# NetSupport

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>NetSupport | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | December 8, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

**1 NetSupport** **1**

1.1 NetSupport.doc . . . 1
1.2 netsupport.library/AllocMemPooled . . . 1
1.3 netsupport.library/ExpungeResident . . . 2
1.4 netsupport.library/FreeMemPooled . . . 3
1.5 netsupport.library/GetConfig . . . 3
1.6 netsupport.library/GetConfigEntry . . . 4
1.7 netsupport.library/GetNSPConfigName . . . 6
1.8 netsupport.library/GetSeq . . . 6
1.9 netsupport.library/IsFileLocked . . . 7
1.10 netsupport.library/LoadResident . . . 8
1.11 netsupport.library/LockFile . . . 9
1.12 netsupport.library/LockFileAttempt . . . 10
1.13 netsupport.library/MakeLogEntry . . . 11
1.14 netsupport.library/PClose . . . 12
1.15 netsupport.library/POpen . . . 12
1.16 netsupport.library/SetConfigEntry . . . 13
1.17 netsupport.library/TempName . . . 14
1.18 netsupport.library/TempNameT . . . 16
1.19 netsupport.library/UnLockFile . . . 17
1.20 netsupport.library/UnLockFiles . . . 17

# Chapter 1

# NetSupport

## 1.1 NetSupport.doc

```
AllocMemPooled()
ExpungeResident()
FreeMemPooled()
GetConfig()
GetConfigEntry()
GetNSPConfigName()
GetSeq()
IsFileLocked()
LoadResident()
LockFile()
LockFileAttempt()
MakeLogEntry()
PClose()
POpen()
SetConfigEntry()
TempName()
TempNameT()
UnLockFile()
UnLockFiles()
```

## 1.2 netsupport.library/AllocMemPooled

```
NAME
   AllocMemPooled -- allocate a block of memory out of the
           libraries memory pool

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

   void *  memblock;
   ULONG   blocksize;
   ULONG   attributes;

FUNCTION
```

```
   This routines allocates a memory block of the requested size
   using the libraries internal memory pool. This is a big
   advantage when several smaller allocations have to be made,
   because memory pooling reduces fragmentation significantly.

   Using this routine you won't have to care about managing the
   pool, the library will do for you. If the routine is started
   under exec.library v39 or later, the exec pool routines will be
   used. Under Kickstart 37, the library will handle the pool
   itself.

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
   The library tracks all allocations you do and frees them itself
   when CloseLibrary() is called. However, it is strongly
   recommended to free all memory IMMEDIATELY when it isn't needed
   anymore. It is very bad programming style to rely on the library
   freeing everything for you.

   If you're calling the routine in another process, you created
   using CreateNewProc() or a similar routine, you MUST open the
   library again! Do not just pass the LibraryBase to the new
   process or you'll break the memory-tracking mechanism, either
   causing loss of memory or even a complete system crash.

SEE ALSO
   FreeMemPooled()
```

## 1.3 netsupport.library/ExpungeResident

```
NAME
   ExpungeResident -- expunges all resident files and thus frees
           memory

SYNOPSIS
   ExpungeResident(void);

FUNCTION
   This routine frees all allocated buffers and expunges all loaded
   resident modules. This routine is usually called by the OS when
   memory is lacking and should not be used by other programs.

INPUTS
   none
```

```
RESULT
   none

SEE ALSO
   LoadResident()
```

## 1.4 netsupport.library/FreeMemPooled

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

SYNOPSIS
   success = FreeMemPooled(memblock);
   D0                      A1

   LONG    success;
   void *  memblock;

FUNCTION
   This routines frees a memory block, allocated with AllocMemPooled()
   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
   AllocMemPooled()
```

## 1.5 netsupport.library/GetConfig

```
NAME
   GetConfig -- return a configuration entry or system variable

SYNOPSIS
   entry = GetConfig(filename, keyword, dosvarname, default);
   D0                A0        A1       A2          A3

   STRPTR  entry;
   STRPTR  filename;
   STRPTR  keyword;
```

```
   STRPTR  dosvarname;
   STRPTR  default;

FUNCTION
   GetConfig() does pretty much the same as GetConfigEntry(),
   except that this routine is able to recognize local and global
   DOS variables.


INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the configfile in case no variable is set.

           When <filename> is NULL, the routine uses the configfile,
           you specified as 'MasterConfig' in the library's own
           configfile. The default is 'UULib:Config'.

   keyword - Pointer to a keyword string. All comparisons are
           case-independent!

   dosvarname - pointer to a name of the local shell- and global
           enviroment variable to check. When this parameter is
           NULL, <keyword> is used. However, some purposes may
           require a different variable name than config file
           keyword.

   default - pointer to a string containing the default parameter,
           which is returned in case, the keyword can't be found.

RESULT
   GetConfig() returns a pointer to a static buffer, holding the
   provided parameter. Please take note that the contents of this
   buffer is lost, when GetConfig() or GetConfigEntry() is called
   the next time. You MUST not free or modify this library-internal
   buffer!

   If no config entry matching your keyword was available, the
   provided default is returned.

NOTE
   GetConfig() does NOT lock the file in order to avoid deadlocks.

SEE ALSO
   GetConfigEntry(), ParseConfigFile()
```

## 1.6 netsupport.library/GetConfigEntry

```
NAME
   GetConfigEntry -- return a configuration entry

SYNOPSIS
   entry = GetConfigEntry(filename, keyword, default);
   D0                     A0        A1       A2

   STRPTR  entry;
```

```
   STRPTR  filename;
   STRPTR  keyword;
   STRPTR  default;

FUNCTION
   This routine is designed to replace the old Dillon UUCP config
   routines and to perform some simple configuration issues.

   It checks if the provided keyword is available as local or
   global system variable and eventually returned its contents. If
   such a variable isn't set, it loads specified configfile and
   searches for the keyword, returning the specified parameter.

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the configfile in case no variable is set.

           When <filename> is NULL, the routine uses the configfile,
           you specified as 'MasterConfig' in the library's own
           configfile. The default is 'UULib:Config'.

   keyword - Pointer to a keyword string. All comparisons are
           case-independent!

   default - pointer to a string containing the default parameter,
           which is returned in case, the keyword can't be found.

RESULT
   GetConfigEntry() returns a pointer to a static buffer, holding
   the provided parameter. Please take note that the contents of
   this buffer is lost, when GetConfigEntry() or GetConfig() is
   called the next time. You MUST not free or modify this
   library-internal buffer!

   If no config entry matching your keyword was available, the
   provided default is returned.

NOTE
   GetConfigEntry() does NOT lock the file in order to avoid
   deadlocks.

CONFIGFILE SYNTAX
   A valid config file may contain one keyword per line. Keywords
   MUST begin at the first column. The parameter can be separated
   from the keyword using either space(s) or tab(s).

   GetConfigEntry() skips all leading blanks or tabs until the
   first "non-blank" character. Then it returns the whole line
   until the return, truncating any blanks or tabs at the end of
   the line. If you specify a keyword alone on a line,
   GetConfigEntry() will return a pointer to an empty string.

   Lines starting with a '#' are commentlines and will be ignored.


   Example:
           KEYWORD parameter
```

```
            FOO     bar
            # comment
            FO   " O  B A R "

   GetConfigEntry does not parse the parameter and therefore does
   NOT remove the double-quotes '"' in the last line!

SEE ALSO
   GetConfig(), ParseConfigFile()
```

## 1.7 netsupport.library/GetNSPConfigName

```
NAME
   GetNSPConfigName -- return the name and path of the libraries
           own config file.

SYNOPSIS
   name = GetNSPConfigName(void);
   D0

   STRPTR  name;

FUNCTION
   This routine tries to determine which path should be used for the
   libraries own config file. You can put the file anywhere you want
   and set the variable "NSPConfig" to the path and name of the file
   or "S:NSPConfig" will be assumed as default.

INPUTS
   none

RESULT
   Name and path of the configfile. If the routine fails due to
   memory lack, NULL is returned.

NOTE
   You must not free or modify the returned pointer!

SEE ALSO
```

## 1.8 netsupport.library/GetSeq

```
NAME
    GetSeq -- get a sequential, unique number

SYNOPSIS
    number = GetSeq(bump)
    D0              D0

    ULONG   number;
    ULONG   bump;
```

```
FUNCTION
    GetSeq() returns a number which is guaranteed to be unique.
    Despite the internal numbers used by TempName(), this number is
    still unique when the library was closed and flushed or the machine
    has been switched off.

