Technotes

Download Acrobat file (K)	Download AppleWorks version

Background-Only Applications

Technote 1070 **October 1996**

Background-only applications (BOAs) are, as the name implies, Macintosh applications that run only in the background. BOAs are the preferred alternative to INITs, drivers, and stand-alone code for most startup-time and background "daemon" services. This Note discusses various issues that affect development of BOAs, as well as useful implementation strategies for BOAs.

This Technote, originally PS 02, has been revised to include information about the appe code resource used in controlling application extensions, details about SetApplLimit, and a bug workaround for application extensions with INITs under system software prior to System 7.5.5. The Note supersedes the information in PS 02.

CONTENTS

Overview

Background-only applications, also called faceless background applications (FBAs) or application extensions, are the easiest way to provide either startup-time or continuous background services. Although BOAs work under System 6 and MultiFinder, the process management and interprocess communication facilities of System 7 allow BOAs to provide far more sophisticated services.

This Note covers some of the uses and pitfalls of developing background-only applications. For an introduction to writing BOAs, see "Be Our Guest" in issue 9 of *develop*. Sample Pascal and C code for background-only applications is available in the Snippets collection on the *Developer Series CD* as > SmallDaemon in both C and Pascal.

How To Be a BOA

Background-only applications tend to look like reduced Macintosh applications. A BOA should call InitGraf to initialize QuickDraw globals (which is necessary to use the Apple Event Manager), but a BOA must not call `InitWindows`, `InitMenus`, or any toolbox or operating system routine that might draw on the screen. This rules out making any call from a BOA that might raise a dialog either intentionally (like `PPCBrowser`) or unintentionally (like `ResolveAlias`). There are typically alternative routines that avoid the need for user interaction (such as `IPCListPorts` and `MatchAlias`) that can be used from within a BOA. One-way communication from a BOA to the user can be done via the Notification Manager (see "Posting Notifications" later in this Note).

Like all other Macintosh applications, a BOA should handle the four required Apple events. It is particularly important that the QuitApplication Apple event be supported, as System 7 will refuse to shut down if a BOA does not quit as requested.

Be sure to set the canBackground and backgroundOnly bits in the application's SIZE -1 resource to indicate to the system that the application is a background-only process.

Stack Issues

The stack space allocated to BOAs by default is only 2K. This is much smaller than the usual default stack size of 8K (on Macintosh computers without Color QuickDraw) or 24K (on Macintosh models with Color QuickDraw). Realistically, a 2K stack is too small on PowerPC machines. Any recursive or stack-intensive operation can easily cause the BOA's stack to crash into its heap. This can typically be avoided by using standard looping constructs rather than recursion and by using heap allocations (`NewHandle`, `NewPointer`, or for transitory needs, `TempNewHandle`) instead of large parameters or local variables. Be careful to avoid declaring local variables of type `Str255`, as those will quickly use up stack space.

If it is necessary to increase a BOA's stack, it can be done by calling `GetApplLimit` and `SetApplLimit` to reduce the BOA's heap space. Be sure to increase the stack size before calling `MaxApplZone`() or the expansion will fail:

```
// Increase the space allocated for the background only application stack.
//
// Warning: SetApplLimit always sets the stack to at least as large as the
//    default stack for the machine (8K on machines with original QuickDraw,
//    24K on machines with Color QuickDraw), so the application partition
//    must be large enough to accommodate an appropriate stack and heap.
//    Call this only once, at the beginning of the BOA.
//
// Another warning:
//    Don't bother trying to set the stack size to something lower than 24K.
//    If SetApplLimit is called to do this, it will silently lower ApplLimit
//    to a 24K stack. (The limit is 8K on machines without Color QuickDraw.
//    In this sample, we don't allow an increment less than 24K.)

OSErr IncreaseBOAStack(Size additionalStackSize)
{
  OSErr retCode;

  // Check that we aren't running with a corrupt heap. If we are,
  // fix the problem.  This was a bug with FBA's before System 7.5.5.
  // With System Software later than 7.5.5, this "fix" is harmless.
  myZone = GetZone();
  if (myZone->bkLim != LMGetHeapEnd())
      LMSetHeapEnd(myZone->bkLim);

  // Increase the stack size by lowering the heap limit.
  SetApplLimit((Ptr) ((unsigned long) GetApplLimit() - additionalStackSize));
  retCode = MemError();
  if (retCode == noErr) MaxApplZone();

  return retCode;
}
```

Why Do It Before `MaxApplZone`?

`MaxApplZone` works by expanding the application heap up to the address contained in the ApplLimit low-memory global, and SetApplLimit works by setting the ApplLimit global (for our purposes, lower). If `MaxApplZone` is called before setting the stack size, the heap has already been expanded as far as it can go, so ApplLimit cannot be set any lower. Unfortunately, SetApplLimit doesn't return an error (from the routine or via MemError), so there is no way to know it failed.

Posting Notifications

When an exceptional condition needs to be made known to the user, a BOA can use the Notification

Manager to display a string or play a sound. Since a BOA is not listed in the application menu and cannot be brought to the front like other applications, the user may not even know that it is running, so notifications can be startling. Use the Notification Manager only to report a serious problem or to request that the user take some needed action.

Some developers have insisted on trying to achieve user interaction from a BOA. The most compatible solution we have devised is to have the BOA launch a regular application, have the application do the user interaction, gather any user input, and send the user input to the BOA via AppleEvents. A BOA can not draw to the screen, nor make toolbox calls which draw to the screen.

