SerialShimLib

Interface Specification/Guide

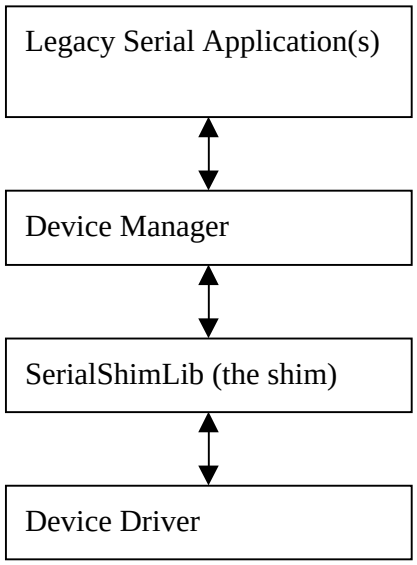
CPU Software

Version: 1.0.1

Date: 9/10/99

Overview

The SerialShimLib is intended to provide an abstraction for serial drivers when in a transient environment, such as USB. For the sake of this document the following diagram outlines the basic “architecture” and defines some of the terms used.



The terms shim, serial shim and SerialShimLib are synonymous and refer to the thin layer that provides the abstraction from the Device Manager and the Communications ToolBox Resource Manager (CRM). The term driver and Device Driver refer to the actual driver that controls the hardware or interfaces to the underlying hardware.

The shim provides an abstraction that hides the details of the Unit Table DRVR and CRM mechanisms. It also provides basic housekeeping when devices “disappear” and “re-appear”. This is important in numerous environments especially USB where devices can be “hot plugged/unplugged”, sometimes inadvertently (someone disconnects an upstream hub) while applications are open.

The shim stays in memory and manages calls from open applications, returning errors, after the driver has been removed. Once the application closes, the shim cleans up and removes device entries from the Unit Table and the CRM.

Interface to the Shim

The interface with the shim is fairly straightforward and both the driver and the shim make use of the Code Fragment Manager (CFM) to accomplish the appropriate linkage. There are basically three calls the driver makes to the shim. One to install the driver, one to remove the driver and one to complete any deferred I/O. The shim only has one call to a driver which uses a selector code to indicate the action.

Installing a driver

When a device is detected (e.g. enumerated via USB) the driver that controls the device is loaded and when ready makes the initial call to the shim.

```
OSErr SerialShimInstallDriver(SerialShimInterface IntBlk, ShimRefNum *ref)
```

This call uses an interface parameter block which contains information the shim uses to register the driver with the Device Manager and the CRM. It returns a shim reference number which must be used on all subsequent calls to the shim. Refer to the USBModem example for exact details on how this call is setup and made. An error is returned if the install can not take place for whatever reason.
The interface block contains the following:

```
typedef   struct     SerialShimInterface
{
          StringPtr                 DRVRInName;
          StringPtr                 DRVROutName;
          StringPtr                 CRMName;
          IconPtr                   CRMIcon;
          UInt32                    MaxSpeed;
          UInt32                    RefCon;
          CFragConnectionID         ConnID;

} SerialShimInterface;
```

The DRVRInName is the name used by the Device Manager for the input side of the driver. Refer to Inside Macintosh: Devices for more information about this. The name must be a valid DRVR name such as .XYZIn. If it is not the shim substitutes the default name of .SSIn instead. The shim also makes these names unique, by appending a number (.SSIn1) if necessary.

The DRVROutName is the name used by the Device Manager for the output side of the driver. The name must be a valid DRVR name such as .XYZOut. If it is not the shim substitutes the default name of .SSOut instead. The shim also makes these names unique,

by appending a number (.SSOut1) if necessary. As these names are pairs the shim will make sure that both are valid before using them.
The CRMName is the name used to register with the Communications Resource Manager. There is no checking done on this name it is passed to the CRM as is. If this is a null string no registration is done with the CRM.

The CRMIcon is the icon displayed in the Connect Via or Connect Port dialog box that applications use to select a device. CRMName is displayed under the icon. This field is ignored if the CRMName is null.

MaxSpeed is the maximum DTE speed (in bits per second), required by the CRM.

RefCon is a general purpose 32 bit value passed to the shim. The shim then returns this value on all subsequent calls to the driver.

The ConnID is the Code Fragment Managers identification of the driver. It allows the shim to call back into the appropriate driver. The USBModem example shows how to obtain the Connection ID.