    Obviously, this requires the number to be saved on disk.

INPUTS
    bump - The returned number is sequential. >bump< is the value that
                    is added to the current value. Using bump, you
                    can obtain (allocate) several numbers with one
                    call--reducing the number of necessary disk
                    accesses.

RESULT
    GetSeq() returns the current value of the counter and writes the
    increased counter back to disk.

NOTE
    GetSeq(0L) returns the current value, but does NOT increase the
    counter!

    The file that holds the current seq-counter can be configured in
    the library configfile. Please do never change this value manually
    unless you know what you do.

EXAMPLE
    The current counter is, say, 500. Now you call:

            num = GetSeq(4);

    num will have the value of 501. Additionally, you have the right
    to use the numbers 502, 503 and 504, because you specified a bump
    of 4.

SEE ALSO
```

## 1.9 netsupport.library/IsFileLocked

```
 NAME
    IsFileLocked -- returns the (lock-)status of a file

 SYNOPSIS
    boolean = IsFileLocked(filename);
    D0                     A0

    LONG    boolean;
    STRPTR  filename;

 BACKGROUND
    Please refer to LockFile() for a brief description.

 FUNCTION
    Sometimes you want to know wether a file is locked or not,
```

```
   without actually getting the exclusive access on the file--so
   this is the routine to use.

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the file.

RESULT
   IsFileLocked() returns either ZERO (0L) for an unlocked file,
   any positive value for 'file is locked' or a negative value for
   indicating an error.

NOTE
   Lockfile() is able to recognize, if the same file is accessed
   using two different paths, like "SYS:file" / "WB_2.x:file" for
   example. This is archived using the system call SameLock().
   However, due to this feature, LockFile() has to store the DOS
   lock of an already locked file, meaning that the file has to
   be created if it doesn't exist already.

SEE ALSO
   LockFile(), LockFileAttempt()
```

## 1.10 netsupport.library/LoadResident

```
NAME
   LoadResident -- loads a file and stores it resident, to buffer
           further access

SYNOPSIS
   buffer = LoadResident(filename)
   D0                    A0

   STRPTR  buffer;
   STRPTR  filename;

FUNCTION
   This routine loads the specified file and stores it in an
   internal resident list. The next time this file is accessed,
   the buffer will be returned immediately without disk access.

   The routine tests wether the file on disk is newer than the
   resident version and re-loads it in case it is.

INPUTS
   filename - name and path of the file that should be loaded

RESULT
   If the file couldn't be loaded for any reason, NULL is returned.
   Otherwise the routine returns a pointer to the buffer holding the
   file. The length of the file is returned by IoErr(). You must not
   modify or free the returned buffer!

NOTE
   The routine does NOT lock the file using LockFile()!
```

```
   The loaded file is always null terminated and thus can be used
   with string routines.

SEE ALSO
   ExpungeResident()
```

## 1.11 netsupport.library/LockFile

```
NAME
   LockFile -- lock a file before actually accessing it

SYNOPSIS
   success = LockFile(filename);
   D0                 A0

   LONG    success;
   STRPTR  filename;

BACKGROUND
   In a multitasking enviroment, like the Amiga is, it is important
   to coordinate and syncronize the usage of the available
   resources. This includes devices, memory and, of course, files.
   What happens, if your program needs a certain config file, while
   another program is reading/writing the file? You're attempt will
   fail, the access will be denied. Now things become difficult.

   Say, you just noticed that a file access failed. What should you
   do? Maybe the file is free in a few seconds, maybe you can do
   some other things while waiting for the file to become
   available.

   Using the libraries locking mechanism solves many problems for
   you. Most packages have recognized this problem and use some
   rather strange file locking mechanism, like Dillon UUCP 1.15.
   or the OwnDevUnit.Library for this purpose. Well, these
   mechanism had several disadvantages: They were not really
   designed for files!