The Finder and BOA Launching

There are numerous ways to launch a BOA. If a BOA has the file type `APPL`, it can be opened by the user in the Finder just like other applications. It can also be placed in the Startup Items folder under System 7 for launching when the system starts up.

If a BOA has the file type `appe`, it is an *application extension*. If the user drags an application extension file to the System folder, the extension will be routed to the Extensions folder and then launched when the system starts up.

If an INIT resource is in an application extension file and the file is in the Extensions folder, the system will execute the INIT code along with the INITs of other extensions. This can be useful for displaying an icon during booting. (Displaying an icon from an INIT at startup is typically done by using the > ShowInit code, available on the *Developer Series CD.*) Since the launch of the BOA will not actually occur until after all INIT resources have been run, a BOA's startup icon cannot be informative. For example, the BOA can't display an X over its icon to indicate that loading has failed, because the BOA has not yet been launched at INIT time. However, showing an icon during booting is still useful to remind users that the application extension may be installed and eating up memory.

A BOA can be launched with the Process Manager routine `LaunchApplication`. `LaunchApplication` ignores the type of files it is asked to launch, so the BOA can be launched with `LaunchApplication` whether its file type is APPL, appe, or something else.

As of System 7.1.1, a BOA that is launched as an application extension (file type `appe`) from the Extensions folder at startup can include a standalone code resource of type appe to control whether or not the application extension gets launched. The appe code resource just contains a Pascal function of no parameters that returns a Boolean; if the code resource returns false, the application extension will not be launched.

For example, here is a routine to let an application extension be launched only if Color QuickDraw was available:

```
pascal Boolean ShouldThisBeLaunched(void)
{
   OSErr err;
   long response;

   err= Gestalt(gestaltQuickdrawVersion, &response);
   if (err == noErr && response >= gestalt8BitQD)
      return true;

   else
      return false;

}
```

The System 7.x Finder is unable to determine when a BOA stops running. As a result, a BOA icon that was dimmed when the user launched the application by opening it in the Finder will not be undimmed if the BOA quits. This is just a cosmetic problem, as the user can relaunch the BOA by double-clicking the still-dimmed icon.

BOA Tasks

Faceless background applications can do most of the jobs formerly handled by stand-alone code and by periodically executed drivers. The primary reason to use an INIT instead of a BOA is to make global trap patches. Because a BOA is an application, a trap address set from within the BOA will apply only to the BOA, not globally.

A BOA can not ever explicitly draw on the screen. While safely creating and initializing an appropriate world for drawing from stand-alone code (typically from an INIT) is quite difficult, it can be done. If any drawing will be necessary, do not use a BOA.

It is safe and appropriate to store interrupt routines within the heap of a BOA. For example, a Time Manager task's code or a completion routine may actually be a procedure of a BOA. Since all of its heap space is released once a BOA quits, a BOA must not quit while any interrupt or completion routines are pending. A BOA may be put to sleep (by passing a large sleep parameter to WaitNextEvent) and, under System 7, later woken up at interrupt time with a call to WakeUpProcess.

Any application that is not performing a periodic task should use a large sleep value in its calls to `WaitNextEvent`. Since an application is woken up whenever any events are pending for it, using large sleep values will help avoid slowing down the execution of other processes without hurting the sleeping process's responsiveness. A background-only application can run periodically (as a daemon) by passing an appropriately small sleep value to WaitNextEvent.

Background Apple Events

Although Apple events are an excellent medium for communication between a BOA and other processes, be careful to avoid situations that may require user interaction. For example, attempting to target a process on another machine for which there is no current session would raise the program linking dialog. A BOA can usually avoid those situations by handling Apple events and responding with the reply event, or by saving the address of an event sender (by calling AEGetAttributePtr to extract the keyAddressAttr attribute of an Apple event sent to the BOA) and using it later to target a response.

BOA's Before System 7.5.5

Prior to System 7.5.5, there was a bug in the Process Manager which affected BOAs. If you had two or more BOAs installed, the second BOA would have a corrupted heap. (In a previous version of this Technote, this bug was incorrectly documented with the sentence, "Currently there is a bug with 'appe's that have an 'INIT' under System 7.5 with PowerTalk installed." The bug only showed up in this configuration because PowerTalk had a BOA which it installed.) The BOA's heap can be corrupted if `MaxApplZone` is called.

If you are not running System 7.5.5 or later, the workaround is to compare the low-memory global HeapEnd to the BOA's bkLim field of its zone (obtained by calling `GetZone`). If the two values aren't the same, set HeapEnd to the value contained in `theZone->bkLim` (use the low memory accessors `LMGetHeapEnd` and `LMSetHeapEnd` to access/modify the low memory global.) This should be the first thing done in your BOA code, before setting the stack size or calling `MaxApplZone`.

Further Reference:

- *Inside Macintosh:Macintosh Toolbox Essentials* , Event Manager, Finder Interface
- *Inside Macintosh: Interapplication Communication* , Apple Events Manager
- *Inside Macintosh: Processes* , Process Management
- "Be Our Guest: Background-Only Applications in System 7" in issue 9 of *develop* introduces BOAs.
- Technical Note PT 35 - Stand-alone code, *ad nauseam* (discusses requirements for INITs)
- Technote 1069 - System 7.5.5