Removing a driver
When a driver is terminated, the device has been turned off or physically removed (in the case of USB unplugged) the following call must be made before the driver terminates.

```
OSErr SerialShimRemoveDriver(ShimRefNum ref, Boolean forced)
```

The shim reference number must be used to indicate to the shim which driver is being terminated. The shim locates the driver and tries to remove the Unit Table entry and the CRM entry. If this is successful no error is returned. If, for example, an application is open then the Unit Table entry cannot be removed. The shim will mark the driver as “unavailable” and will periodically try to remove it. Any calls made to the driver at this point will result in an error being returned. If Forced is set to true, this indicates to the shim that the driver will be unloaded and the shim must manage removing the Unit Table entry. If Forced is set to false, this indicates the driver will continue to be around and the shim should not periodically try to remove the Unit Table entry. The assumption here is that the driver will continue to issue SerialShimRemoveDriver until no error is returned. The driver will still be marked as “unavailable” and an error will be returned to the application. If Forced is true and the driver cannot be removed the function returns “pending” (1) to indicate to the driver that the shim is in the process of cleaning up.

Forced set to true is **strongly** recommended as this allows the shim to clean up with minimal or no effort on the part of the driver.

Deferred I/O

Any deferred I/O, asynchronous requests not completed (returned pending to the application) need to make the following call once the I/O has actually completed.

```
void SerialShimIOComplete(ShimRefNum ref, ParmBlkPtr pb)
```

The shim reference number indicates which driver has completed the I/O and the parameter block is the completed I/O parameter block. See the USBModem example for more details on deferred I/O handling.

Interface to the Driver

The interface to the driver is again straightforward and uses the CFM to handle this. The Connection ID passed into the Shim, in the interface block of the SerialShimInstallDriver call, is used to locate the driver and then communicate with it.

Passing commands to the driver

Once a command, in this context this means anything received from the Device Manager destined for a driver (i.e. Open, Close, Prime (read/write), Control and Status), is received by the Shim and is passed on to the driver. The Shim does some basic checking, the driver is "available" (i.e. not been unloaded – the remove call has not been made) and then builds the selector and makes the following call.

```
OSErr SerHAL_Entry(UInt16 HdwSelector, ParmBlkPtr pb, UInt32 RefCon)
```

The Hardware Selector code is defined as follows:

```
enum
{
        SerHAL_Initialize       =       0,      // Open
        SerHAL_Terminate        =       1,      // Close
        SerHAL_Read             =       2,      // Prime (Read)
        SerHAL_Write            =       3,      // Prime (Write)
        SerHAL_SetConfiguration =       4,      // kSERDConfiguration
        SerHAL_SetInputBuffer   =       5,      // kSERDInputBuffer
        SerHAL_SetFlowControl   =       6,      // kSERDSerHShake/kSERDHandshake
        SerHAL_SetBreak         =       7,      // kSERDSetBreak/kSERDClearBreak
        SerHAL_SetDTERate       =       8,      // kSERDBaudRate/kSERD115kBaud/kSERD230kBaud
        SerHAL_SetDTR           =       9,      // kSERDAssertDTR/kSERDNegateDTR
        SerHAL_SetParity        =       10,     // kSERDSetPEChar/kSERDSetPEAltChar
        SerHAL_SetXOffFlag      =       11,     // kSERDSetXOffFlag/kSERDClearXOffFlag
        SerHAL_SendXOn          =       12,     //kSERDSendXOn/kSERDSendXOnOut
        SerHAL_SendXOff         =       13,     // kSERDSendXOff/kSERDSendXOffOut
        SerHAL_Miscellaneous    =       14,     // kSERDMiscOptions
        SerHAL_GetBuffer        =       15,     // kSERDInputCount
        SerHAL_GetStatus        =       16,     // kSERDStatus
        SerHAL_GetVersion       =       17,     // kSERDVersion
        SerHAL_ControlExtend    =       18,     // Unrecognized code
        SerHAL_StatusExtend     =       19,     // Unrecognized code
        SerHAL_KillRead         =       20,     // KillC ode
        SerHAL_KillWrite        =       21      // KillCode
};
```

Most of the codes are self explanatory, however one or two may need further explanation. SerHAL_Initialize is the result of an Open call. SerHAL_Terminate is the result of a Close call. The SerHAL_ControlExtend and SerHAL_StatusExtend are the result of

csCodes not recognized by the Shim and could be driver specific. They are passed to the driver as is and can be acted upon or ignored as appropriate.

The SerHAL_KillRead and the SerHAL_KillWrite events result when the Serial Driver has been issued a KillCode on a parameter block. The Parameter block is the same one that was passed to the shim by the Device Manager. The RefCon parameter is the same one that was passed to the shim in the interface block of the SerialShimInstallDriver call. It is a general purpose field for driver use and is returned to the driver unmodified.