   Both of them were not able to recognize that "UUMAIL:foobar" and
   "UUCP:Mail/foobar" is the same file! And if you'd just used the
   filenames without paths (what many authors did to come around
   the above problem), a lock to "UUMAIL:foobar" would fail if
   "ARCHIVE:foobar" was locked. All this problems are gone with
   LockFile().

   Forget about all this pseudo-names like "LOG-UPDATE.LOCK" and
   similar names, lock the complete filename, including path!
   LockFile() can handle all those situations, using SameLock() to
   determine of the same file is access with two different paths.

   Remember: Use the same filename to lock the file, you use for
   the actual Open() call, or the mechanism will break!

FUNCTION
   LockFile() determines the complete path to the file you
```

```
   specified and tests wether this file is locked by some other
   process and eventually waits for the file to become available.

   UnLockFile() has to be called when the file isn't required
   anymore.

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the file.

RESULT
   LockFile() returns TRUE if the file could be locked
   successfully. If an error occurs, FALSE will be returned.

SEE ALSO
   LockFileAttempt(), IsFileLocked(), UnLockFile(), UnLockFiles()
```

## 1.12 netsupport.library/LockFileAttempt

```
NAME
   LockFileAttempt -- try to lock a file

SYNOPSIS
   success = LockFileAttempt(filename);
   D0                        A0

   LONG    success;
   STRPTR  filename;

FUNCTION
   LockFileAttempt() does mainly the same as LockFile(). The only
   difference is, that LockFileAttempt() does not wait for the file
   to become available, but returns immediately.

   If the locking attempt was successful, UnLockFile() has to be
   called when the file isn't required anymore.

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the file.

RESULT
   LockFile() returns TRUE if the file could be locked
   successfully. If an error occurred, or the file is locked
   already, FALSE is returned.

NOTE
   Lockfile() is able to recognize, if the same file is accessed
   using two different paths, like "SYS:file" / "WB_2.x:file" for
   example. This is archived using the system call SameLock().
   However, due to this feature, LockFile() has to store the DOS
   lock of an already locked file, meaning that the file has to
   be created if it doesn't exist already.

SEE ALSO
```

```
   LockFile(), IsFileLocked(), UnLockFile(), UnLockFiles()
```

## 1.13 netsupport.library/MakeLogEntry

```
NAME
   MakeLogEntry -- generates an entry in the appropriate logfile

SYNOPSIS
   MakeLogEntry(system, class, logmessage, ...)
                A0      D0     A1          A2

   STRPTR  system;
   ULONG   class;
   STRPTR  logmessage;

FUNCTION
   This routine determines the appropriate logfile for an entry,
   using the specified system and class parameters (please refer to
   the libraries user manual for further information). Then it
   locks this file and writes the logmessage into it, prefaced with
   the current date and time.

INPUTS
   system - Pointer to a string, specifying the 'system', making the
            log entry. The system should be a short, descriptive
            name for your program, maybe the package it belongs to.
            The user can use the system name to direct the log-entries
            to certain logfiles, so better be verbose. :-)
            You should document the system name your program uses!
            If NULL is specified, the log-entry goes into the
            default logfile. If no default is specified, "T:Logfile"
            is used.

   class - value, describing the kind of the logmessage

   logmessage - pointer to a string containing the actual message
            text. Please refer to RawDoFmt() for usage.

RESULT
   none

CLASSES
   MLE_DEFAULT - This class is used if no special class can be
            determined (0L). This makes it easier to port older
            sources to the new routine--however, this entry should
            be avoided.

   MLE_INFO - This class should be used for plain information logs,
            what happened to what time.

   MLE_DEBUG[1-9] - This class should be used for debugging output.
            The number following the actual name is the debug level
            of the log-entry. MLE_DEBUG9, for example, is a rather
            unimportant debug message, while MLE_DEBUG0 is rather
```

```
            elementary and important to find bugs.

      MLE_ERROR - This class should be used to log occurring non-fatal
            errors.

      MLE_FATAL_ERROR - This class should be used to log fatal errors.
            Fatal errors are very important and usually need
            immediate fixing. So be careful using this class.

