AVStream Capture Driver
Osprey-100/2X0
Osprey-540/560
Version 3.1.0
Developers' SDK Guide

May 2004

Table of Contents

Overview of the SDK

The Osprey custom properties provide programmatic access to enhanced and non-standard features of the Osprey DirectShow AVStream driver.

The best way to get an overall picture of the custom properties is to look at the property tabs or sheets that are displayed as driver properties. To see these pages, run SwiftCap and click the "circuit board" Video Properties toolbar button. Or, in GraphEdt, right-click on an Osprey device and select "Filter Properties…".

The SDK interfaces described here are for the "Osprey Multimedia" devices – Osprey-100 and 2X0 – as well as for the Osprey-540/560.

Interfaces

- **The Pin Management interface:** A helper interface for controlling custom parameters that can be different for each pin of a device. The description of this interface explains the pin ID scheme used by the Crop and Logo interfaces, so it is recommended that you read the background portions of this section if you plan to use either of these interfaces. On the driver's custom property sheet tabs, the "Pin Select" control groups on the "Size and Crop" and "Logo Tabs" implement the Pin Management interface.

- **The Crossbar interface:** This interface is an alternative implementation of the Video Decoder and Crossbar standard interfaces. The functionality and parameter structures are identical to the standard interfaces, but they access the video capture filter directly rather than a logically separate video crossbar filter. It is therefore possible to interface to crossbar and decoder functions in a graph that does not have a crossbar. On the property sheet tabs, this interface is titled "Input".

- **The RefSize interface:** Defines the reference size of the incoming video. The reference size is used directly in sizing the input analog video, and also provides the sizing framework for the Crop interface.

- **The Size and Crop interface:** Defines cropping parameters for each pin. Also sets the default video output size for each pin. On the property sheet tabs, this interface is titled "Size and Crop".

- **The Logo interface:** Controls placement and display mode of logo bitmaps on the video.

- **The Device Configuration interface:** Miscellaneous controls and items of information that are available on a per-video-device basis. On the property sheet tabs, this interface is titled "Device".

SDK Contents

All SDK c++ files are contained in the \inc and \samples directories under your SDK root directory.

The following three .h files contain the definitions for the Osprey custom properties:

- \inc\o200avsCom.h
- \inc\OspProps.h
- \inc\OspreyAudio.h

Note that o200avsCom.h #includes OspProps.h. OspreyAudio.h appears to be referenced only by the AudioConfig applet.

The crossbar interface (IOspreyCrossbar) also requires the DirectShow file #include ksmedia.h.

Most of the sample applications also reference .cpp and .h files in the \samples\common directory.

The samples directory includes code samples for the following applets:

- AudioConfig
- CCApp
- CropApp
- LogoApp
- TCApp
- VbiGraph
- VidControl

There are also three filters:

- CCFilter.ax (CCTransFilter)
- OverlayRect.ax (COverlayRectTransFilter)
- TCOverlay.ax

Also included is the following as-is sample:

- SwiftCap

The section “Sample Applications” of this document describes these samples.

Building the SDK

We are currently building the SDK with Microsoft Visual Studio 6.0, and 6.0 is the minimum level required. To our knowledge there should be no problems using version 7.0. You will need to install the unicode versions of the MFC libraries.

In addition to Visual Studio, you will need to install the following two Microsoft components:

- The Platform SDK.
- The DirectX SDK, version 9.0 or later.

Before building the samples, set environmental variables as follows. The paths below suppose that you have installed the PSDK at c:\psdk, and the DXSDK at c:\dxsdk.

- MSSDKHOME = c:\psdk
- DXSDKROOT = c:\dxsdk
- DXMCLASS = c:\dxsdk\samples\c++\directshow\baseclasses

Compile the Directshow base classes and libraries in the DXMCLASS directory. You will want to compile both release and debug verions.

Add the MS Platform SDK include directory to the Visual C++ search paths. In Visual C++, select the Options choice from the Tools menu. On the Directories Tab, add the include path and move it to the top of the list.

You can use the buildAll.bat script to build all the sample applications. Or you can build them one at a time in Visual Studio from their individual .dsw or .dsp workspace files.

Support and Feedback

Support for the SDK is by email only.

Email questions and comments to support@ospreyvideo.com. Reference "Osprey AVStream SDK" or "Osprey MM 3.X SDK".

Make sure your question is specifically about the custom properties, not about general DirectShow issues. Obtain the DirectX 9.0 documentation directly from Microsoft and learn DirectShow basics before contacting us.

