nonvolatile

COLLABORATORS			
	TITLE : nonvolatile		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 19, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

nonvolatile

1.1 nonvolatile.doc

```
--background--
DeleteNV()
FreeNVData()
GetCopyNV()
GetNVInfo()
GetNVList()
SetNVProtection()
StoreNV()
```

1.2 nonvolatile.library/--background--

```
 PURPOSE
The nonvolatile library provides a simple means for an application
developer to manage nonvolatile storage.

 OVERVIEW
The nonvolatile library is meant to be used transparently across all
configurations. Currently, nonvolatile storage may consist of NVRAM
and/or disk devices. nonvolatile.library will automatically
access the best nonvolatile storage available in the system. Disk
based storage will be selected first and if not available, NVRAM
storage will be accessed.

* NVRAM

On low-end diskless Amiga platforms, NVRAM may be available. This
RAM will maintain its data contents when the system is powered down.
This is regardless of whether batteries or battery-backed clock are
present. The data stored in NVRAM is accessible only through the
ROM-based nonvolatile library funtion calls. The size of NVRAM
storage is dependant on the system platform and is attainable through
the GetNVInfo() function.

* Disk
```

```
In keeping with the general configurability of the Amiga, the actual
disk location used by nonvolatile library when storing to disk may be
changed by the user.

The prefs directory is used on the Amiga for storing many user
configurable options. The location for nonvolatile disk storage
is contained in the file prefs/env-archive/sys/nv_location. This
file should contain a data string that specifies a lockable location.
If the string does not specify a lockable location, the file will
be ignored.

When opened, the nonvolatile library will search all drives within
the system until it finds this file and successfully accomplishes
a Lock on the location specified in the file. To force a rescan of
all drives, the library may be closed and reopened or execute the
GetNVInfo() function.

A simple method for creating a floppy disk for saving nonvolatile
data is the following:

Format a disk with the volume name NV
Create a file prefs/env-archive/sys/nv_location on this disk with
  the following contents:  NV:nonvolatile
Create a directory nonvolatile

The following is a script file that can be used to make a floppy
for use with nonvolatile library:

.KEY DRIVE/A,DISK
.BRA {
.KET }
format Drive {DRIVE} Name {DISK$NV} noicons ffs
makedir {DRIVE}prefs
makedir {DRIVE}nonvolatile
makedir {DRIVE}prefs/env-archive
makedir {DRIVE}prefs/env-archive/sys
echo {DISK$NV}:nonvolatile >{DRIVE}prefs/env-archive/sys/nv_location

!!!NOTE!!!

Because NVRAM performs disk access, you must open and use its
functionality from a DOS process, not an EXEC task.  Normally
your CDGS application is invoked as a DOS process so this
requirement generally should cause you no concern.  You just
need to be aware of this requirement should you create an
EXEC task and try to invoke nonvolatile.library from that task.
```

1.3 nonvolatile.library/DeleteNV

```
 NAME
DeleteNV -- remove an entry from nonvoltatile storage. (V40)

 SYNOPSIS
success = DeleteNV(appName, itemName, killRequesters);
D0       A0     A1        D1
```

```
BOOL DeleteNV(STRPTR, STRPTR, BOOL);

 FUNCTION
Searches the nonvolatile storage for the indicated entry and removes
it.

The strings appName and itemName may not contain the '/' or ':'
characters. It is recommended that these characters be blocked
from user input when requesting AppName and ItemName strings.

 INPUTS
appName - NULL terminated string identifing the application that
    created the data. Maximum length is 31.
ItemName - NULL terminated string uniquely identifing the data
     within the application. Maximum length is 31.
killRequesters - suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
success - TRUE will be returned if the entry is found and deleted.
    If the entry is not found, FALSE will be returned.
```

1.4 nonvolatile.library/FreeNVData

```
 NAME
FreeNVData -- release the memory allocated by a function of this
        library. (V40)

 SYNOPSIS
FreeNVData(data);
     A0

VOID FreeNVData(APTR);

 FUNCTION
Frees a block of memory that was allocated by any of the following:
GetCopyNV(), GetNVInfo(), GetNVList().

 INPUTS
data - pointer to the memory block to be freed. If passed NULL,
       this function does nothing.

 SEE ALSO
GetCopyNV(), GetNVInfo(), GetNVList()
```

1.5 nonvolatile.library/GetCopyNV

```
 NAME
GetCopyNV -- return a copy of an item stored in nonvolatile storage.
       (V40)
```

```
 SYNOPSIS
data = GetCopyNV(appName, itemName, killRequesters);
D0     A0   A1      D1

APTR GetCopyNV(STRPTR, STRPTR, BOOL);

 FUNCTION
Searches the nonvolatile storage for the indicated appName and
itemName. A pointer to a copy of this data will be returned.

The strings appName and itemName may not contain the '/' or ':'
characters. It is recommended that these characters be blocked
from user input when requesting appName and itemName strings.

 INPUTS
appName - NULL terminated string indicating the application name
    to be found. Maximum length is 31.
itemName - NULL terminated string indicated the item within the
     application to be found. Maximum length is 31.
killRequesters - Suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
data - pointer to a copy of the data found in the nonvolatile storage
       assocated with appName and itemName. NULL will be returned
       if there is insufficient memory or the appName/itemName does
       not exist.

 SEE ALSO
FreeNVData(), <libraries/nonvolatile.h>
```

