadName(indy)
     V5.1   applied some more changes (ak)
     V5.2   add "()" to function names in text for better autodocs
            support(indy)
     V6.3   bumped version
            updated copyright note
            updated email address list (ak)
```

1.3 SPObject.library/SPO_AllocHandle

```
                NAME
     SPO_AllocHandle  (V1)

SYNOPSIS

     APTR SPO_AllocHandle(APTR future)
     D0   -$1e            A1

FUNCTION

     Allocates a handle for accessing a Sample/Module via SPObjects.

INPUT(S)

     future - always NULL yet

RESULT

     A pointer to a new allocated Handle or NULL, if allocation failed.

WARNING

     Test, if the result was NULL, or not !

SEE ALSO


              SPO_FreeResources()
              ,
              SPO_FreeHandle()
```

1.4 SPObject.library/SPO_FreeHandle

```
                NAME
     SPO_FreeHandle  (V1)

SYNOPSIS

     VOID SPO_FreeHandle(APTR handle)
     D0   -$24           A1

FUNCTION

    Stops playing, frees all Resources and delocates a Handle, which has
    been allocated with
             SPO_AllocHandle()
              before.

INPUT(S)

     handle - a valid handle
```

```
RESULT

     -

SEE ALSO


             SPO_AllocHandle()
             ,
             SPO_StopReplay()
             ,
             SPO_FreeResources()
```

1.5 SPObject.library/SPO_Read

```
                NAME
     SPO_Read  (V1)

SYNOPSIS

     ULONG SPO_Read(APTR handle)
     D0    -$2a     A1

FUNCTION

     Loads the Sample/Module described by FileName, which then may
     e.g. be replayed by
             SPO_StartReplay()
             .

INPUT(S)

     handle   - a valid handle

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


             SPO_AllocHandle()
             ,
             SPO_StopReplay()
             ,
             SPO_StartReplay()
             ,

             SPO_ContinueReplay()
             ,
             SPO_FastForward()
             ,
             SPO_FastBackward()
```

```
              ,

              SPO_FreeHandle()
```

1.6 SPObject.library/SPO_StartReplay

```
                NAME
     SPO_StartReplay  (V1)

SYNOPSIS

     ULONG SPO_StartReplay(APTR handle)
     D0    -$30            A1

FUNCTION

     Play the Sample/Module described by FileName, which has been loaded
     via
             SPO_Read()
              before.

     Playing can be stopped either via full delocation of the handle
     or temporal interruption with
             SPO_StopReplay()
             .

INPUT(S)

     handle   - a valid handle

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


             SPO_AllocHandle()
             ,
             SPO_Read()
             ,
             SPO_StopReplay()
             ,

             SPO_ContinueReplay()
             ,
             SPO_FastForward()
             ,
             SPO_FastBackward()
             ,

             SPO_FreeHandle()
```

1.7 SPObject.library/SPO_Write

```
                NAME
     SPO_Write  (V1)

SYNOPSIS

     ULONG SPO_Write(APTR handle, struct SPObjectBase *SourceBase,
     D0    -$36      A1           A2

                     APTR source_handle)
                     A3

FUNCTION

     This functions allows to access an other SPObject, get the Sample-
     Data from there (only works with SampleType-SPObjects) and
     then writes this data in its own FileFormat.

     If "SourceBase" and "source_handle" are both NULL,
     it is checked, whether a SampleList has been set via
     "SPO_SetSampleList()", and then this SampleList is completely
     saved.

INPUT(S)

     handle        - a valid handle
                     (used for Write Access)
     SourceBase    - pointer to Base of other SPObject
                     (may be NULL for V2+ SPObjects)
     source_handle - handle for accessing the other SPObject
                     (may be NULL for V2+ SPObjects)

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_AllocHandle()
              ,
              SPO_Read()
              ,
              SPO_FreeHandle()
```

1.8 SPObject.library/SPO_StopReplay

```
                NAME
     SPO_StopReplay  (V1)

SYNOPSIS
```

```
     VOID SPO_StopReplay(APTR handle)
     D0   -$3c           A1

FUNCTION

     Stops playing the Sample/Module, indentified by the handle.
     Some SPObjects support to continue the Sample/Module with

             SPO_ContinueReplay()
              after calling SPO_StopReplay().

INPUT(S)

     handle - a valid handle

RESULT

     -

SEE ALSO


             SPO_ContinueReplay()
             ,
             SPO_FreeResources()
             ,
             SPO_FreeHandle()
```

1.9 SPObject.library/SPO_FreeResources

```
                NAME
     SPO_FreeResources  (V1)

SYNOPSIS

     VOID SPO_FreeResources(APTR handle)
     D0   -$42              A1

FUNCTION

     Frees all resources belonging to the specific Sample/Module,
     indentified by the handle, which are not needed to just replay it.
     Playing of the Sample/Module is not stopped and/or interrupted.

