Technote 1103

Uniquely Identifying a Mac OS Computer

CONTENTS

There is no single unique serial number available across the entire range of Mac OS based computers. There are some techniques which can be used to help differentiate between two Mac OS computers. These techniques are described in this technote.

One use of such unique identification is copy protection. DTS does not support copy protection because of the significant compatibility liabilities it involves. This technote is a summary of some specific techniques which we have used or recommended in the past. DTS does not maintain expertise in copy protection techniques. Serious copy protection is much more complicated than this technote describes.

Uniquely Identifying a Mac OS Computer

In general, the only consistent serial number on a Mac OS based computer is on the bar code label which is attached to the outside of the case. There is no internal serial number on the logic board of any current Mac OS based computer. There is no internal serial number associated with the operating system. The only Macintosh with a real built-in serial number was the Macintosh XL. This machine had a serial number only because the Lisa (upon which it was based) had a serial number as part of the Lisa design. Since the Macintosh XL was discontinued in August of 1986, it is very rare, and not very relevant in today's market.

There are some hardware devices, found on a limited range of Mac OS computers, which contain some unique identifier which could be used as a serial number. Examples are an installed Ethernet card or built-in Ethernet. (This is because Ethernet devices have an unique ID defined for each device.) Some (but not all) hard drives have a serial number which can be read using the SCSI Manager or ATA Manager. Other devices such as Token Ring cards or FireWire may also provide a unique identifier which may be used as a serial number. None of these hardware devices can be found across the entire Macintosh product line.

Semi-Unique Characteristics

There are some "semi-unique" characteristics which you can use to help you determine if you are running software on the same machine which you were running on before. These are listed roughly in the order I'd recommend considering them.

File ID References

You can create and store the file ID reference for the application file. Create a file ID reference using `PBCreateFileIDRef()` and store it in an appropriate place. (An example of an appropriate place would be the preferences file. An inappropriate place would be the application resource fork, since this would prevent your application from running from locked media or a network server.) Compare the file ID reference to the file ID reference of the running application (the file ID reference will be in the `ioFileID` field returned by `PBGetCatInfo`). If it isn't the same, you are running on a different volume. File ID references are unique within a particular volume; they don't get reused. It's unlikely that two machines will give you the same file ID reference for the same file.

```
// AddFileID creates a file ID reference for the
// file specified by the FSSpec. It returns the
// created file ID reference so that you can store
// this reference for future use.
//
OSErr   AddFileID(FSSpec *file, long *fileID)
{
        OSErr                   err;
        HParmBlkPtr             h;

        h = (HParmBlkPtr)NewPtrClear(sizeof(HParamBlockRec));

        h->fidParam.ioCompletion = nil;
        h->fidParam.ioNamePtr = file->name;
        h->fidParam.ioVRefNum = file->vRefNum;
        h->fidParam.ioSrcDirID = file->parID;

        err = PBCreateFileIDRefSync(h);
        *fileID = h->fidParam.ioFileID;
     DisposePtr((void *)h);
        return(err);
}

// GetFileID returns the File ID reference for a file
// where the File ID reference has previously been
// created (by calling PBCreateFileIDRef)
//
OSErr   GetFileID(FSSpec *file, long *fileID)
{
        OSErr                   err;
        CInfoPBPtr              cInfo;

        cInfo = (CInfoPBPtr)NewPtrClear(sizeof(CInfoPBRec));

        cInfo->hFileInfo.ioCompletion = nil;
        cInfo->hFileInfo.ioNamePtr = file->name;
        cInfo->hFileInfo.ioVRefNum = file->vRefNum;
        cInfo->hFileInfo.ioFDirIndex = 0;
        cInfo->hFileInfo.ioDirID = file->parID;

        err = PBGetCatInfoSync(cInfo);
        *fileID = cInfo->hFileInfo.ioDirID;
        DisposePtr((void *)cInfo);
        return(err);
}
```

Gestalt

The Gestalt Manager gives you access to many pieces of information which may help you uniquely identify a machine. The obvious choices to help uniquely identify a machine are the machine type (represented by the `gestaltMachineType` selector) and the kind of keyboard present (represented by the `gestaltKeyboardType` selector.)

```
OSErr GetMachineAndKeyboardIDs(long *characteristics)
{
    long    response;
    OSErr   err;

    err = Gestalt(gestaltMachineType, &response);
    *characteristics = response << 16;
    if (!err)
        err = Gestalt(gestaltKeyboardType, &response);
    *characteristics |= response;
    return err;
}
```

Be aware that new revisions of System Software may change Gestalt selectors in ways you may not have anticipated.

