New Technical Notes

Macintosh

Developer Support

Checking for Specific Functionality

Overview **M.OV.ChkForFunction**

Revised by: March 1988
Written by: Jim Friedlander September 1987

This technical note explains how to check at run time to see if specific functionality, such as the "new" TextEdit, is present.

Applications should strive to be compatible across all Macintoshes, but there are times when an application must have knowledge about the machine that it is running on. The new trap, `SysEnvirons`, will give an application most of the information that it requires (what hardware, what version of system software…).

Using SysEnvirons

In most cases, if you examine why you want to test for the existence of a specific trap, you will find that there is an alternative method, for example:

I need to see if the "new" TextEdit calls are available.

Call `SysEnvirons` and check to see that `SysEnvRec.machineType >= 0` (128K ROMs or newer) and that we are running System 4.1 or later (System 4.1 and later support the new TextEdit on 128K and greater ROM machines—we can check this by just seeing if the `SysEnvirons` trap exists, if we get an `envNotPresent` error, we know it doesn't). In Pascal:

```
CONST
      CurrentVersion = 1;  {Current version of SysEnvirons}
VAR
      newTextAvail  : BOOLEAN;
      theWorld      : SysEnvRec;
BEGIN
{
This code checks to see if System 4.1 or later is running by calling
SysEnvirons. If SysEnvirons returns an envNotPresent error, we know that
we are running a system prior to 4.1, so we know we don't have the new
TextEdit. If SysEnvirons doesn't return envNotPresent, we check machine
type to make sure we aren't running on 64K ROMs (note: we assume that
envMachUnknown doesn't have 64K ROMs when we check machineType >= 0)
}
      IF SysEnvirons(CurrentVersion,theWorld) = envNotPresent THEN
            newTextAvail:= FALSE
```

```
        ELSE
                newTextAvail:= (theWorld.machineType >= 0);
END;
```

In C:

```
/* Current version of SysEnvirons */
#define       CurrentVersion      1
{
       Boolean       newTextAvail;
       SysEnvRec     theWorld;

/*
       see comment in the above Pascal
*/
if (SysEnvirons(CurrentVersion,&theWorld) == envNotPresent)
       newTextAvail = false;
       else
              newTextAvail = (theWorld.machineType >= 0);
}
```

I need to see if `PopUpMenuSelect` is implemented.

The same answer as above applies here, since the "new" Menu Manager calls are only implemented in System 4.1 on 128K or larger ROM machines (and, as we found above, `PopUpMenuSelect` has the same trap number as `Rename`, so calling `NGetTrapAddress` won't work on 64K ROMs).

Checking for Specific Functionality

There are rare times when you may feel that it is necessary to test for specific functionality. In order to allow for testing of specific trap functionality, there is an official unimplemented trap. This trap (`$A89F`) is unimplemented on all Macintoshes. To test to see if a particular trap that you wish to use is implemented, you can compare its address with the address of the unimplemented trap. Here are two fragments that show how to check to see if `Shutdown` is implemented. First, Pascal:

```
CONST
 ShutDownTrapNum    = $95;{trap number of Shutdown}
 UnImplTrapNum      = $9F;{trap number of "unimplemented trap"}

VAR
 ShutdownIsImplemented   : BOOLEAN;  {is Shutdown implemented}

BEGIN
{Is Shutdown implemented?}
       ShutdownIsImplemented :=
       NGetTrapAddress(ShutDownTrapNum,ToolTrap) <>
       NGetTrapAddress(UnImplTrapNum,ToolTrap);
END;
```

Here's a C fragment:

```
/*trap number of Shutdown*/
#define       ShutDownTrapNum     0x95
/*trap number of "unimplemented trap"*/
#define       UnImplTrapNum       0x9F

{
       Boolean ShutdownIsImplemented;

       /*Is Shutdown implemented?*/
```

```
        ShutdownIsImplemented =
        NGetTrapAddress(ShutDownTrapNum,ToolTrap) !=
        NGetTrapAddress(UnImplTrapNum,ToolTrap);
}
```

`NGetTrapAddress` is used because it ensures that you will get the correct trap in case there is a `ToolTrap` and an `OSTrap` with the same number. Please note that calling `NGetTrapAddress` does not cause compatibility problems with 64K ROMS. When run on those ROMs, it just becomes a `GetTrapAddress` call. You have to be careful on 64K ROMs—you can't test for `PopUpMenuSelect` (`$A80B`), for example, because it has the same trap number as `Rename`(`$A00B`). The 64K ROM didn't really differentiate between `ToolTraps` and `OSTraps` (there was **no** overlap in trap numbers). So, if you wanted to test for `PopUpMenuSelect`, you would need to first check to make sure you weren't running on 64K ROMs (see below).

You can get the trap number of the trap you wish to test for from *Inside Macintosh* (Appendix C of Volumes I-III and Appendix B of Volume IV). You can tell if the trap is an `OSTrap` or a `ToolTrap` by checking to see if bit 11 in the trap word is set, that is, traps like `$A8xx` (or `$A9xx` or `$AAxx`) that have the "8" component set, are `ToolTraps` and traps that don't (`$A0xx`) are `OSTraps`. The trap number that you pass to `NGetTrapAddress` for `ToolTraps` is the low 10 bits of the trap word (the trap number for `PopUpMenuSelect[$A80B]` is `$00B)`. The trap number that you pass to `NGetTrapAddress` for `OSTraps` is the low 8 bits of the trap word (the trap number for `MoveHHi[$A064]` is `$064`).

`Shutdown` (`$A895`) is just an example of a trap that we might need to check before calling. Most applications won't call `ShutDown`, so this is just an example of how to do the testing.

Further Reference:

- Operating System Utilities
- Assembly Language
- M.OV.GestaltSysenvirons