INPUT(S)

     handle - a valid handle

RESULT

     -

SEE ALSO
```

```
SPO_AllocHandle()
,
SPO_StopReplay()
,
SPO_FreeHandle()
```

1.10 SPObject.library/SPO_SetAccessMode

```
NAME
     SPO_SetAccessMode  (V1)

SYNOPSIS

     ULONG SPO_SetAccessMode(APTR handle, ULONG mode)
     D0    -$48              A1           D1

FUNCTION

     Initializes a Handle for ClipBoard or AmigaDOS access,
     depending on the "mode" parameter.

INPUT(S)

     handle - a valid handle
     mode   - access mode value


RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO

     -
```

1.11 SPObject.library/SPO_SetWriteSubType

```
               NAME
     SPO_SetWriteSubType  (V1)

SYNOPSIS

     ULONG SPO_SetWriteSubType(APTR handle, ULONG write_type, APTR future)
     D0    -$4e                A1           A2                A3

FUNCTION

     Sets the write_type for a
             SPO_Write()
              call.
```

```
     See description there and example SourceCodes for more and
     detailed information.

INPUT(S)

     handle     - a valid handle
     write_type - a valid temporary write_type code form the SPObject List
     future     - always NULL yet


RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_Write()
```

1.12 SPObject.library/SPO_SetWriteName

```
                NAME
     SPO_SetWriteName  (V1)

SYNOPSIS

     ULONG SPO_SetWriteName(APTR handle, UBYTE *write_name, APTR future)
     D0    -$54             A1           A2                 A3

FUNCTION

     Sets the write_name for a
             SPO_Write()
              call.
     See description there and example SourceCodes for more and
     detailed information.

INPUT(S)

     handle     - a valid handle
     write_name - a valid AmigaDOS FilePath and -Name description
     future     - always NULL yet


RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_Write()
```

1.13 SPObject.library/SPO_SetReadName

```
                NAME
     SPO_SetReadName  (V1)

SYNOPSIS

     ULONG SPO_SetReadName(APTR handle, ULONG reade_name, APTR future)
     D0    -$5a            A1           A2                A3

FUNCTION

     Sets the reade_name for a
             SPO_Read()
              call.
     See description there and example SourceCodes for more and
     detailed information.

INPUT(S)

     handle     - a valid handle
     write_name - a valid AmigaDOS FilePath and -Name description
     future     - always NULL yet


RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_Write()
```

1.14 SPObject.library/SPO_FileInfoRequest

```
                NAME
     SPO_FileInfoRequest  (V1)

SYNOPSIS

     ULONG SPO_FileInfoRequest(APTR handle, struct Window *window,
     D0    -$60                A1           A2

                               APTR future)
                               A3

FUNCTION

     Pops up an Info-Requester with more or less detailed information
     on the currently loaded Sample/Module.
     A window pointer may be given to select the place to pop it up.
```

```
INPUT(S)

     handle - a valid handle
     window - a valid Window Pointer or NULL
     future - always NULL yet


RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_SetReqIOWindow()
```

1.15 SPObject.library/SPO_CheckFileType

```
NAME
     SPO_CheckFileType  (V1)

SYNOPSIS

     ULONG SPO_CheckFileType(BPTR filehandle, UBYTE *filename,
     D0    -$66              A1                 A2

                             struct SPOCheckFile *spo_check)
                             A3

FUNCTION

     Checks, if the given file (or ClipBoard entry, or whatever)
     fits to this SPObject and can be handled therein.

     Other media as SPO_MEDIUM_DISK requite spo_check to
     be non-NULL and initialized.

     Be prepared, to handle NULL pointers to filename and handle.

INPUT(S)

     handle    - a valid handle
     name      - a valid AmigaDOS FileName
     spo_check - a pointer to a SPOCheckFile structure or NULL
                 for disk-access (default)

RESULT

     NULL or an adequate SVERR-Errorcode.

SEE ALSO

     -
```

1.16 SPObject.library/SPO_SetReqIOWindow

```
NAME
     SPO_SetReqIOWindow  (V1)

SYNOPSIS

     ULONG SPO_SetReqIOWindow(APTR handle, struct Window *window)
     D0    -$6c               A1           A2

FUNCTION

     Sets a new Default-Window (default : IntuitionBase->FirstWindow)
     for any Requester IO.

