Macintosh Programmer's Workshop
A5 Driver Runtime Library
Writing a Desk Accessory or Device Driver that Uses Global Data.

Steve Hartwell
June 12, 1986
Revised July 30, 1986

Notice:

This documents the **A5DRVRRuntime.o** library, which is *not released with MPW 1.0.* For information about writing desk accessories or drivers which don't need global data, use the **DRVRRuntime.o** library provided with the MPW 1.0 release, and see the MPW manual section on "*Writing a Desk Accessory or Other Driver Resource*".

The A5DRVRRuntime.o library is available internally to Apple engineers; contact the Development Systems Group for more information.

Overview.

Desk accessories and other drivers have traditionally been written in assembly language, partly because of the peculiar format of the 'DRVR' resource needed for drivers. Setting up the DRVR layout header, passing register-based procedure parameters, and coping with the nonstandard exit conventions of the driver routines has made it fairly difficult to implement drivers in higher-level languages like Pascal or C.

The library **A5DRVRRuntime.o** and the resource type 'DRVW' declared in **MPWTypes.r** simplifies the task of writing a desk accessory or device driver in Pascal or C. Together they compose the driver layout header and the five entry points which set up the open, prime, status, control, and close functions of a driver.

The advantages of using A5DRVRRuntime.o are:

- No assembler source is required.
- The resource compiler is an integral step in the build process, permitting the easy addition of a desk accessory menu or other owned resources.
- The programmer's interface to the open, prime, status, control, and close routines use Pascal calling conventions. Since the C compiler can easily be directed to declare functions which use Pascal calling conventions, the driver header can by used by both languages without additional glue. Each function returns a result code which is passed back to the system.
- Previously, desk accessories and drivers have been required to allocate their own storage for global data. A5DRVRRuntime uses the new A5Lib routines which allow access to initialized global data as the languages and assembler support them.
- Because access to global data is available, drivers may link with the language libraries, including routines which reference global data (but see note below).

- Drivers have peculiar exit conventions, requiring immediate calls to exit via an RTS instruction, but non-immediate calls to JMP to the IODone routine. The A5DRVRRuntime glue handles the proper exit conventions.

Some programming restrictions still apply to non-application code resources such as drivers. Most notably is the restriction against the use of one variety of global data -- global variables which are initialized to procedure pointers. Unfortunately, the I/O portions of the language libraries and the Integrated Environment functions use initialized tables of procedure pointers, and therefore cannot be called from drivers (this is an *ex post facto* design flaw). The generic portions of the language libraries can be used with no difficulty.

For more information about this and other global data allocation considerations (and its impact on the Macintosh runtime environment), refer to the document *"The MPW Global Data Allocation (A5) Library"*, dated July 28, 1986, available in the Software Library.

The Structure of an MPW driver.

Putting together a desk accessory or driver requires two parts:

[1] Linked object code, put together from the A5DRVRRuntime library and your driver code. The object code is linked into a code resouce type ***DRVW***, which (roughly) stands for the "MPW DRVR" intermediate form of the **DRVR** resource.

[2] A resource compiler source file using the **DRVW** type declared in MPWTypes.r. This enables you to set the driver flags, event mask, menu ID, driver name, etc.

Building the *DRVW* code resource.

The A5DRVRRuntime library consists of a main entry point which should override or replace the usual language runtime main entry point. This main entry point contains driver "glue" which sets up the driver runtime environment for you, calls one of your driver routines, and restores the application runtime environment when your driver routine returns. Generally, the driver glue code performs the following when executing one of the driver entry points:

[1] Saves the application register A5 on the stack
[2] Sets up the driver's global data area
[3] Pushes the register parameters A0 and A1 onto the stack for your routine to use
[4] Calls your routine with Pascal calling conventions
[5] Restore the caller's global area
[6] Stores your routine's result code in register D0
[7] Returns to the caller appropriately

What your routines need to perform.

The A5DRVRRuntime library can be used for drivers written in either Pascal, C, or assembler. In Pascal, you need to declare a UNIT which declares these 5 functions in your interface: ***DRVROpen***, ***DRVRPrime***, ***DRVRStatus***, ***DRVRControl***, and ***DRVRClose***. The calling sequence for all functions is the same: the parameter *ioPB* is the pointer the driver's I/O parameter block (from register A0) and *dCtl* is the pointer to the driver's device control entry (from register A1). The function result is an integer which is the routine's result code, which is returned to the ROM in register D0.