1.6 nonvolatile.library/GetNVInfo

```
 NAME
GetNVInfo -- report information on the current nonvolatile storage.
       (V40)

 SYNOPSIS
information = GetNVInfo(killRequesters);
D0      D1

struct NVInfo *GetNVInfo(BOOL);

 FUNCTION
Finds the user's preferred nonvolatile device and reports information
about it.

 INPUTS
killRequesters - suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
```

```
information - pointer to an NVInfo structure. This structure contains
        information on the NV storage media with the largest
        storage. The structure contains 2 longword fields:
        nvi_MaxStorage and nvi_FreeStorage. Both values are
        rounded down to the nearest ten. The nvi_MaxStorage
        field is defined as the total amount of nonvolatile
        storage available on this device. The nvi_FreeStorage is
        defined as the amount of available space for NVDISK or
        the amount of non-locked storage for NVRAM. For NVDISK,
        the nvi_FreeStorage takes into account the amount of
        overhead room required to store a new App/Item. This
        amount is 3 blocks to allow room for storing a new Item
        file and possibly a new App directory. For NVRAM, the
        amount of overhead is 5 bytes. However, the amount of
        room required to store a new NVRAM item depends on the
        length of the App and Item names. Refer to StoreNV()
        function for storage details.

        This function may return NULL in the case of failure.

 SEE ALSO
FreeNVData(), StoreNV(), <libraries/nonvolatile.h>
```

1.7 nonvolatile.library/GetNVList

```
 NAME
GetNVList -- return a list of the items stored in nonvolatile
       storage. (V40)

 SYNOPSIS
list = GetNVList(appName, killRequesters);
D0     A0   D1

struct MinList *GetNVList(STRPTR, BOOL);

 FUNCTION
Returns a pointer to an Exec list of nonvolatile Items associated
with the appName requested.

The string appName may not contain the '/' or ':' characters.
It is recommended that these characters be blocked from user input
when requesting an appName string.

 INPUTS
appName - NULL terminated string indicating the application name
    to be matched. Maximum length is 31.
killRequesters - Suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
list - a pointer to an Exec MinList of NVEntries. A NULL will be
       returned if there is insufficient memory. If there are no
       entries in the nonvolatile storage for the AppName, an
       empty list will be returned.
```

```
 NOTE
The protection field contains more bits than are required for
storing the delete protection status. These bits are reserved
for other system usage and may not be zero. When checking for
the delete status use either the field mask NVIF_DELETE, or the
bit definition NVIB_DELETE.

 SEE ALSO
FreeNVData(), SetNVProtection()
```

1.8 nonvolatile.library/SetNVProtection

```
 NAME
SetNVProtection -- set the protection flags. (V40)

 SYNOPSIS
success = SetNVProtection(appName, itemName, mask, killRequesters);
D0        A0     A1      D2    D1

BOOL SetNVProtection(STRPTR, STRPTR, LONG, BOOL);

 FUNCTION
Sets the protection attributes for an item currently in the
nonvolatile storage.

Although 'mask' is LONG only the delete bit, NVEF_DELETE/NVEB_DELETE,
may be set. If any other bits are set this function will return
FALSE.

The strings appName and itemName may not contain the '/' or ':'
characters. It is recommended that these characters be blocked
from user input when requesting AppName and ItemName strings.

 INPUTS
appName - NULL terminated string indicating the application name
    to be matched. Maximum length is 31.
itemName - NULL terminated string indicated the item within the
     application to be found. Maximum length is 31.
mask - the new protection mask. Only set the delete bit otherwise
       this function WILL CRASH.
killRequesters - suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
success - FALSE if the protection could not be change (ie the data
    does not exist).

 SEE ALSO
GetNVList(), <libraries/nonvolatile.h>
```

1.9 nonvolatile.library/StoreNV

```
 NAME
StoreNV -- store data in nonvolatile storage. (V40)

 SYNOPSIS
error = StoreNV(appName, itemName, data, length, killRequesters);
D0    A0   A1    A2    D0  D1

UWORD StoreNV(STRPTR, STRPTR, APTR, ULONG, BOOL);

 FUNCTION
Saves some data in nonvolatile storage. The data is tagged with
AppName and ItemName so it can be retrieved later. No single
item should be larger than one fourth of the maximum storage as
returned by GetNVInfo().

There is no data compression associated with this function.

The strings, AppName and ItemName, should be short, but descriptive.
They need to be short since the string is stored with the data and
the nonvolatile storage for a stand alone game system is limited.
The game system allows the user to selectively remove entries from
storage, so the string should be desriptive.

The strings AppName and ItemName may not contain the '/' or ':'
characters. It is recommended that these characters be blocked
from user input when requesting AppName and ItemName strings.

 INPUTS
appName - NULL terminated string identifying the application
    creating the data. Maximum length is 31.
itemName - NULL terminated string uniquely identifying the data
     within the application. Maximum length is 31.
data - pointer to the memory block to be stored.
length - number of bytes to be stored in the units of tens of
   bytes. For example, if you have 23 bytes to store length = 3;
   147 byte then length = 15.
killRequesters - suppress system requesters flag. TRUE if all system
     requesters are to be suppressed during this function.
     FALSE if system requesters are allowed.

 RESULT
error - 0                means no error,
        NVERR_BADNAME    error in AppName, or ItemName.
        NVERR_WRITEPROT  Nonvolatile storage is read only.
        NVERR_FAIL       Failure in writing data (nonvolatile storage
       full, or write protected).
        NVERR_FATAL      Fatal error when accessing nonvolatile
       storage, possible loss of previously saved
       nonvolatile data.

 SEE ALSO
GetCopyNV(), GetNVInfo()
```