   EXAMPLE ENTRY
      (10-Nov-93/13:04:18) UUCP, Sending mail to foo@bar.UUCP
      (10-Nov-93/13:04:56) TCP/IP, Connected wuarchive.wust.edu
      (10-Nov-93/13:05:01) TCP/IP, Starting FTP request for foobar.lha

   NOTE

   SEE ALSO
      RawDoFmt()
```

## 1.14 netsupport.library/PClose

```
   NAME
      PClose -- closes a pipe previously opened with POpen()

   SYNOPSIS
      returncode = PClose(fh);
      D0                  d0

      LONG    returncode;
      BPTR    fh;

   FUNCTION
      This routine closes a pipe previously created with POpen() and
      returns the returncode of the executed command.

   INPUTS
      fh = filehandle returned by POpen()

   RESULT
      Code returned by the executed command.

   BUGS
      Until NetSupport.Library version 1.27, PClose() did not catch the
      returncode of the executed program. This has been fixed.

   SEE ALSO
      POpen()
```

## 1.15 netsupport.library/POpen

```
   NAME
      POpen -- starts a command and redirects its input stream to a
```

```
            pipe, whose file handle is returned.

    SYNOPSIS
       fh = POpen(command, mode);
       D0         A0       D0

       BPTR    fh;
       STRPTR  command;
       LONG    mode;

    FUNCTION
       This is an equivalent of the UNIX popen() routine, designed for
       AmigaOS. POpen() is usually used to start an external program
       that expects data from the input stream or provides data via the
       output stream. The command is started asynchrony, of course.

    INPUTS
       command = String with the command to be started. The command
                 should not contain any i/o redirectors like "<file",
                 although it will work. Additionally, you should not
                 start a command via POpen("run ...", MODE_PIPETO) either.

       mode = This is the mode the pipe should be opened in. MODE_PIPETO
                 will start the command to read from the pipe and
                 MODE_PIPEFROM will start the command with its output re-
                 directed to the returned pipe-handle.

    RESULT
       The routine returns the file handle of the opened pipe, which you
       can use for either reading or writing, depending on the mode you
       opened the pipe with. If an error occurs, NULL will be returned.

    EXAMPLE
       The following code will send some dummy letter to the postmaster
       of your UUCP/inet site:

                BPTR fh;
                STRPTR sendmail = GetConfig(NULL, SENDMAIL, NULL, SENDMAIL);

                if (fh = POpen(sendmail, MODE_PIPETO)) {
                    FPrintf(fh, "To: postmaster\nSubject: Bug report\n\n");
                    FPuts(fh, "Sorry to bother you, but error #1 occurred\n);
                    PClose(fh);
                }


    SEE ALSO
       PClose()
```

## 1.16 netsupport.library/SetConfigEntry

```
    NAME
       SetConfigEntry -- set a configuration entry
```

```
SYNOPSIS
   success = SetConfigEntry(filename, keyword, parameter);
   D0                       A0        A1       A2

   LONG    success;
   STRPTR  filename;
   STRPTR  keyword;
   STRPTR  parameter;

FUNCTION
   This routine can be used to set a certain entry in a config
   file. SetConfigEntry() will load the file and look for the
   specified keyword. If the keyword can be found, it will replace
   the parameter with the provided one and write the file back.

   When the keyword couldn't be found, SetConfigEntry() will append
   the keyword plus parameter at the end of the file.

   Please read GetConfigEntry() for a brief description of the
   configfile format.

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the configfile in case no variable is set.

           When <filename> is NULL, the routine uses the configfile,
           you specified as 'MasterConfig' in the library's own
           configfile. The default is 'UULib:Config'.

   keyword - Pointer to a keyword string. All comparisons are
           case-independent!

   parameter - pointer to a string containing the parameter to be
           set behind the keyword in the configfile.

RESULT
   SetConfigEntry() either TRUE (!0L) or FALSE (0L) for success/
   failure.

NOTE
   The specified configfile MUST at least exist or the routine will
   fail!

   SetConfigEntry() does NOT lock the file in order to avoid
   deadlocks.

SEE ALSO
   GetConfigEntry(), GetConfig()
```

## 1.17 netsupport.library/TempName

```
NAME
   TempName -- build an unique temporary filename

SYNOPSIS
```

```
    tmpname = TempName(buffer);
    D0                A0

    STRPTR tmpname;
    STRPTR buffer;

FUNCTION
    TempName() builds an unique filename you can use for your
    temporary files.

