

Handling Multiple Assigns with Conventional Directories

One of the features introduced by Release 2 is Multiple Assigning. This feature allows an AmigaDOS assign to carry over several directories which can be on different volumes. This makes it possible to split up assigns such as *libs:* and *fonts:*.

The article “Directory Scanning” on page II-49 contains an example called *find.c* that illustrates scanning a path that can contain a multiassign. However, besides being rather complicated, *find.c* makes a special case of scanning assigns, which isn’t necessary (*find.c* also did something evil--*find.c* uses DOSBase’s private pointer to the *utility.library*, essentially using the *utility.library* without opening it). The method needed to scan a multiassign directory also works on conventional directories.

Scanning a multiassign requires calling the *dos.library* function `GetDeviceProc()` in a loop to see each directory of the multiassign. Using `GetDeviceProc()`, the application doesn’t have to concern itself with the differences between assigns, multiassigns, and volumes. The application just keeps calling `GetDeviceProc()` until it gets back a `NULL`.

```
struct DevProc *GetDeviceProc( STRPTR name, struct DevProc *dp );
```

The name can be any valid dos path. If there is a device name present, `GetDeviceProc()` will find the device’s entry in the dos list and copy some information into a `DevProc` structure:

```
struct DevProc {
    struct MsgPort *dvp_Port; /* Device’s Message port, also called a Process identifier */
    BPTR          dvp_Lock; /* Lock on root of assign or lock on root of volume */
    ULONG         dvp_Flags;
    struct DosList *dvp_DevNode; /* DON’T TOUCH OR USE! */
};
```

The important fields here are `dvp_Lock` and `dvp_Port`. The `dvp_Lock` field is a lock on the root of the object named in `GetDeviceProc()`. It serves as a starting point in locating the named object. If the object name contains an assign (i.e. “*libs:*”), `dvp_Lock` is the root of the assign. For example, on a typical Release 2 system, the *libs:* assign refers to the *libs* directory on the *System2.0:* volume. Calling `GetDeviceProc()` on “*libs:*” in this case will yield a lock on *System2.0:libs*.

If the named object contains a dos volume, `dvp_Lock` is either a lock on the root of the dos volume or `NULL`. If the object named in `GetDeviceProc()` contains a non-filesystem device (i.e., “*ser:*”, “*par:*”, “*pri:*”, etc.) or it does not contain a device name, `dvp_Lock` is `NULL`.

The `dvp_Port` field points to a message port. This message port is connected to the handler process of a DOS device. The handler process controls a DOS device. DOS functions (like the `Lock()` function) use this message port to talk to the handler process of the named object. For example, from the “`libs:`” example above, the `dvp_Port` field refers to the message port of the handler process for the `System2.0:` volume.

Note that `dvp_Lock` is only a lock on the root of the named object. If the named object is a path several directories deep (for example, `libs:gadgets/colorwheel.gadget`), it’s up to the application to handle the rest of the path. The application also has to handle the case where the named object is a path without a device name.

Although an application can send DOS packets directly to the message port (`dvp_Port`) of a handler process, normally it is easier to use functions from `dos.library`. The `multilist.c` example uses the `Lock()` function to lock the named object. `Multilist` has to do something a little unorthodox to use `Lock()`. `Lock()` accepts a path name to the object to lock. `Lock()` understands absolute paths (i.e. paths with a logical device name like “`df1:`” or “`libs:`”) and relative paths. If `Lock()` receives an absolute path name, `Lock()` can find the device’s handler process using the logical device name in the absolute path. For a relative path, `Lock()` does not have enough information to find the named object, so it assumes the path is relative to the current directory and file system (each process has a current directory and file system).

This makes `Lock()` a little more difficult to use in `multilist.c` because, when processing an absolute path, `multilist` has to process the logical device name separately from the rest of the path. It has to use `GetDeviceProc()` to find the root of a logical device name (which can be an assign, multiassign, volume name, etc.) then it has to strip the logical device name from the absolute path. Without a logical device name, the path has become relative rather than absolute. The path is now relative to `dvp_Lock` and `dvp_Port`. In order for `Lock()` to work with this relative path, `multilist` must temporarily set the current directory and file system to the values in `dvp_Lock` and `dvp_Port`, respectively.

Note that the Autodoc for `GetDeviceProc()` says to check `IoErr()` for `ERROR_NO_MORE_ENTRIES` after receiving a `NULL` from `GetDeviceProc()`. Due to a bug, DOS does not set the error value correctly in all cases. Also note that the Autodoc says to check the `DevProc` structure’s `dvp_Flags` field for the `DVPF_ASSIGN` flag. This was necessary in the 2.00 and 2.01 releases of the operating system due to a bug in DOS, but is no longer necessary.