Porting Guide – from Osprey VfW SDK to Osprey AVStream SDK

The SDK for the Osprey Video for Windows driver implements some capabilities that are built in to the Microsoft DirectShow interface to AVStream drivers, and some other capabilities that continue to be implemented as Osprey custom properties. Here is a summary of the differences.

Function	***Osprey VfW Driver***	***Osprey AVStream Driver***
Device Select	MMAC_DEVICEN_SET	DirectShow standard
Input Select	MMAC_o100SRC_CONTROL	IAMCrossbar or IOspreyCrossbar
Video Standard (ntsc, pal, secam)	MMAC_o100SRC_CONTROL	IAMAnalogVideoDecoder or IOspreyCrossbar
Brightness, etc.	MMAC_o100SRC_CONTROL	IAMVideoProcAmp
Cropping	MMAC_o100_CROP_CONTROL	IOspreyCrop
Logo	MMAC_o100_LOGO_CONTROL	IOspreyLogo
Closed Caption	MMAC_o100_CC_CONTROL, MMAC_o100_CC_CONTROL2	DirectShow CC pin

VBI (vertical blanking interval)	MMAC_o100_VBI_CONFIG, MMAC_o100_VBI_CONTROL	DirectShow Vbi pin
Ct878 Register Peek / Poke	MMAC_o100_REGISTER_GET, MMAC_o100_REGISTER_SET	IOspreyDevice

The descriptions immediately below are not meant to be a complete description of the AVStream custom properties, but only to indicate the porting pathway. The custom properties are described in full detail in a later section of the document.

Device Select

VfW did not provide a very convenient way to identify and select between multiple devices of the same type. The Osprey VfW SDK therefore provided several proprietary methods for assisting this process.

The DirectShow / AVStream environment provides very adequate methods for enumerating and selecting devices, so there are no custom Osprey extensions. The code samples show how to access devices via standard COM-based methods. All of the applets except VbiGraph have this code in them.

Obtaining an interface to the Custom Properties

In the VfW SDK, the interface to the driver is message-based and built around three Microsoft Platform SDK multimedia functions, OpenDriver(), CloseDriver(), and SendDriverMessage().

The custom properties of the AVStream driver are accessed by COM-based DirectShow methods. Once an Osprey device has been selected and an interface obtained to it, query its interface as follows. The first example obtains the DShow standard Video Proc Amp interface; the second example obtains the Osprey custom Logo interface.

```
IAMVideoProcAmp* pVPA;
hr = baseFilter->QueryInterface(IID_IAMVideoProcAmp,(void **)&pVPA);

IOspreyLogo* pLogo;
hr = baseFilter->QueryInterface(IID_IOspreyLogo,(void **)&pLogo);
```

Once you have obtained the interface pointer pVPA, pLogo, etc., you can call the methods that are defined for that interface – for example,

```
// . . .
pVPA ->Get(lProperty, &lValue, &lFlags);
pLogo->Set(&Logo);

// . . .
pVPA ->Release();
pLogo->Release();
```

The methods and parameters for standard DirectShow properties are described in the DirectX 9 documentation. The methods and parameters for Osprey custom properties are described in this document.

Input Select

The Video for Windows environment does not provide built-in support for enumerating, getting, or selecting video inputs. The Osprey VfW SDK provides a custom interface for these purposes as part of the MMAC_o100_SRC_CONTROL message.

In the AVStream environment, there are two ways to control the video inputs.

First, you can use the DShow standard IAMCrossbar interface that is part of the Analog Video Crossbar Filter. The advantage of using the interface is that it is standard and should work the same for all video capture devices that implement a crossbar filter. The disadvantage is the location of this interface as part of the logically distinct Crossbar Filter rather than part of the Video Capture Filter. In practice it is easy to locate the Crossbar Filter in a graph by calling the FindInterface() method of the ICaptureGraphBuilder2 interface.

The second alternative is to use the custom IOspreyCrossbar interface. The disadvantage to this interface is that it is non-standard. Its advantage is that is attached to the logical Video Capture Filter rather than the logical Crossbar Filter, and so it is slightly easier to access, especially if you are not building the graph with ICaptureGraphBuilder2. As a convenience, the IOspreyCrossbar interface also includes functions to enumerate, get, and set the video standard without separately connecting to the IAMAnalogVideoDecoder interface.

Video Standard

The Video for Windows environment does not provide built-in support for querying or selecting the NTSC, PAL, or SECAM video standard. The Osprey VfW SDK provides a custom interface as part of the MMAC_o100_SRC_CONTROL message.

