Launching Other Applications

The information in this section about launching applications replaces the corresponding discussion in the *Programmer's Guide to MultiFinder*.

You can launch other applications using the high-level **LaunchApplication** function. The **LaunchApplication** function lets your application control various options associated with launching an application. For example, you can

- allow the application to be launched in a partition smaller than the preferred size but greater than the minimum size, or allow it to be launched only in a partition of the preferred size
- launch an application without terminating your own application, bring the launched application to the front, and get information about the launched application
- request that your application be notified if any application that it has launched terminates

Earlier versions of system software used a shorter parameter block as a parameter to the _Launch trap macro. The _Launch trap macro still supports the use of this parameter block. Applications using the **LaunchApplication** function should use the new **LaunchParamBlockRec** . Use the **Gestalt** function and specify the selector gestaltOSAttr to determine which launch features are available. See the **Compatibility Guidelines** for information on how to use the **Gestalt** function.

Most applications do not need to launch other applications. However, if your application includes a desk accessory or another application, you might use either the high-level **LaunchApplication** function to launch an application or the **LaunchDeskAccessory** function to launch a desk accessory. For example, if you have implemented a spelling checker as a separate application, you might use the **LaunchApplication** function to open the spelling checker when the user chooses Check Spelling from one of your application's menus.

You specify a **LaunchParamBlockRec** as a parameter to the **LaunchApplication** function. You use the **LaunchParamBlockRec** to specify the filename of the application to launch, to specify whether to allocate the preferred size for the application's heap or to allow a partition size less than the preferred size, and to set various other options-for example, whether your application should continue or terminate after it launches the specified application.

The **LaunchApplication** function launches the application from the specified file and returns the process serial number, preferred partition size, and minimum partition size if the application is successfully launched.

Note that if you launch another application without terminating your application, the launched application does not actually begin executing until you make a subsequent call to **WaitNextEvent** or **EventAvail**.

The launch parameter block is defined by the **LaunchParamBlockRec**. Specify the constant extendedBlock in the launchBlockID field to identify the parameter block and to indicate that you are using the fields following it in the **LaunchParamBlockRec**.

The code example shows the use of the **LaunchApplication** function

```
// Launching an application

// Assuming inclusion of <MacHeaders>

#include <Processes.h>

void LaunchAnApplication (StandardFileReply *mySFReply);
void DoError (OSErr myErr);

void LaunchAnApplication (StandardFileReply *mySFReply)
{
    LaunchParamBlockRec         myLaunchParams;
    ProcessSerialNumber         launchedProcessSN;
    OSErr           launchErr;
    long            prefSize;
    long            minSize;
    long            availSize;


    myLaunchParams.launchBlockID = extendedBlock;
    myLaunchParams.launchEPBLength = extendedBlockLen;
    myLaunchParams.launchFileFlags = 0;
    myLaunchParams.launchControlFlags = launchContinue + launchNoFileFlags;
    myLaunchParams.launchAppSpec = &(mySFReply->sfFile);
    myLaunchParams.launchAppParameters = nil;

    launchErr = LaunchApplication(&myLaunchParams);

    prefSize = myLaunchParams.launchPreferredSize;
    minSize = myLaunchParams.launchMinimumSize;
    if (!launchErr)
        launchedProcessSN = myLaunchParams.launchProcessSN;
    else  if (launchErr == memFullErr)
        availSize = myLaunchParams.launchAvailableSize;
    else
        DoError(launchErr);
}
```

In the code example above, the application file to launch is specified by using a file system specification record returned by the **StandardGetFile** routine and specifying a pointer to this record in the launchAppSpec field. The launchControlFlags field indicates that **LaunchApplication** should extract the Finder flags from the application file, launch the application in the preferred size, bring the launched application to the front, and that **LaunchApplication** should not terminate the current process.

Specifying Launch Options

When you use the **LaunchApplication** function, you specify the launch options in the launchControlFlags field of the **LaunchParamBlockRec**. These are the constants you can specify in the launchControlFlags field.

```
launchContinue      = 0x04000,
launchNoFileFlags   = 0x00800,
launchUseMinimum    = 0x00400,
launchDontSwitch    = 0x00200,
launchInhibitDaemon = 0x00080
```

Set the launchContinue flag if you want your application to continue after the specified application is launched. If you do not set this flag, **LaunchApplication** terminates your application after launching the specified application, even if the launch fails.

Set the launchNoFileFlags flag if you want the **LaunchApplication** function to ignore any value specified in the launchFileFlags field. If you set the launchNoFileFlags flag, the **LaunchApplication** function extracts the Finder flags from the application file for you. If you want to extract the file flags, clear the launchNoFileFlags flag and specify the Finder flags in the launchFileFlags field of the **LaunchParamBlockRec**.

Clear the launchUseMinimum flag if you want the **LaunchApplication** function to attempt to launch the application in the preferred size (as specified in the application's 'SIZE' resource). If you set the launchUseMinimum flag, the **LaunchApplication** function attempts to launch the application using the largest available size greater than or equal to the minimum size but less than the preferred size. If the **LaunchApplication** function returns the result code memFullErr or memFragErr, the application cannot be launched given the current memory conditions.

Set the launchDontSwitch flag if you do not want the launched application brought to the front. If you set this flag, the launched application runs in the background until the user brings the application to the front-for example, by clicking in one of the application's windows. Note that most applications expect to be launched in the foreground. If you clear the launchDontSwitch flag, the launched application is brought to the front, and your application is sent to the background.

Set the launchInhibitDaemon flag if you do not want **LaunchApplication** to launch a background-only application. (A background-only application has the onlyBackground flag set in its 'SIZE' resource.)

Controlling Launched Applications

When your application launches another application using **LaunchApplication** , the launched application is automatically brought to the front, sending the foreground application to the background. If you do not want to bring the application to the front when it is first launched, set the launchDontSwitch flag in the launchControlFlags field of the **LaunchParamBlockRec**.

In addition, if you want your application to continue to run after it launches another application, you must set the launchContinue flag in the launchControlFlags field of the **LaunchParamBlockRec**.

You can control the scheduling of the launched application in a limited way by using the **Process Manager** routines **SetFrontProcess** and **WakeUpProcess** .

If you want your application to be notified when an application it has launched terminates, set the acceptAppDiedEvents flag in your 'SIZE' resource. If you set this flag and an application launched by your application terminates, your application receives an Application-Died event ('aevt' 'obit').

See the **Apple Event Manager** for information on receiving the Application-Died event and other Apple events.

Launching Desk Accessories

In version 7.0, when a desk accessory is opened, the **Process Manager** launches the desk accessory in its own partition. The **Process Manager** gives it a process serial number and an entry in the process list. The **Process Manager** puts the name of the desk accessory in the list of open applications in the Application menu and also gives the active desk accessory its own About menu item in the Apple menu that contains the name of the desk accessory. This makes desk accessories more consistent with the user interface of small applications.

Your application can launch desk accessories using the **LaunchDeskAccessory** function. However, you should use this function only when your application needs to launch a desk accessory for some reason other than in response to the user choosing a desk accessory from the Apple menu. In version 7.0, the Apple menu can contain any Finder object that the user chooses to add to the menu. When the user chooses any item from the Apple menu that you did not add specifically for your application, use the **OpenDeskAcc** function.