New Technical Notes

Macintosh

Developer Support

Sub(Launching) From a High-Level Language

Processes

Revised by: C. K. Haun <TR> May 1993
Revised by: Rich Collyer and Mark Johnson April 1989
Written by: Rick Blair and Jim Friedlander May 1987

Note: Developer Technical Support takes the view that launching and sublaunching in systems earlier than 7.0 are features that are best **avoided** for compatibility (and other) reasons, but we want to make sure that when it is absolutely necessary to implement launching and sublaunching, they are done in the safest possible way.

This Technical Note discusses the "safest" method of calling `_Launch` from a high-level language that supports inline assembly language with the option of launching or sublaunching another application.

Changes since April 1989: Added note about LaunchApplication in System 7 and later.
Changes since August 1988: Incorporated Technical Note #52 on calling `_Launch` from a high-level language, changed the example to offer a choice between launching or sublaunching, added a discussion of the `_Launch` trap under MultiFinder, and updated the MPW C code to include inline assembly language.

Note: The information about launching in this note is primarily geared to applications that are written for Macintosh computers running System 6, or earlier versions of the Macintosh Operating System.

If you are writing an application for System 7 and later systems, you will find the `LaunchApplication` routine much easier to use and more appropriate. LaunchApplication is documented in the Process Manager chapter of *Inside Macintosh* Volume VI, and in the Processes volume of the *New Inside Macintosh* series.

Launching and sublaunching from an application are fully supported under System 7, and the previous warnings concerning launching do not apply to System 7 or later systems.

The Segment Loader chapter of *Inside Macintosh* Volume II, page 53, states the following about the `_Launch` trap:

> “The routines below are provided for advanced programmers; they can be called only from assembly language.”

While this statement is technically true, it is easy to call _Launch from any high-level language that supports inline assembly code, and this Note provides examples of calling _Launch in MPW Pascal and C.

Before calling _Launch, you need to declare the inline procedure, which takes a variable of type pLaunchStruct as a parameter. Since the compiler pushes a pointer to this parameter on the stack, you need to include code to put this pointer into A0. The way to do this is with a MOVE.L (SP)+,A0 instruction, which is $205F in hexadecimal, so the first word after INLINE is $205F. This instruction sets up A0 to contain a pointer to the filename and 4(A0) to contain the configuration parameter, so the last part of the inline is the _Launch trap itself, which is $A9F2 in hexadecimal. The configuration parameter, which is normally 0, determines whether the application uses alternate screen and sound buffers. Since not all Macintosh models support these alternate buffers, you should avoid using them unless you have a specific circumstance which requires them.

The Finder does a lot of hidden cleanup and other tasks without user knowledge; therefore, it is best if you do not try to replace the Finder with a "mini" or try to launch other programs and have them return to your application. In the future, the Finder may provide better integration for applications, and you will circumvent this if you try to act in its place by sublaunching other programs.

If you have a situation where your application **must** launch another and have it return, and where you are not worried about incompatibility with future System Software versions, there is a "preferred" way of doing this that fits into the current system well. System file version 4.1 (or later) includes a mechanism for allowing a call to another application; we term this call a *sublaunch.* You can perform a sublaunch by adding a set of simple extensions to the parameter block you pass to the _Launch trap.

_Launch and MultiFinder

Under MultiFinder, a sublaunch behaves differently than under the Finder. The application you sublaunch becomes the foreground application, and when the user quits that application, the system returns control to the next frontmost layer, which will not necessarily be your application.

If you set both high bits of LaunchFlags, which requests a sublaunch, your application will continue to execute after the call to _Launch. Under MultiFinder, the actual launch (and suspend of your application) will not happen in the _Launch trap, but rather after a call or more to _WaitNextEvent.

Under MultiFinder, _Launch currently returns an error if there is not enough memory to launch the desired application, if it cannot locate the desired application, or if the desired application is already open. In the latter case, that application will **not** be made active. If you

attempted to launch, MultiFinder will call `_SysBeep`, your application will terminate, and control will given to the next frontmost layer. If you attempted to sublaunch, control will return to your application, and it is up to you to report the error to the user.

Currently, `_Launch` returns an error in register `D0` for a sublaunch, and you should check it for errors (`D0<0`) after any attempts at sublaunching. If `D0>=0` then your sublaunch was successful.

You should refer to the *Programmer's Guide to MultiFinder* (APDA) and Macintosh Technical Notes #180, MultiFinder Miscellanea, and #205, MultiFinder Revisited: The 6.0 System Release, for further discussion of the `_Launch` trap under MultiFinder.