Ethernet Address

For those Mac OS computers which have an Ethernet card installed or built-in Ethernet, you can use the `EGetInfo()` call with a csCode of `ENetGetInfo` to get the currently assigned Ethernet address. See *Inside Macintosh: Networking* , page 11-26 and 11-36 for more information. Apple publishes sample code demonstrating how to obtain the Ethernet address under Open Transport.

The complication for this technique is that the default Ethernet address may be overridden by a resource of type 'eadr' in the System file. This is documented in *Inside Macintosh: Networking* on page 11-26. Because it is easily possible to override the hardware address, and because Ethernet is not guaranteed to exist on any particular model, this is not a good scheme.

Similar techniques can be used for a Mac OS computer with Token Ring cards or other cards, but a discussion of these cards is outside the scope of this document.

Hard Disk Serial Number

Some hard disks have serial numbers. Many ATA/IDE drives have such a serial number, but most SCSI hard disks do not. Apple publishes ATA demo sample code which shows how to get the serial number of an ATA or IDE drive. Only some Macintosh models have ATA or IDE drives, so this is not a good general purpose scheme.

SCSI Defect List

Some developers use the defect list from a SCSI drive. This won't work for a non-SCSI machine, but might be a good approach for a SCSI based Mac. See the SCSI-2 specification and *Inside Macintosh:Devices* for details of using the SCSI Manager. Source code demonstrating use of the SCSI Manager in a general case is on the tool chest developer CD. Only some Macintosh models have SCSI drives, so this is not a good general purpose scheme.

Directory ID of the System Folder

Compare the dirID of the System Folder. Use `FindFolder()` to get the dirID of the System Folder, and compare it to a previously stored dirID. This isn't quite as unique as the file ID, since the system folder as preinstalled will tend to have the same directory ID from one machine to another. If the user creates a new System Folder (e.g. by doing a clean install of System Software, which appears to be a frequent activity when upgrading) then you will have different directory IDs. This is not a good scheme.

```
// GetVolumeDirID returns the dirID of the System Folder
//
OSErr   GetVolumeDirID(long *dirID)
{
        OSErr                   err;
        short                   notUsed;

        err=FindFolder(kOnSystemDisk, kSystemFolderType, no,
                notUsed,dirID);
        return(err);
}
```

Volume Creation Date

Compare the creation date of the volume. You can get this information from `PBHGetVInfo()`. This isn't quite as unique as the file ID, since the volume creation date will be set at the factory when system software is placed on the volume and will only be reset when a volume is reinitialized. Because this value tends to remain the same for a given set of machines, this is not a good scheme.

```
long    GetVolCreationDate(short vRefNum )
{
        OSErr           err = noErr;
        HParamBlockRec  pb;
        Str255          vName;

        vName [ 0 ] = '\0';
        pb.volumeParam.ioCompletion  = nil;
        pb.volumeParam.ioNamePtr     = vName;
        pb.volumeParam.ioVRefNum     = vRefNum;
        pb.volumeParam.ioVolIndex    = 0;
        err = PBHGetVInfoSync ( &pb );

        return ( pb.volumeParam.ioVCrDat);
}
```