In the AVStream environment, there are two ways to enumerate and set the video standard. First, the DShow IAMAnalogVideoDecoder interface exposed by the Video Capture Filter provides a DirectShow-standard way. Second, the IOspreyCrossbar custom interface to the Osprey video capture driver integrates the video standards functions with input select functions.

Brightness, etc.

The Video for Windows environment does not provide built-in support for brightness, contrast, hue and saturation controls. The Osprey VfW SDK provides a custom interface as part of the MMAC_o100_SRC_CONTROL message.

In the AVStream environment, brightness, contrast, hue and saturation are controlled by the standard DirectShow IAMVideoProcAmp interface of the WDM Video Capture Filter. Three methods are exposed – GetRange (), Get (), and Set(). Refer to the DirectX 9 documentation for their description.

Cropping

Both the Osprey VfW driver and the Osprey AVStream driver implement cropping as a custom interface. The VfW driver implements the message MMAC_o100_CROP_CONTROL and the structure CROP_CTL. The AVStream driver implements the IOspreyCrop interface and the KSPROPERTY_OSPREY_CROP_S structure.

The VfW CROP_CTL structure is basically a subset of the KSPROPERTY_OSPREY_CROP_S structure. The members translate as follows:

- bEnabled -> ulCropEnable
- dwVideoWidth -> ulRefWd
- dwVideoHeight -> ulRefHt
- dwCropXOffset -> ulCropLeft
- dwCropYOffset -> ulCropTop
- dwCropWidth -> ulCropRight – ulCropLeft
- dwCropHeight -> ulCropBottom – ulCropTop

The additional members of KSPROPERTY_OSPREY_CROP_S have functions that have no equivalents in the VfW scheme – refer to the detailed explanation of the Crop interface.

Logo

Logos are a custom feature of both the Osprey VfW and AVStream drivers. The VfW driver implements the message MMAC_o100_LOGO_CONTROL, and the structure LOGO_CTL. The AVStream driver implements the IOspreyLogo custom interface and the structures KSPROPERTY_OSPREY_LOGO_S and KSPROPERTY_OSPREY_LOGO_OP_S.

The LOGO_CTL structure is basically a subset of the KSPROPERTY_OSPREY_LOGO_S structure. The members map as follows:

- bEnabled -> ulFlags
- dwLogoHt -> ulHeight
- dwLogoWd -> ulWidth
- dwVideoHt -> ulVideoHt
- dwVideoWd -> ulVideoWd
- dwLogoTop -> ulTop
- dwLogoLeft -> ulLeft
- dwKeyColor -> ulKeyColor and ulFlags
- bEmbossed -> ulFlags
- awFName -> awFName

The two command interfaces are somewhat different and you are referred to the main section on logos herein for details.

Closed Caption (CC)

Video for Windows does not support CC as a native stream type. The Osprey VfW driver therefore implements a complete custom CC interface via the MMAC_o100_CC_CONTROL and MMAC_o100_CC_CONTROL2 messages.

The AVStream driver implements the DShow standard CC pin type, PIN_CATEGORY_CC. There are no custom extensions to the CC interface. The interface is significantly different from the Osprey VfW interface – in particular there is no “line mode” CC – all CC data is returned as raw character pairs; line mode decoding, and rendering onto video, are performed by external filters. The CCApp sample shows two CC applications: First, a standard graph is built with the a Line 21 decoder followed by the Overlay Mixer to render CC on preview video; second, a custom filter extracts CC data pairs and returns them directly to the application for display in a text box.

Vertical Blanking Interval (VBI)

Video for Windows does not support VBI as a native stream type. The Osprey VfW driver therefore implements a complete custom VBI interface.

The AVStream driver for the Osprey-100 and -200 series devices implements the DShow standard VBI pin type, PIN_CATEGORY_VBI. There are no custom extensions to the VBI interface. Refer to the VbiGraph applet as a sample of how to set up a VBI graph and capture VBI data into a user-mode application.

Since the Osprey-540/560 hardware does not deliver raw VBI, the AVStream driver for these devices does not provide a standard VBI pin. The driver has an option to expose a VBI pin for enumeration purposes as a workaround for a problem in WME9; however, this pin is not functional.