Working Directories and Sublaunching With the Finder

Putting aside the compatibility issue for the moment, the only problem sublaunching creates under the **current** system is one of Working Directory Control Blocks (WDCBs). Unless the application you are launching is at the root directory or on an MFS volume, you must create a new WDCB and set it as the current directory when you launch the application.

In the example that follows, the new working directory is opened (allocated) by Standard File and its `WDRefNum` is returned in `reply.vRefNum`. If you do not use Standard File and cannot assume, for instance, that the application was in the blessed folder or root directory, then you must open a new working directory explicitly via a call to `_OpenWD`. You should give the new WDCB a `WDProcID` of `'ERIK'`, so the Finder (or another shell) would know to deallocate when it saw it was allocated by a "sublaunchee."

Although the sublaunching process is recursive (that is, programs that are sublaunched may, in turn, sublaunch other programs), there is a limit of 40 on the number of WDCBs that can be created. With this limit, you could run out of available WDCBs very quickly if many programs were playing the shell game or neglecting to deallocate the WDCBs they had created. Make sure you check for **all** errors after calling `_PBOpenWD`. A `tMWDOErr` (–121) means that all available WDCBs have been allocated, and if you receive this error, you should alert the user that the sublaunch failed and continue as appropriate.

Warning:	Although the example included in this Note covers sublaunching, Developer Technical Support strongly recommends that developers **not** use this feature of the `_Launch` trap. This trap will change in the not-too-distant future, and when it does change, applications that perform sublaunching **will** break. The only circumstance in which you could consider sublaunching is if you are implementing an integrated development system and are prepared to deal with the possibility of revising it every time Apple releases a new version of the system software.

MPW Pascal

```
{It is assumed that the Signals are caught elsewhere; see Technical
Note #88 for more information on the Signal mechanism}

{the extended parameter block to _Launch}
TYPE
```

```
pLaunchStruct = ^LaunchStruct;
LaunchStruct = RECORD
   pfName      : StringPtr;
   param       : INTEGER;
   LC          : PACKED ARRAY[0..1] OF CHAR; {extended parameters:}
   extBlockLen : LONGINT; {number of bytes in extension = 6}
```

```
     fFlags      : INTEGER; {Finder file info flags (see below)}
     launchFlags : LONGINT; {bit 31,30=1 for sublaunch, others reserved}
  END; {LaunchStruct}

FUNCTION LaunchIt(pLaunch: pLaunchStruct): OSErr; {< 0 means error}
  INLINE $205F, $A9F2, $3E80;
{ pops pointer into A0, calls Launch, pops D0 error code into result:
 MOVE.L  (A7)+,A0
 _Launch
 MOVE.W  D0,(A7)  ; since it MAY return }

PROCEDURE DoLaunch(subLaunch: BOOLEAN);  {Sublaunch if true and launch if false}

      VAR
         myLaunch     : LaunchStruct;   {launch structure}
         where        : Point;          {where to display dialog}
         reply        : SFReply;        {reply record}
         myFileTypes  : SFTypeList;     {we only want APPLs}
         numFileTypes : INTEGER;
         myPB         : CInfoPBRec;
         dirNameStr   : str255;

      BEGIN
         where.h := 20;
         where.v := 20;
         numFileTypes:= 1;
         myFileTypes[0]:= 'APPL';       {applications only!}
      {Let the user choose the file to Launch}
         SFGetFile(where, '', NIL, numFileTypes, myFileTypes, NIL, reply);

         IF reply.good THEN BEGIN
            dirNameStr:= reply.fName;   {initialize to file selected}

      {Get the Finder flags}
            WITH myPB DO BEGIN
                ioNamePtr:= @dirNameStr;
                ioVRefNum:= reply.vRefNum;
                ioFDirIndex:= 0;
                ioDirID:= 0;
            END; {WITH}
            Signal(PBGetCatInfo(@MyPB,FALSE));
      {Set the current volume to where the target application is}
            Signal(SetVol(NIL, reply.vRefNum));

      {Set up the launch parameters}
            WITH myLaunch DO BEGIN
               pfName := @reply.fName;   {pointer to our fileName}
               param := 0;               {we don't want alternate screen or sound
                                 buffers}
               LC := 'LC';               {here to tell Launch that there is nonjunk
                                 next}
               extBlockLen := 6;         {length of param. block past this long
                                 word}
               {copy flags; set bit 6 of low byte to 1 for RO access:}
               fFlags := myPB.ioFlFndrInfo.fdFlags;  {from GetCatInfo}

      {Test subLaunch and set LaunchFlags accordingly}
            IF subLaunch THEN
               LaunchFlags := $C0000000             {set BOTH high bits for a
                                    sublaunch}
            ELSE
               LaunchFlags := $00000000;            {Just launch then quit}
            END; {WITH}
```

```
        {launch; you might want to put up a dialog box that explains that
         the selected application couldn't be launched for some reason.}
             Signal(LaunchIt(@myLaunch));
          END; {IF reply.good}

       END; {DoLaunch}
```