To illustrate, the declaration for your open routine written in Pascal must be:

```
FUNCTION DRVROpen(ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
```

In C, you need to write 5 global-scope functions with the same names as above. Since the A5DRVRRuntime library uses Pascal calling conventions, you need to declare the functions with the `pascal` keyword, and match the Pascal datatypes like this:

```
pascal short
DRVROpen(ioPB, dCtl)
    CntrlParam *ioPB;
    DCtlPtr *dCtl;
```

In all, the following 5 functions must be declared:

```
FUNCTION DRVROpen    (ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
FUNCTION DRVRPrime   (ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
FUNCTION DRVRStatus  (ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
FUNCTION DRVRControl (ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
FUNCTION DRVRClose   (ioPB: IOParamBlk; dCtl: DCtlEntryPtr): INTEGER;
```

The C equivalents should be declared as in the C example illustrated above. Driver routines written in assembler should be written to expect Pascal calling conventions.

WARNING:

Your driver must **NOT** store into the Device Control Entry field *dCtlStorage*. This field was typically used in the past to store the handle to the driver's globals allocated as a record on the heap. Since A5DRVRRuntime allows access to "real" program globals, you don't need this any longer; you can just allocate UNIT globals in the VAR section of your unit (or C globals, as appropriate). The A5DRVRuntime glue uses the driver's *dCtlStorage* field to save the desk accessory's global data area pointer (register A5). Using it is not optional -- whether or not your driver actually uses globals or not, if the value of *dCtlStorage* is changed, your driver will crash. You can get at your global data pointer this way if you want to use it with any of the A5 routines; there are cases where you may wish to do this. The discussion below illustrates some of them.

The A5DRVRRuntime glue.

The following explains in detail what happens before and after your DRVR routines are called:

The Open glue:

The global data referenced by the driver will be allocated from the current heap and initialized by the A5DRVRRuntime entry point open glue before your function ***DRVROpen*** is called. The pointer to Quickdraw's globals and the rest of the Application Parameters area is copied to your driver's global area, duplicating the application environment for the driver. Just before calling your ***DRVROpen*** routine, register A5 is set up to point to the driver's global area. After your ***DRVROpen*** routine returns, the caller's global area is restored.

Subseqent calls to the driver open routine will call your ***DRVROpen*** routine again but skip over initializing the global area (which is done only once). Your ***DRVROpen*** routine will probably use some heuristic to detect a re-entrant open (such as checking `dCtlWindow` to see if

you already have a window), and only execute the relevant portions of your open code as necessary. If you want to handle multiple opens specially (multiple opens can only happen with desk accessories), you can set global variables, and arrange for your ***DRVRClose*** routine to prevent your global area from being released after the driver is closed (see below).

Note:

Because the open glue calls **A5Alloc**, which uses *NewPtr* to allocate the global area from the current heap zone, globals typically end up being allocated in the application heap. If you need your globals to be allocated on the system heap, perhaps setting the SysHeap bit in the DRVR resource will put the code in the system heap and also leave the current zone set to the system heap as the open glue executes. If this is true, *NewPtr* will allocate the globals from the system heap as desired. I haven't checked this out, so the only reliable way to do this is to have your ***DRVROpen*** routine redo some of the work of the A5DRVRRuntime open glue. This is a pain, but it's still easier than doing without A5DRVRRuntime altogether. In C, your ***DRVROpen*** routine would do this as a first-open preamble:

```
A5Dispose(dCtl->dCtlStorage);      /* Dispose of our old A5 area  */
GetZone(&curZone);                 /* Save this to restore later  */
SetZone(SysZone);                  /* Switch to the System heap   */
A5Init(mySysA5 = A5Alloc());       /* Get & Init a new A5 area    */
SetZone(curZone);                  /* Restore the current zone    */
dCtl->dCtlStorage = mySysA5;       /* Set the sysHeap global area */
(void) A5Swap(mySysA5);            /* and switch to it.           */
/* ... continue with DRVROpen code ... */
```

If it's OK to have your globals allocated on the application heap (as is the case for desk accessories), then you won't need to do any of this shuffling global areas around. Refer to *The MPW Global Data Allocation (A5) Library* document for details on the A5 routines.

