NetSupport_internal

COLLABORATORS			
	TITLE : NetSupport_internal		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 8, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

NetSupport_internal

1.1 NetSupport_internal.doc

```
FreeStaticBuffer()
GetStaticBuffer()
NSPAllocMemPooled()
NSPFreeMemPooled()
FreePoolList()
LockPoolList()
StuffChar()
LoadFile()
SPrint()
StringCat()
StringCopy()
```

1.2 MemoryHandling.asm/FreeStaticBuffer

```
NAME
   FreeStaticBuffer -- deallocate a task's static buffer

SYNOPSIS
   FreeStaticBuffer(void);

FUNCTION
   This routine gets the tasks static buffer, frees all allocated
   memory that has been tracked in the lists and  removes the static
   buffer itself from the library internal list.

   This routine may be modified as more entries are added to the
   static buffer structure--currently holding only memory allocations.
   However, it is possible to track opened files, and more...

INPUTS
   none

RESULT
   none
```

```
NOTE
   This routine is usually called when the library is closed!

SEE ALSO
   GetStaticBuffer()
```

1.3 MemoryHandling.asm/GetStaticBuffer

```
NAME
   GetStaticBuffer -- find a task's static data buffer

SYNOPSIS
   staticbuf = GetStaticBuffer(void);

   struct StaticBuffer *staticbuf;

FUNCTION
   Because routines in a shared library have to be re-entrant, it's
   a little bit difficult to store certain data statically, as you
   can't just use an entry in the library header.

   GetStaticBuffer() initializes a special structure for each task
   and returns a pointer, so the routine can store the required data
   in this structure for later usage.

INPUTS
   none

RESULT
   GetStaticBuffer() returns a pointer to the task's individual
   structure or NULL for failure.

EXAMPLE
   Here's a short example to demonstrate the usage of
   GetStaticBuffer. The following routine will count the number of
   times it has been called and returns the integer. (Not very
   useful, agreed. :->)

           LONG CountMe(void)
           {
                   struct StaticBuffer *sb;

                   if (sb = GetStaticBuffer())
                           return ++(sb->nspsb_CountMe);
                   else
                         return 0L;
           }

SEE ALSO
   FreeStaticBuffer()
```

1.4 MemoryHandling.asm/NSPAllocMemPooled

```
NAME
   NSPAllocMemPooled -- allocate memory for library internal usage

SYNOPSIS
   memblock = NSPAllocMemPooled(blocksize, attributes);
   D0                           D0         D1

   void *  memblock;
   ULONG   blocksize;
   ULONG   attributes;

FUNCTION
   This routine does mainly the same as AllocMemPooled(). The only
   difference is, that the allocated memory is tracked in a different
   list. The advantage is, that FreeStaticBuffer() is able to
   determine wether this was an internal required memoryblock or the
   user has forgotten to free one of his allocations. Future versions
   of the library might pop up a requester a la: "The following
   allocated memoryblocks were not freed at run-time:...".

   This might be useful for debugging purposes.

INPUTS
   blocksize = this is the size the requested block in byte

   attributes = AllocMemPooled() understands exactly the same
           attributes as the original exec routines. Please refer to
           AllocMem() or exec/memory.h for further details.

RESULT
   The routine returns either the address of the allocated block or
   NULL, if the routine failed due to low-memory condition.

NOTE

SEE ALSO
   NSPFreeMemPooled(), AllocMemPooled(), FreeMemPooled()
```

1.5 netsupport.library/NSPFreeMemPooled

```
NAME
   NSPFreeMemPooled -- free a memory block previously allocated
                   with AllocMemPooled()

SYNOPSIS
   success = NSPFreeMemPooled(memblock);
   D0                         A1

   LONG    success;
   void *  memblock;

FUNCTION
   This routine frees a block, allocated with NSPAllocMemPooled()
```

```
   earlier. Just the address is required, the library keeps track
   of the blocksizes itself.

INPUTS
   memblock = pointer the the memory block

RESULT
   Either -1L for success or 0L for failure.

NOTE
   The memory allocation tracking mechanism will catch an attempt
   to free a memory block twice. However, please do NOT rely on
   this feature!

SEE ALSO
   NSPAllocMemPooled(), AllocMemPooled(), FreeMemPooled()
```

1.6 NetSupportLibrary.asm/FreePoolList

```
NAME
   FreeAnyMemPool -- easy way of releasing the AnyMemPoolSemaphore


SYNOPSIS
   FreeAnyMemPool();

FUNCTION
   Releases the AnyMemPoolSemaphore for usage by other tasks.

INPUTS
   none

RESULT
   none

NOTE
   All registers are preserved.

SEE ALSO
   LockAnyMemPool()
```