MPW C

```
typedef struct LaunchStruct {
       char          *pfName;            /* pointer to the name of launchee */
       short int     param;
       char          LC[2];              /*extended parameters:*/
       long int      extBlockLen;        /*number of bytes in extension == 6*/
       short int     fFlags;             /*Finder file info flags (see below)*/
       long int      launchFlags;        /*bit 31,30==1 for sublaunch, others
                              reserved*/
} *pLaunchStruct;

pascal OSErr LaunchIt( pLaunchStruct pLnch) /* < 0 means error */
       = {0x205F, 0xA9F2, 0x3E80};

/* pops pointer into A0, calls Launch, pops D0 error code into result:
         MOVE.L  (A7)+,A0
         _Launch
         MOVE.W  D0,(A7)  ; since it MAY return */

OSErr DoLaunch(subLaunch)
       Boolean             subLaunch;    /* Sublaunch if true and launch if false
                                    */
{  /* DoLaunch */
       struct LaunchStruct myLaunch;
       Point               where;        /*where to display dialog*/
       SFReply             reply;        /*reply record*/
       SFTypeList          myFileTypes;  /* we only want APPLs */
       short int           numFileTypes=1;
       HFileInfo           myPB;
       char                *dirNameStr;
       OSErr               err;

       where.h = 80;
       where.v = 90;
       myFileTypes[0] = 'APPL';          /* we only want APPLs */
       /*Let the user choose the file to Launch*/
       SFGetFile(where, "", nil, numFileTypes, myFileTypes, nil, &reply);

       if (reply.good)
       {
              dirNameStr = &reply.fName; /*initialize to file selected*/

       /*Get the Finder flags*/
              myPB.ioNamePtr= dirNameStr;
              myPB.ioVRefNum= reply.vRefNum;
              myPB.ioFDirIndex= 0;
              myPB.ioDirID = 0;
              err = PBGetCatInfo((CInfoPBPtr) &myPB,false);
              if (err != noErr)
                     return err;
```

```
	/*Set the current volume to where the target application is*/
		err = SetVol(nil, reply.vRefNum);
		if (err != noErr)
			return err;

	/*Set up the launch parameters*/
		myLaunch.pfName = &reply.fName;   /*pointer to our fileName*/
		myLaunch.param = 0;               /*we don't want alternate screen
		                                    or sound buffers*/
	/*set up LC so as to tell Launch that there is non-junk next*/
		myLaunch.LC[0] = 'L'; myLaunch.LC[1] = 'C';
		myLaunch.extBlockLen = 6;         /*length of param. block past
		                                    this long word*/
	/*copy flags; set bit 6 of low byte to 1 for RO access:*/
		myLaunch.fFlags = myPB.ioFlFndrInfo.fdFlags;   /*from
		                                    _GetCatInfo*/
	/* Test subLaunch and set launchFlags accordingly    */
	if ( subLaunch )
		myLaunch.launchFlags = 0xC0000000;      /*set BOTH hi bits for a
		                              sublaunch  */
	else
		myLaunch.launchFlags = 0x00000000;      /* Just launch then quit         */

		err = LaunchIt(&myLaunch);        /* call _Launch                  */
		if (err < 0)
		{
		/* the launch failed, so put up an alert to inform the user */
			LaunchFailed();
			return err;
		}
		else
			return noErr;
	} /*if reply.good*/
} /*DoLaunch*/
```

Further Reference:

- *Inside Macintosh*, Volume I, page 12; Volume II, page 53; and Volume IV, page 83, The Segment Loader
- *Programmer's Guide to MultiFinder* (APDA)
- *Inside Macintosh*, Volume VI, Process Manager
- *New Inside Macintosh:Processes* Process Manager
- Technical Note M.OV.GestaltSysenvirons —
  Gestalt and SysEnvirons - a Never Ending Story
- Technical Note M.TB.Multifinder Misc —
  MultiFinder Miscellanea
- Technical Note M.OV.Multifinder—
  The 6.0 System Release