Network Registration

Rather than identifying a unique Macintosh, you may decide that you want to prevent multiple copies of the same application running on a network. The method some developers use is to register a fictitious device on the network using NBP (Name Binding Protocol) with the name being the single serial number of the license. (Of course, you still need a way of generating that serial number.) Other attempts to register the same device and serial number give an error that the program acts on to deny the use of the program. Chapter 3 of *Inside Macintosh: Networking* is a useful reference for NBP.

```
/*
 *  Registers an entity with the specified object and type on the
 *  specified socket. The pointer to the NamesTableEntry is
 *  returned in ntePtr if the function returns noErr.
 *
 */
OSErr MyRegisterName(ConstStr32Param entityObject,
                     ConstStr32Param entityType,
                     short socket,
                     NamesTableEntry **ntePtr)
{
    MPPParamBlock mppPB;
    OSErr   result;
    Str32   entityZone = "\p*";

    /* Allocate non-relocatable memory in the system heap for
     * the names table entity
     */
    *ntePtr = (NamesTableEntry *)
        NewPtrSys((Size)sizeof(NamesTableEntry));

    if ( ntePtr == NULL )
    {
        result = MemError(); /* Return memory error */
    }
    else
    {
        /* Build the names table entity */
        NBPSetNTE((Ptr) *ntePtr,
                  (Ptr) entityObject,
                  (Ptr) entityType,
                  (Ptr) entityZone,
                  socket);

        /* ioRefNum and csCode are filled in by
         * PRegisterName's glue */
        mppPB.NBPinterval = 0x0f; /* Reasonable values for the */
        mppPB.NBPcount = 0x03;    /* interval and retry count */
        mppPB.NBPentityPtr = (Ptr) *ntePtr;
        mppPB.NBPverifyFlag = (char) true; /* unique name */

        result = PRegisterName(&mppPB, false);

        if ( result != noErr )
            DisposePtr((Ptr) *ntePtr);
    }

    return ( result );
}
```

Note:
DTS advises against applications searching the entire internet for a matching entity. Such a search is time consuming. On a large network with many zones, you can spend substantial amounts of time doing this search. This would be not be considered reasonable in startup code. Instead, we advise that you search the local zone. If necessary, implement an asynchronous background search into the other zones.

Things You Shouldn't Do

You should not rely on undocumented values with extended Parameter RAM (PRAM). Apple has only documented a limited portion of PRAM and will not document other parts; see *Inside Macintosh:Operating System Utilities* chapter 7 for details. Apple reserves the right to modify the meaning of any PRAM values not documented. Do not attempt to store information into PRAM beyond the documented areas. Such misuse of PRAM may result in extreme system instability.

We discourage you from trying to use special tracks on formatted floppies, or special floppies. Apple does not document the floppy drive sufficiently for DTS to support such an action. Also, DiskCopy and other disk copying programs work very well at copying floppies, thus defeating such schemes. (DiskCopy was written inside Apple with access to the source of the floppy driver; we do not publish these details externally.) You should not rely on specific bizarre sectors of the hard disk (Apple relies on multiple vendors for its components. You cannot make undocumented assumptions about a particular machine or class of machines.)

Note:
At some point in the future, Apple will create machines which do not support 800K GCR-formatted floppies. You should not depend upon specific hardware features such as 800K GCR formatting in order to uniquely identify a Mac OS computer.

Conclusions

The schemes described here are simple ways to help uniquely identify a computer. Such schemes may be useful for simple copy protection. There are many more sophisticated schemes for serious copy protection. Some very good approaches use external hardware, such as ADB devices (called dongles) which help uniquely identify an authorized machine. If you are serious about copy protection, you should probably be contacting one of the many companies which specialize in copy protection solutions, rather than writing a solution yourself. Both hardware solutions (such as ADB dongles) and software solutions (such as licensing software) are widely available from third parties.

Summary

The Mac OS was not designed with copy protection in mind, and there is no unique serial number available across the entire set of Mac OS based computers. There are some techniques which can be used to help differentiate between Mac OS computers, and these techniques are described in this technote.

DTS does not support copy protection because of the significant compatibility liabilities it involves. DTS does not maintain expertise in copy protection techniques.

Further References

- *Inside Macintosh: Devices*
- ATA Device Software Guide
- *Inside Macintosh: Networking* , chapter 3, Name Binding Protocol
- SCSI-2 specification

Downloadables

Acrobat version of this Note (K)

Updated: 9-February-98

Technotes
Previous Technote | Contents | Next Technote