The Prime, Status, and Control glue:

The A5 world is set up and restored around these calls, as with the Open glue. For desk accessories only, ***DRVRPrime*** and ***DRVRStatus*** must do nothing and return a result code of 0 (they can not be simply omitted). Device drivers may have special returnCode values; if they do, they are faithfully returned to the system in register D0. The proper exit conventions are handled (JMP to ioDone or RTS, depending on whether it is an IMMED call or not), and the caller's A5 is restored upon return.

Note:

If your ***DRVRPrime***, ***DRVRStatus***, or ***DRVRControl*** routines want to do asynchronous I/O, your completion routines can set up their global data pointer by retrieving their A5 value which is stored in the `dCtlStorage` field of the device control entry. By bracketing their code with calls to `oldA5 = A5Swap(dCtl->dCtlStorage)` on entry, and `(void) A5Swap(oldA5)` on exit, global data can be used from the body of completion routines. Refer to *The MPW Global Data Allocation (A5) Library* document for details on the A5 routines.

The Close glue:

The A5 world is set up and then the ***DRVRClose*** routine is called. If the resultCode it returns is 0, the global data area is permanently released. If the resultCode is non-zero, the global data area is preserved. A non-zero resultCode will allow the programmer to "fake" a close call, closing its windows but leaving its data around so that if it's re-opened, the previous data values are still retained. An example of this behavior is the Calculator desk accessory,

which puts away the calculator window when closed, but when it's re-opened, it still has the same number in its display that it had when it was closed. Regardless of your ***DRVRClose*** resultCode, the Close glue always returns a 0 to the system.

A Desk Accessory Example.

Appendix A is the Pascal source for an example desk accessory using UNIT globals. The source demonstrates the calling sequences of the A5DRVRRuntime routines.

The sample desk accessory provided in the document *"The MPW Global Data Allocation (A5) Library"* demonstrates how you would have to use the DRVRRuntime.o library to get the same features that the A5DRVRRuntime.o library already provides. This example is identical, except the explicit calls to the A5 routines are removed.

Warning:

Use of the A5DRVRRuntime library in desk accessories cause some applications to crash when the desk accessories are used. This is because these applications patch traps or use low-memory hooks which assume that A5 is always the application's A5. Until the technical problems with the A5 architecture have been worked out, you should not expect desk accessories which use A5DRVRRuntime to run without crashing such applications.

If you compile the source supplied in Appendix A, you can use this Link command to create the intermediate DRVW resource:

```
Link -rt DRVW=0 ∂
     "{Libraries}"A5DRVRRuntime.o   # This must be first ∂
     "{Libraries}"Runtime.o         # This contains A5Lib ∂
     MyDeskAcc.p.o                  # This is your compiled object code ∂
     "{Libraries}"Interface.o       # You might need this ∂
     "{PLibraries}"PasLib.o         # and/or this ∂
     -sg "MyDeskAcc=Main,%A5Init"   # This maps all segments into MyDeskAcc ∂
     -o MyDeskAcc.DRVW
```

The Link command will generate two warnings about Interface.o and PasLib.o not being needed; the Appendix A example doesn't use them, but your own desk accessory might. You might also use other libraries, such as {Libraries}AppleTalk.o, or {PLibraries}SANELib.o.

Appendix B is the Rez source to build this example desk accessory. If you place this in the file "MyDeskAcc.r", you can use this Rez command to build the DRVR resource:

```
Rez -c DMOV -t DFIL MyDeskAcc.r -o MyDeskAcc
```

To try out the desk accessory, use the Font/DA Mover to install it in your System file:

```
"Font/DA Mover" MyDeskAcc
```

Finder 5.3 provides its own DeskHook procedure which assumes that A5 is its own. If you run MyDeskAcc under this Finder, it will crash when you close the window. This has been fixed in the next release of the Finder. It will work with most other applications, such as the MPW Shell. We would be interested in hearing which other applications crash with this desk accessory; please contact the Development Systems group if you find any.