Ct878 Register Peek / Poke

Both the Video for Windows and AVStream drivers provide custom interfaces to peek and poke values to and from the Ct878 registers. The VfW driver uses the MMAC_o100_REGISTER_GET/ SET messages and the REGISTER_CTL structure. The AVStream driver uses the GetRegister() and SetRegister() functions that are part of the IOspreyDevice interface. The KSPROPERTY_OSPREY_REGISTER_S struct allows you to set and reset individual bits of a register in a single read-modify-write operation.

The Osprey AVStream Custom Interfaces

The information here assumes that you have basic familiarity with DirectShow and COM programming concepts. It does not attempt to document standard DirectShow properties that the driver supports that are described in Microsoft documentation. The code samples in this section are as succinct as possible. They may not have complete error checking. Refer to the SDK sample apps for complete code samples.

The interface functions are implemented in the DLL o200avs.ax that is installed when the Osprey AVStream driver is installed. Note that o200avs.ax is the interface DLL for the Osprey-540/560 as well as the -100 and -200 devices.

The Pin Management Interface

DirectShow does not inherently support persistent or static sets of custom data for individual pins maintained by drivers. A DirectShow “pin” is a volatile entity that is created, run, and then destroyed.

It is convenient, however, for the driver to maintain its own static data sets on a per-pin basis, so that standard applications will work with custom features of the driver – in particular with SimulStream, which allows multiple capture pins per device. For example, the driver supports cropping specifications and logo specifications that are per-pin. That is, these specifications can be different for a device’s capture and preview pins. Furthermore, if SimulStream is installed, these specification can be different for different capture pin instances and different preview pin instances.

In the non-SimulStream case, the Pin Management interface is of minor importance, since each device has just one capture pin and one preview pin. In the SimulStream case, however, the existence of multiple capture pins and multiple preview pins make the situation more complicated. The problem is how to link multiple static data sets to multiple dynamic pin instantiations. Static pin data sets can be referred to by number or name, but DirectShow pin instantiations do not have any method for mapping pin names or numbers other than (1) the basic pin type, capture or preview, and (2) the temporal order in which the pins are created.

The Pin Management interface organizes pin data sets by pin type, capture or preview, and by instance number within each type. That is, in the SimulStream case there can be Capture Pin 0, Preview Pin 0, Capture Pin 1, Preview Pin 1, and so forth.

Each of these pins can have an entirely different and independent data set. An alternate model that is useful for many applications is that the Capture and Preview pins are grouped as pairs, so that for example Capture Pin 0 and Preview Pin 0 have the same settings. Osprey's SwiftCap is an example of an application that uses this simplified model. The Pin Management scheme provides a convention that supports both paired and non-paired models of pin settings.

With SimulStream enabled, the video pins of standard applications are mapped to pin instances in the order in which they start. The capture pin of the first application process or instance to start will use the default size, crop, and logo settings that have been set up for Capture Pin 0. The capture pin of the second process started will map as Capture Pin 1. Preview pins map similarly. If Application 0 has two capture pins but no preview pin, and Application 1 has one of each, Application 1's pins will be Capture Pin 2 and Preview Pin 0.

How Pins are Designated

Throughout the Osprey custom properties, the following declarations are used to designate pins.

```
#define PIN_TYPE_CAPTURE  0
#define PIN_TYPE_PREVIEW  1
#define PIN_TYPE_PAIRED   0xFF

typedef union
{
  struct
  {
    UCHAR ucInstance;
    UCHAR ucType;
  };
  USHORT us;
}
PIN_ID, *PPIN_ID;
```

ucType: May be PIN_TYPE_CAPTURE (0x00) or PIN_TYPE_PREVIEW (0x01), or PIN_TYPE_PAIR (0xFF), referring to a capture / preview pin pair.

ucInstance: A pin instance number. If SimulStream is not installed, the instance number must be 0x00. If SimulStream is installed, instance number can be in the range 0x00 .. 0xFE. 0xFF is reserved.

us: A shorthand reference in which the upper byte is the pin type and the lower byte is the instance of that type. For example, 0x0100 is preview pin 0. Use 0xFFFF to mean "undefined" – that is, that the PIN_ID is not referring to a defined pin.

How Pin Data is Accessed

The Crop and Logo interfaces, in particular, access data that is organized per-pin. When you make a call such as a Get() or Set() to one of these interfaces, the structure that you pass includes a PIN_ID member which you set before entering the function to specify which pin's data you want to read or write.

