

NOTE: This Technical Note has been [retired](#). Please see the [Technical Notes](#) page for current documentation.

# Technical Note TB14

## MultiFinder Frequently Asked Questions

### CONTENTS

[How can I tell if WaitNextEvent is implemented?](#)

[How can I tell if the MultiFinder Temporary Memory Allocation calls are implemented?](#)

[How can I tell if my application is running in the background?](#)

[When exactly does juggling take place?](#)

[Can I disable suspend/resume events by passing the appropriate event mask to WNE/GNE/EventAvail?](#)

[Should my application use WaitNextEvent?](#)

[Is there anything else that I can do to be MultiFinder friendly?](#)

[Can I use a debugger with MultiFinder?](#)

[What happened to animated icons under MultiFinder?](#)

[How can I ensure maximal compatibility with MultiFinder?](#)

[References](#)

[Downloadables](#)

This technical note provides answers to some of the more frequently asked questions about MultiFinder. The development name for MultiFinder was Juggler, so the term "juggle" is used in this technical note to denote a context switch.

[Nov 01 1993]

---

## How can I tell if WaitNextEvent is implemented?

Most applications should not need to tell if MultiFinder is running. Most of the time, the application really needs to know something like: "How can I tell if WaitNextEvent is implemented?" Here's a Pascal fragment that demonstrates how to check to see if WaitNextEvent is implemented:

```

FUNCTION TrapAvailable(tNumber: INTEGER; tType: TrapType): BOOLEAN;

    CONST
        UnimplementedTrapNumber = $A89F; {number of "unimplemented trap"}

    BEGIN {TrapAvailable}

        {Check and see if the trap exists.}
        {On 64K ROM machines, tType will be ignored.}

        TrapAvailable := ( NGetTrapAddress(tNumber, tType) <>
                           GetTrapAddress(UnimplementedTrapNumber) );

    END; {TrapAvailable}

FUNCTION WNEIsImplemented: BOOLEAN;

    CONST
        WNETrapNumber = $A860; {trap number of WaitNextEvent}

    VAR
        theWorld      : SysEnvRec; {to check if machine has new traps}
        discardError   : OSErr; {to ignore OSErr return from SysEnvirons}

    BEGIN {WNEIsImplemented}

        { Since WaitNextEvent and HFSDispatch both have the same trap
          number ($60), we can only call TrapAvailable for WaitNextEvent
          if we are on a machine that supports separate OS and Toolbox
          trap tables. We call SysEnvirons and check if machineType < 0.}

        discardError := SysEnvirons(1, theWorld);

        { Even if we got an error from SysEnvirons, the SysEnvirons glue
          has set up machineType.}

        IF theWorld.machineType < 0 THEN
            WNEIsImplemented := FALSE
            {this ROM doesn't have separate trap tables or WaitNextEvent}
        ELSE
            WNEIsImplemented := TrapAvailable(WNETrapNumber, ToolTrap);
            {check for WaitNextEvent}
        END; {WNEIsImplemented}

        {Note that we call SystemTask if WaitNextEvent isn't available.}

        ...
        hasWNE := WNEIsImplemented;
        ...
        IF hasWNE THEN BEGIN
            {call WaitNextEvent}
            ...
        END ELSE BEGIN
            {call SystemTask and GetNextEvent}
            ...
        END;

```

Here's a C fragment:

```

Boolean TrapAvailable(tNumber, tType)
short    tNumber
TrapType tType
{
    /* define trap number for old MPW or non-MPW C */
    #ifndef _Unimplemented
    #define _Unimplemented 0xA89F
    #endif

    /* Check and see if the trap exists. */
    /* On 64K ROM machines, tType will be ignored. */

    return( NGetTrapAddress(tNumber, tType) !=
            GetTrapAddress(_Unimplemented) );
}

Boolean WNEIsImplemented()
{
    /* define trap number for old MPW or non-MPW C */
    #ifndef _WaitNextEvent
    #define _WaitNextEvent 0xA860
    #endif

    SysEnvRec theWorld; /* used to check if machine has new traps */

    /* Since WaitNextEvent and HFSDispatch both have the same trap
       number ($60), we can only call TrapAvailable for WaitNextEvent
       if we are on a machine that supports separate OS and Toolbox
       trap tables. We call SysEnvirons and check if machineType < 0. */

    SysEnvirons(1, &theWorld);

    /* Even if we got an error from SysEnvirons, the SysEnvirons glue
       has set up machineType. */

    if (theWorld.machineType < 0) {
        return(false)
        /* this ROM doesn't have separate trap tables or WaitNextEvent */
    } else {
        return(TrapAvailable(_WaitNextEvent, ToolTrap));
        /* check for WaitNextEvent */
    }
}

/* Note that we call SystemTask if WaitNextEvent isn't available. */

...
hasWNE = WNEIsImplemented();
...
if (hasWNE) {
    /* call WaitNextEvent */
    ...
} else {
    /* call SystemTask and GetNextEvent */
    ...
}

```

**Note:**

Testing to see if `WaitNextEvent` is implemented is **not** the same as testing to see whether MultiFinder is running. Systems 6.0 and newer include `WaitNextEvent` whether or not MultiFinder is running.