Appendix A. Pascal Source for a Desk Accessory using Global Data.

```
{     Desk accessory with UNIT globals.  Doesn't do much,
      serves as an example of the layout of A5DRVRRuntime calls.
      Notice how much simpler (and shorter) it is than the example
      in the Global Data Allocation Library document.

      Steve Hartwell, July, 1986.
      Copyright Apple Computer, 1986.  All rights reserved.
}
UNIT        MyDeskAccessory;

INTERFACE
USES        MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf;

VAR         { Our UNIT Globals }

      MyWindow:           WindowPtr;    { The desk accessory's window }
      MyLastCommand:      INTEGER;      { Used for UNDO }

{ Standard UNIT interface for A5DRVRRuntime.o (Same as DRVRRuntime.o) }

FUNCTION DRVROpen         (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
FUNCTION DRVRControl      (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
FUNCTION DRVRStatus       (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
FUNCTION DRVRPrime        (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
FUNCTION DRVRClose        (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;

IMPLEMENTATION

FUNCTION DRVROpen;
VAR   savePort:     GrafPtr;
      wRect:        Rect;

CONST
      wName = 'Desk Accessory with UNIT Globals';

BEGIN

{ If we already have a window then we're already open }
{ so there's nothing else for us to do, just return.  }

      IF (dCtl^.dCtlWindow <> NIL) THEN BEGIN
            DRVROpen := NOErr;
            EXIT(DRVROpen)
      END;
                                    { Otherwise, get ourselves a window }
      GetPort(savePort);
      SetRect(wRect, 10, 322, 500, 338);

{ Store the WindowPtr in the global MyWindow }

      MyWindow := NewWindow(NIL, wRect, wName, TRUE, 0, NIL, TRUE, 0);
      WindowPeek(MyWindow)^.WindowKind := dCtl^.dCtlRefNum;
      dCtl^.dCtlWindow := Ptr(MyWindow);
      SetPort(savePort);

      DRVROpen := NOErr;
END;
```

```
FUNCTION DRVRPrime;             { Not used in a desk accessory }
BEGIN
     DRVRPrime := NOErr;
END;
FUNCTION DRVRStatus;            { Not used in a desk accessory }
BEGIN
     DRVRStatus := NOErr;
END;

FUNCTION DRVRControl;
BEGIN

{ Use our global data, such as MyLastCommand }

     CASE ctlPB^.csCode OF
     accUndo:      IF MyLastCommand = accPaste THEN BEGIN
                         { Execute undo Paste }
                   END;
     accEvent:
           BEGIN  { The only accEvent is update, no need to check csParam }
                  BeginUpdate(MyWindow);
                  SetPort(MyWindow);
                  TextMode(SrcCopy);
                  TextFont(Monaco);
                  TextSize(9);
                  MoveTo(6,10);
                  DrawString('This wouldn''t work without globals!');
                  EndUpdate(MyWindow);
           END;
     accCursor:    ;              { Ignore cursor change requests }
     OTHERWISE
                   SysBeep(30);
     END;

     DRVRControl := NOErr;
END;

FUNCTION DRVRClose;
BEGIN

{ Throw away our window on close }

     DisposeWindow(MyWindow);
     dCtl^.dCtlWindow := NIL;

{ We return NOErr (0), so A5DRVRRuntime will dispose of our       }
{ global data area for us.  If we wanted to, we could return     }
{ a nonzero value, and it would keep our global area around for  }
{ us.  This would retain the value of our global variables,      }
{ such as MyLastCommand, etc.                                    }

     DRVRClose := NOErr;
END;

END. { of UNIT MyDeskAccessory }
```

Appendix B. Rez source for a Desk Accessory with Global Data.

```
/*
 * Resource compiler input for MyDeskAcc.
 *
 * Steve Hartwell, July, 1986
 * Copyright Apple Computer, Inc. 1986.
 * All rights reserved.
 */

#include "MPWTypes.r"                   /* To get 'DRVW' type */

type 'DRVR' as 'DRVW';                  /* Map 'DRVW' => 'DRVR' */

resource 'DRVR' (12, "\0x00MyDeskAcc", purgeable) {
     dontNeedLock,                  /* OK to float around, not saving ProcPtrs */
     dontNeedTime,                  /* No need for periodic Control calls */
     dontNeedGoodbye,               /* No special requirements */
     noStatusEnable,
     ctlEnable,                     /* Desk accessories only do Control calls */
     noWriteEnable,
     noReadEnable,
     0,                             /* drvrDelay tick count, not used */
     updateMask,                    /* This DA only handles update events */
     0,                             /* This DA has no menu */
     "MyDeskAcc",                   /* DRVR name isn't used by the DA */
     $$resource("MyDeskAcc.DRVW", 'DRVW', 0)
};
```