If PIN_ID.ucType is PIN_TYPE_CAPTURE or PIN_TYPE_PREVIEW, it is clear that data belonging respectively to a capture pin or a preview pin will be accessed. If, however, PIN_ID.ucType is PIN_TYPE_PAIRED, the following will happen:

- A Get() with PIN_TYPE_PAIRED reads the data for the referenced capture pin. Really, there is no difference between a Get() with PIN_TYPE_CAPTURE and a Get() with PIN_TYPE_PAIRED – the latter is there for convenience, and does not cause the driver to perform any additional hidden operations.

- A Set() with PIN_TYPE_CAPTURE writes the same data to both the capture and preview pin. It is no different from a Set() with PIN_TYPE_CAPTURE followed by a Set() with PIN_TYPE_PREVIEW. Again, there are no additional hidden operations performed inside the driver.

The Pin Lock Function

Normally, when a user using the Size/Crop or Logo pages from an application can choose to apply settings to a capture pin, a preview pin, or to both. In some cases their choice could be in conflict with the expectations of the application. For example, the Osprey SwiftCap application always expects to the capture and preview settings to be the same; if the user applies preview-only settings in the driver's property pages, SwiftCap's behavior will be confusing or incorrect.

The solution is a function that locks the capture / preview / both choice for the duration of the next Osprey property pages invocation, so that the capture / preview / both control group (or, with SimulStream enabled, the "Apply to Both Capture and Preview" checkbox) is greyed.

Summary of Pin Management Functions

- Enum() – Use this read-only function to retrieve a list of all pins that are currently defined. This function also tells you whether SimulStream is enabled on the device.

- GetCurrentPin() – This read-only function retrieves a CurrentPin variable for the device. The driver does not use CurrentPin internally; rather, CurrentPin establishes a convention for accessing per-pin data sets. If you follow this convention, the o200avs.ax property sheets and SwiftCap, among others, will cooperate with your pin selections.

- SetCurrentPin() – A write-only function to store CurrentPin for the device.

- DeletePin() – A write-only function to delete all data for the specified pin id.

- GetPinLock() and SetPinLock() – Functions to get and set the pin locking status.

Pin Management Functions – Non-SimulStream

When SimulStream is not installed for the device, a single preview pin and a single capture pin are available for all applications. It is possible for the capture pin and the preview pin to have different settings. Or, it is possible to treat the two as a pair with the same settings.

The Pin Management Enum() function is not very meaningful in the non-SimulStream case, since there are always one capture pin and one preview pin. It does return a flag that is a convenient way to find out if SimulStream is in fact enabled on the device.

GetCurrentPin() can return three possible values:

- 0x0000 means that the capture pin is the current target for parameter setting.
- 0x0100 means that the preview pin is the current target for parameter setting.
- 0xFF00 means that the capture and preview pins are linked. For read operations, parameters are read from the capture pin. For write operations, both the capture and preview pins are set to the same values.

Call SetCurrentPin() to save the target CurrentPin – 0x0000, 0x0100, or 0xFF00.

The DeletePin() function is not applicable to non-SimulStream – the definitions for the single capture pin and single preview pin cannot be deleted.

The SetPinLock() function disables the Pin Select control group (“Capture / Preview / Both”) for the current invocation of the Size/Crop and Logo pages in the current user-mode process. The session will reflect the pin type setting that was chosen prior to the SetPinLock() call.

Pin Management Functions – SimulStream

When SimulStream is installed for the device, there can be multiple capture pin and preview pin instantiations, and multiple data sets defining the default size, crop, and logo settings for each of those pins. Capture and preview pins can be considered as pairs – capture / preview pin pair 0, capture / preview pin pair 1, etc. The pins in each pair can be allowed to have different settings. Or, they can be conceptually linked so that the same settings are always maintained for each.

The Pin Management Enum() function is important in the non-SimulStream case. Details are given in the next subsection on how to use Enum() to obtain the list of pins that are currently defined in the registry.

GetCurrentPin() can return values as follows. The “??” in each case is the instance number 0x00..0xFE of each pin type:

- 0x00?? means that a capture pin is the current target for parameter setting.
- 0x01?? means that a preview pin is the current target for parameter setting.
- 0xFF?? means that a linked capture and preview pin pair is the target, and parameter changes will be set for both pins. For read operations, parameters are read from the capture pin’s data. For write operations, both the capture and preview pins are set to the same values.

Call SetCurrentPin() to save the target CurrentPin according to this convention.

Call DeletePin() to clean the registry of unneeded pin definitions.