[Back to top](#)

## How can I tell if the MultiFinder Temporary Memory Allocation calls are implemented?

Date Written: 9/87

Last reviewed: 3/88

The technique that's used to determine this is similar to the above technique. The `TrapAvailable` routine above is reused. In Pascal:

```

FUNCTION TempMemCallsAvailable: BOOLEAN;

    CONST
        OSDDispatchTrapNumber = $A88F; {number for temporary memory calls}

    BEGIN {TempMemCallsAvailable}

    { Since OSDDispatch has a trap number that was always defined
      to be a toolbox trap ($8F), we can always call TrapAvailable.
      If we are on a machine that does not have separate OS and
      Toolbox trap tables, we'll still get the right trap address.}

    TempMemCallsAvailable := TrapAvailable(OSDispatchTrapNumber, ToolTrap);
    {check for OSDDispatch}

```

In C:

```

Boolean
TempMemCallsAvailable()
{

    /* define trap number for old MPW or non-MPW C */
    #ifndef _OSDispatch
    #define _OSDispatch 0xA88F
    #endif

    /* Since OSDDispatch has a trap number that was always defined to
      be a toolbox trap ($8F), we can always call TrapAvailable.
      If we are on a machine that does not have separate OS and
      Toolbox trap tables, we'll still get the right trap address. */

    return(TrapAvailable(_OSDispatch, ToolTrap));
    /* check for OSDDispatch */
}

```

[Back to top](#)

## How can I tell if my application is running in the background?

Date Written: 9/87

Last reviewed: 3/88

To run in the background under MultiFinder, an application must have set the `canBackground` bit (bit 12 of the first word) in the `SIZE` resource. In addition, the `acceptSuspendResumeEvents` bit (bit 14) should be set. An application can tell it is running in the background if it has received a suspend event but not a resume event.

[Back to top](#)

## When exactly does juggling take place?

Date Written: 9/87

Last reviewed: 3/88

Juggling takes place at `WaitNextEvent/GetNextEvent/EventAvail` time. If you have the `acceptSuspendResumeEvents` bit set in the `SIZE` resource, you will receive suspend/resume events. When you get a suspend event (or, when you call `EventAvail` and a suspend event has been posted), you will be juggled out the next time that you call `WNE/GNE/EventAvail`. When you receive a suspend event, you are going to be juggled, so don't do anything to try to retain control (such as calling `ModalDialog`).

Speaking of `ModalDialog`, MultiFinder will **not** suspend your application when the frontmost window is a modal dialog, though background tasks will continue to get time.

[Back to top](#)

## Can I disable suspend/resume events by passing the appropriate event mask to WNE/GNE/EventAvail?

Date Written: 9/87

Last reviewed: 3/88

suspend/resume events are not queued, so be careful when masking out `app4Evts`. You will still get the event, all that will happen if you mask out `app4Evts` is that your application won't know when it is going to be juggled out (your

application will still be juggled out when you call `WNE/GNE/EventAvail`). If your application sets a boolean to tell whether or not it's in the foreground or the background, you definitely don't want to mask out `app4Evts`.

[Back to top](#)

## Should my application use `WaitNextEvent`?

Date Written: 9/87

Last reviewed: 3/88

Yes, this will enable background tasks to get as much time as possible. All user events that your program needs to handle will be passed to your application as quickly as possible. Applications that run in the background should try to be as friendly as possible. It's best to do things a small chunk at a time so as to give maximum time to the foreground application. "Cooperative multi-tasking" requires cooperation!

If your application calls `WaitNextEvent`, it shouldn't call `SystemTask`.

[Back to top](#)

## Is there anything else that I can do to be MultiFinder friendly?

Date Written: 9/87

Last reviewed: 11/16/93

It is very important that you save the positions of windows that you open, so that the next time the user launches your application, the windows will go where they had them last. This greatly enhances the usability of MultiFinder. See Technote TB575 - [Window Manager Q&As: "How to save and restore window positions."](#)

[Back to top](#)

## Can I use a debugger with MultiFinder?

Date Written: 9/87

Last reviewed: 3/88

Yes, MacsBug will load normally, since it is loaded well before MultiFinder. Since TMON is currently installed as a startup application, you should Set Startup to it, then launch MultiFinder manually (by holding down Option-Command while double-clicking the MultiFinder icon) or use a program that will run multiple startup applications (such as Sequencer), making sure that TMON is run before MultiFinder. If you try to run TMON after MultiFinder has been installed, a system crash will result. The latest version of TMON (2.8) has an INIT that loads it before MultiFinder is present.

It is necessary to check `CurApName` (\$910) when you first enter a debugger (TMON users can anchor a window to \$910) to see which layer (whose code, which low-memory globals and so on) is currently executing, especially if you entered the debugger by pressing the interrupt button.

[Back to top](#)

## What happened to animated icons under MultiFinder?

Date Written: 9/87

Last reviewed: 3/88

Finders 6.0 and newer no longer use the mask that you supply in an ICN# to "punch a hole" in the desktop. Instead, the Finder uses a default mask that consists of a solid black copy of the icon with no hole.

[Back to top](#)

## How can I ensure maximal compatibility with MultiFinder?

Date Written: 9/87

Last reviewed: 3/88

If you follow the guidelines presented in the MultiFinder Developer's Package you will stand a very good chance of being fully compatible with MultiFinder.

[Back to top](#)

## References

[M.OV.GestaltSysenvirons](#)

[M.OV.ChkForFunction](#)

MultiFinder Developer's Package

[Back to top](#)

## Downloadables



Acrobat version of this Note (K).

[Download](#)

---

Technical Notes by [Date](#) | [Number](#) | [Technology](#) | [Title](#)  
[Developer Documentation](#) | [Technical Q&As](#) | [Development Kits](#) | [Sample Code](#)