INPUT(S)

     handle - a valid handle
     window - a valid Window Pointer or NULL

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO

     SPO_FileInfoReq
```

1.17 SPObject.library/SPO_ContinueReplay

```
                NAME
     SPO_ContinueReplay  (V1)

SYNOPSIS

     ULONG SPO_ContinueReplay(APTR handle)
     D0    -$72               A1

FUNCTION

     Some SPObjects support to continue a Sample/Module which
     has been stopped by calling
             SPO_StopReplay()
             .

INPUT(S)

     handle - a valid handle

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO
```

```
     SPO_SuperPlay(),
             SPO_StopReplay()
```

1.18 SPObject.library/SPO_FastForward

```
                NAME
     SPO_FastForward  (V1)

SYNOPSIS

     ULONG SPO_FastForward(APTR handle)
     D0    -$78            A1

FUNCTION

     Some SPObjects might support a "cassette recorder"-like way
     to browse trough a Sample/Module via FastForward and Rewind.

INPUT(S)

     handle - a valid handle

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


             SPO_FastBackward()
```

1.19 SPObject.library/SPO_FastBackward

```
                NAME
     SPO_FastBackward  (V1)

SYNOPSIS

     ULONG SPO_FastBackward(APTR handle)
     D0    -$7e            A1

FUNCTION

     Some SPObjects might support a "cassette recorder"-like way
     to browse trough a Sample/Module via FastForward and Rewind.

INPUT(S)

     handle - a valid handle
```

```
RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_FastForward()
```

1.20 SPObject.library/SPO_GetSampleBuffer

```
                NAME
     SPO_GetSampleBuffer  (V1)    *** OBSOLETE : Use
             SPO_GetSampleList()
                SYNOPSIS

     ULONG SPO_GetSampleBuffer(APTR handle, UBYTE **buffer,
     D0    -$84                A1           A2

                               ULONG *buffersize)
                               A3

FUNCTION

     This function is OBSOLETE. Don't use it.

     Use
             SPO_GetSampleList()
              instead, which can handle more than one
     Sample and also returns the information related to them.

INPUT(S)

     handle     - a valid handle
     buffer     - a pointer to a buffer pointer
     buffersize - a pointer to a buffersize pointer

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


              SPO_GetSampleList()
```

1.21 SPObject.library/SPO_GetSampleInfo

```
            NAME
     SPO_GetSampleInfo  (V1)     *** OBSOLETE : Use
             SPO_GetSampleList()
            SYNOPSIS

     ULONG SPO_GetSampleInfo(APTR handle, ULONG *samplesPerSec,
     D0    -$8a              A1           A2

                             ULONG *volume, APTR future)
                             A3             D1

FUNCTION

     This function is OBSOLETE. Don't use it.

     Use
             SPO_GetSampleList()
              instead, which can handle more than one
     Sample and also returns the information related to them.

     This function only returns info on 8 Bit Samples.

INPUT(S)

     handle        - a valid handle
     samplesPerSec - a pointer to a buffer pointer
     volume        - a pointer to a buffersize pointer
     future        - always NULL yet

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO


             SPO_GetSampleList()
```

1.22 SPObject.library/SPO_GetSampleList

```
            NAME
     SPO_GetSampleList  (V2)

SYNOPSIS

     ULONG SPO_GetSampleList(APTR handle, struct SPO_SampleList **list)
     D0    -$90              A1           A2

FUNCTION

     While/after loading a Sample-File SPObjects might create
     a special list of all Samples, which appear in the File.
     This will usually be one one Sample for SampleFiles, but many more
```

```
     for ModuleFiles.

     Not all SPObjects, especially not all ModuleType-SPObjects,
     may support this and will return an error value or an empty list.

INPUT(S)

     handle - a valid handle
     list   - a pointer to a SPO_SampleList pointer

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO

              SPO_SetSampleList()
```

1.23 SPObject.library/SPO_SetSampleList

```
                NAME
     SPO_SetSampleList  (V2)

SYNOPSIS

     ULONG SPO_SetSampleList(APTR handle, struct SPO_SampleList *list)
     D0    -$96              A1           A2

FUNCTION

     For saving Sample-Files with, there may be used
     a special list of all Samples, which should appear in the File.

     This will usually be one one Sample for SampleFiles, but many more
     for ModuleFiles.

     Not all SPObjects, especially not all ModuleType-SPObjects,
     may support this and will return SPERR_ACTION_NOT_SUPPORTED.

INPUT(S)

     handle - a valid handle
     list   - a pointer to a SPO_SampleList

RESULT

     NULL or an adequate SPERR-Errorcode.

SEE ALSO

              SPO_GetSampleList()
```