The following example, `multilist.c`, accepts an arbitrary path name and lists the contents of it. The function `DoAllAssigns()` does all of the multiassign work. `DoAllAssigns()` accepts a path and a function pointer. It gets a lock on the object named in the path, and passes the lock to the function. This example is based on a Usenet posting by Randell Jesup.

```

/* multilist.c -- execute me to compile me
sc data=near nominc strmer streq nostkchk saveds ign=73 multilist
slink FROM LIB:c.o multilist.o TO multilist LIB LIB:sc.lib LIB:amiga.lib
quit
*/

/* This example illustrates how to scan DOS file names without */
/* having to make a special case for assigns and multiassigns. */
/* The DoAllAssigns() routine accepts an arbitrary dos path and */
/* a pointer to a function. DoAllAssigns() will call this function */
/* passing it a lock to the object named in the path. */

#define BUFSIZE 1024

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dosextens.h>
#include <dos/exall.h>

#include <clib/dos_protos.h>
#include <clib/exec_protos.h>
#include <stdio.h>
#include <strings.h>

BOOL DoAllAssigns(char *, BOOL (*)());
BOOL ListContents(BPTR);

extern struct DosLibrary *DOSBase;

void main(int argc, char **argv)
{
    if (DOSBase->dl_lib.lib_Version >= 37)
    {
        if (argc > 1)
        {
            (void) DoAllAssigns(argv[1], &ListContents);
        }
    }

    /* Pass this routine a directory lock and it prints the names of the */
    /* files and directories in that directory. If you pass this routine */
    /* a file lock, it just prints the file's name. */
    BOOL ListContents(BPTR lock)
    {
        struct ExAllControl *myeac;
        struct ExAllData *myead;
        APTR buffer;
        BOOL done;
        struct FileInfoBlock *myfib;

        if (myfib = AllocDosObject(DOS_FIB, NULL))
        {
            if (Examine(lock, myfib) == DOSTRUE)
            {
                if (myfib->fib_DirEntryType > 0)
                {
                    if (buffer = AllocVec(BUFSIZE, MEMF_PUBLIC))
                    {
                        if (myeac = AllocDosObject(DOS_EXALLCONTROL, NULL))
                        {
                            myeac->eac_LastKey = 0;

                            do
                            {
                                done = ExAll(lock, buffer, BUFSIZE, ED_NAME, myeac);
                                myead = buffer;
                                while (myead)
                                {
                                    printf("%s\n", myead->ed_Name);
                                    myead = myead->ed_Next;
                                }
                            } while (done != 0);
                        }
                    }
                }
            }
        }
    }
}

```

Amiga Mail

Volume II

```
        FreeDosObject(DOS_EXALLCONTROL, myeac);
    }
    FreeVec(buffer);
}
}
else printf("%s\n", myfib->fib_FileName);
}
FreeDosObject(DOS_FIB, myfib);
}
return TRUE;
}

/* This routine accepts a path string that may include a device name. From
/* that string, this routine locks the object named in the path and calls
/* the function passback_func() on the lock. DoAllAssigns() should work on
/* paths with assigns and multiassigns, as well as a filesystem-based device
/* (i.e. df0:, work:, ram:, etc.)
*/

BOOL DoAllAssigns(char *dos_path, BOOL (*passback_func)(BPTR lock))
{
    struct DevProc *dp=NULL;
    struct MsgPort *old_fsport;
    BPTR lock, old_curdir;
    char *rest_of_path;
    LONG err;

    while(dp = GetDeviceProc(dos_path, dp))
    {
        rest_of_path = strchr(dos_path, ':'); /* I need to cut the device name from
        /* the front of dos_path so I can give
        /* that substring to Lock().
        /* There was no device name to
        /* cut off, use the whole string.
        rest_of_path = dos_path;
    else
        rest_of_path++; /* Increment string pointer to just past the colon.

    old_fsport = SetFileSysTask(dp->dvp_Port); /* in case dp->dvp_Lock is NULL.
    old_curdir = CurrentDir(dp->dvp_Lock); /* Lock() locks relative to the
        /* current directory and falls back to the root of
        /* the current file system if dp->dvp_Lock is NULL.

    lock = Lock(rest_of_path, SHARED_LOCK);

    (void) SetFileSysTask(old_fsport); /* reset the process' default filesystem
    (void) CurrentDir(old_curdir); /* port and current dir to their initial
        /* values for clean up later.

    if (lock)
    {
        if (!(*passback_func)(lock))
        {
            printf("function returned false\n");
            UnLock(lock); /* UnLock() will ignore NULL lock
            FreeDeviceProc(dp);
            return FALSE;
        }
        UnLock(lock);
    }
}
err = IoErr();

if ((err == ERROR_NO_MORE_ENTRIES) || (err == 0))
    return TRUE; /* At present, a bug in DOS prevents this case,
else
    return FALSE; /* so DoAllAssigns() always returns FALSE.
}
}
```