INPUTS
    buffer - Pointer to an buffer to hold the filename. The
                    generated filename will not succeed 12
                    characters. If NULL is specified, TempName()
                    will use an internal buffer and return a pointer
                    to this one. Please take note, that the internal
                    buffer will be overwritten the next time you
                    call TempName(), so copying its contents might be
                    a good idea.

RESULT
    The returned result is a pointer to a buffer holding the name of
    the unique filename. The filename will look like this:
    "tmpXXXXXXXX", where the 'X-part' will be a hexadecimal number
    like "tmp0000a214", for example.

    If an error occurs (out of memory), NULL will be returned. If
    you provide the buffer, the routine CANNOT fail!

NOTES
    These filenames are meant for temporary files and you should use
    them only for temporary ones. The number part (which make the
    name unique) will be flushed to zero, every time the library is
    removed from memory. If you want to generate unique filenames
    which won't be overwritten after your program terminates, use
    GetSeq() and similar routines to build the filename.

    If you want the filename to contain the "T:" path for temporary
    files, call TempNameT(), which will result in "T:tmpXXXXXXXX".

EXAMPLE
    Though it doesn't really matter, you might want to place the
    temporary files in certain directories or let them reflect the
    program, they belong to. This can be archives as follows:

            char tmpname[32];

            strcpy(tmpname, "SYS:foobar_");
            strcat(tmpname, TempName(NULL));

    This code will generate a filename like "SYS:foobar_tmp00000001".

SEE ALSO
    TempNameT(), GetSeq()
```

## 1.18 netsupport.library/TempNameT

```
NAME
    TempNameT -- build an unique temporary filename including path

SYNOPSIS
    tmpname = TempNameT(buffer);
    D0                  A0

    STRPTR tmpname;
    STRPTR buffer;

FUNCTION
    TempNameT() builds an unique path and filename you can use for
    your temporary files.

INPUTS
    buffer - Pointer to an buffer to hold the filename. The
                    generated filename will not succeed 14
                    characters. If NULL is specified, TempNameT()
                    will use an internal buffer and return a pointer
                    to this one. Please take note, that the internal
                    buffer will be overwritten the next time you
                    call TempNameT(), so copying its contents might
                    be a good idea.

RESULT
    The returned result is a pointer to a buffer holding the name of
    the unique filename. The filename will look like this:
    "T:tmpXXXXXXXX", where the 'X-part' will be a hexadecimal number
    like "T:tmp0000a214", for example.

    If an error occurs (out of memory), NULL will be returned. If
    you provide the buffer, the routine CANNOT fail!

NOTES
    These filenames are meant for temporary files and you should use
    them only for temporary ones. The number part (which make the
    name unique) will be flushed to zero, every time the library is
    removed from memory. If you want to generate unique filenames
    which won't be overwritten after your program terminates, use
    GetSeq() and similar routines to build the filename.

EXAMPLE
    Though it doesn't really matter, you might the filename to
    reflect the program, it belongs to:

            char tmpname[32];

            strcpy(tmpname, TempNameT(NULL));
            strcat(tmpname, ".foobar");

    This code will generate a filename like "T:tmp00000001.foobar".

SEE ALSO
    TempName(), GetSeq()
```

## 1.19 netsupport.library/UnLockFile

```
NAME
   UnLockFile -- free an previously locked file

SYNOPSIS
   UnLockFile(filename);
              A0

   STRPTR  filename;

FUNCTION
   UnLockFile() releases a lock on a file, previously obtained
   using LockFile().

INPUTS
   filename - Pointer to a string, containing the path and filename
           to access the file.

RESULT
   none

SEE ALSO
   LockFile(), LockFileAttempt(), UnLockFiles()
```

## 1.20 netsupport.library/UnLockFiles

```
NAME
   UnLockFiles -- free all files locked by a process

SYNOPSIS
   UnLockFiles();

BACKGROUND
   Please refer to LockFile() for a brief description.

FUNCTION
   UnLockFiles() releases all filelocks obtained the by calling
   process. This can be useful in error situations.

INPUTS
   none

RESULT
   none

SEE ALSO
   LockFile(), LockFileAttempt(), UnLockFile()
```