1.7 NetSupportLibrary.asm/LockPoolList

```
NAME
   LockAnyMemPool -- easy way of obtaining the AnyMemPoolSemaphore


SYNOPSIS
   LockAnyMemPool();

FUNCTION
```

```
   Tries to obtain the AnyMemPoolSemaphore. If someone else is using
   it at the moment, the task is delayed until it becomes free. This
   is vital for modifying linked lists.

INPUTS
   none

RESULT
   none

NOTE
   All registers are preserved.

SEE ALSO
   FreeAnyMemPool()
```

1.8 NetSupportLibrary.asm/StuffChar

```
NAME
   StuffChar -- routine to write one character into a buffer


SYNOPSIS
   StuffChar(character, buffer);
   D0        D0.b       A3

   UBYTE   character;
   STRPTR  buffer;

FUNCTION
   This is a small assembler routine (two lines, in fact!) that is
   suited to be called by RawDoFmt() and writes the provided
   character into a buffer.

INPUTS
   character = a simple byte value
   buffer = pointer to a text buffer

RESULT
   none

NOTE
   StuffChar() increases the pointer to the buffer by one!! This
   behavior is required by RawDoFmt().

SEE ALSO
   RawDoFmt()
```

1.9 SupportRoutines.c/LoadFile

```
NAME
   LoadFile -- loads a file into a buffer
```

```
SYNOPSIS
    buffer = LoadFile(filename);
    D0                A0

    STRPTR buffer;
    STRPTR filename;

FUNCTION
    LoadFile() -- determines the length of a given file, allocates
    the appropriate buffer and actually loads the file.

INPUTS
    filename - pointer to a zero-terminated name string. The string
                    may contain an relative or absolute path.

RESULT
    If everything works, the address of the buffer is returned and
    the length of the file is available via IoErr().

    The buffer should be free after usage using NSPFreeMemPooled(),
    however the memory allocation tracking mechanism will take
    care if you forget to free it.

NOTES
    A zero-byte is appended to the loaded file, to enable the
    usage of standard C string routines on the contents.

SEE ALSO
```

1.10 SupportRoutines.c/SPrint

```
NAME
    SPrint -- format data into a character stream.

SYNOPSIS
    SPrint(buffer, fmt, ...);

    STRPTR buffer;
    STRPTR fmt;

FUNCTION
    Perform "C"-language-like formatting of a data stream, outputting
    the result a character at a time.  Where % formatting commands are
    found in the FormatString, they will be replaced with the
    corresponding element in the DataStream.  %% must be used in the
    string if a % is desired in the output.

INPUTS
    buffer -- Pointer to a buffer large enough enough to hold the
            formatted string.

    fmt -- Control string describing how the string and the parameters
            shall be formatted. Please read the autodoc of RawDoFmt()
            for further information.
```

```
RESULT
    none

NOTES
    This routine expects it's parameters on the stack. No
    registerized version is available.

SEE ALSO
    RawDoFmt()
```

1.11 SupportRoutines.c/StringCat

```
NAME
    StringCat -- append one string to the end of another one

SYNOPSIS
    end_of_string = StringCat(part1, part2);
    D0                        A0     A1

    STRPTR end_of_string, part1, part2;

FUNCTION
    This routine does the same as the ANSI-C strcat() routine, but
    this version is more suitable for a shared library as it doesn't
    require any link libraries, etc...

INPUTS
    part1 - Pointer to a textbuffer large enough to hold both
                  strings. No buffer overflow-checking is performed.

    part2 - Pointer to a zero-terminated string to be appended at the
                  end of >part1<.

RESULT
    The resulting pointer points to the zero-byte of the copied string
    in the destination pointer. Using this pointer, it is possible to
    append several strings easily. See StringCopy() for a code example.

SEE ALSO
```

1.12 SupportRoutines.c/StringCopy

```
NAME
    StringCopy -- copies a zero-terminated string into a buffer

SYNOPSIS
    end_of_string = StringCopy(buffer, string);
    D0                         A0      A1

    STRPTR end_of_string;
    STRPTR buffer;
```

```
    STRPTR string;

FUNCTION
    This routine does the same as the ANSI-C strcpy() routine, but
    this version is more suitable for a shared library as it doesn't
    require any link libraries, etc...

INPUTS
    buffer - Pointer to a textbuffer large enough to hold the
                    string. No buffer overflow-checking is performed.

    string - Pointer to a zero-terminated string to be copied.

RESULT
    The resulting pointer points to the zero-byte of the copied string
    in the destination pointer. Using this pointer, it is possible to
    append several strings easily. See example.

EXAMPLE
    Append two strings in a target buffer using StringCopy():

            STRPTR buffer[256];
            const char part1[] = "These two strings ";
            const char part2[] = "belong together!";

            StringCopy(StringCopy(buffer, part1), part2);
            Printf("%s\n", buffer);

SEE ALSO
```