The SetPinLock() function disables the Pin Select control group's "Apply to Both Capture and Preview" checkbox, for the current invocation of the Size/Crop and Logo pages in the current process. The session will reflect the pin type setting that was chosen prior to the SetPinLock() call, so that the pin list will contain either "Capture Pin 0, 1…", "Preview Pin 0, 1…", or (for the "both" case) "Pin 0, 1…".

The Pin Management Enum() Function

Prototype:

```
STDMETHODIMP IOspreyPinMgt::Enum(PKSPROPERTY_OSPREY_PIN_ENUM_S pEnum);
```

The KSPROPERTY_OSPREY_PIN_ENUM_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      ulNPins;
  BOOL       bSSEnabled;
  PIN_ID     aPinId[2];
}
  KSPROPERTY_OSPREY_PIN_ENUM_S,
*PKSPROPERTY_OSPREY_PIN_ENUM_S;
```

Property: The property header. Does not have to be initialized by the client.

ulNPins: On entry, set to the array size of aPinId[]. On return, if the array space passed was sufficient, ulNPins will be set to the number of aPinId[] elements actually returned. If on return the array space is insufficient, ulNPins will be set to the number of aPinId[] elements that must be allocated for a second call to Enum()

bSSEnabled: On return, if TRUE, then SimulStream is enabled for the device.

aPinId[]: The array of Pin Ids. If ulNPins on return > ulNPins on entry then the aPinId[] data is not valid. If ulNPins on return is less than or equal to ulNPins on entry then the first ulNPins-on-return elements of aPinId[] are valid.

Here are recommended steps for use of this function:

```
CcomPtr  <IBaseFilter  > m_pVidCap;
CComQIPtr<IOspreyPinMgt> m_pPinMgt;

m_pPinMgt = m_pVidCap;

if (m_pPinMgt)
{
  PKSPROPERTY_OSPREY_PIN_ENUM_S pEnum;

  ULONG ulNPins = 2;

  pEnum = (PKSPROPERTY_OSPREY_PIN_ENUM_S) new KSPROPERTY_OSPREY_PIN_ENUM_S;

  if (pEnum)
  {
    pEnum->ulNPins = ulNPins;  // mandatory initialization
    hr = m_ pPinMgt->Enum(pEnum);

     // Pin data not valid if pEnum->ulNPins > ulNPins.
```

```
      while ((SUCCEEDED(hr)) && (ulNPins < pEnum->ulNPins))
      {
        ulNPins = pEnum->ulNPins;  // returned array size requirement

        delete pEnum;

        pEnum = (PKSPROPERTY_OSPREY_PIN_ENUM_S) new UCHAR
           [
             sizeof(KSPROPERTY_OSPREY_PIN_ENUM_S) +
             (sizeof(PIN_ID) * (ulNPins - 2))
           ];

        if (pEnum == NULL)
        {
          break;
        }

        pEnum->ulNPins = ulNPins;  // mandatory init
        hr = m_pPinMgt->Enum(pEnum);
      }

      if (pEnum && SUCCEEDED(hr))
      {
        // Now you have the complete pin list . . .
        ulNPins = pEnum->ulNPins;  // might have changed in second Enum()
      }

      delete pEnum;
    }
  }
```

The Pin Management GetCurrentPin() Function

This function returns from the driver the ID of the pin or capture / preview pin pair that is the current target for parameter setting.

Prototype:

```
STDMETHODIMP IOspreyPinMgt::GetCurrentPin(PKSPROPERTY_OSPREY_PIN_S pPinOp);
```

The KSPROPERTY_OSPREY_PIN_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  PIN_ID     PinId;
}
  KSPROPERTY_OSPREY_PIN_S,
*PKSPROPERTY_OSPREY_PIN_S;
```

Property: The property header. Does not have to be initialized by the client.

PinId: Not initialized on entry. On exit, contains the current pin ID.

PinId.ucType will be PIN_TYPE_CAPTURE, PIN_TYPE_PREVIEW, or PIN_TYPE_PAIRED for current pin type set respectively to a capture pin, a preview pin, or a capture / preview pin pair.

In the non-SimulStream case, PinId.ucInstance will always be 0x00.

In the SimulStream case, PinId.ucInstance will be in the range 0x00..0xFE.

The Pin Management SetCurrentPin() Function

This function sets in the registry the ID of the pin or capture / preview pin pair that is the current target for parameter setting.

Prototype:

```
STDMETHODIMP IOspreyPinMgt::SetCurrentPin(PKSPROPERTY_OSPREY_PIN_S pPinOp);
```

The KSPROPERTY_OSPREY_PIN_S structure is shown GetCurrentPin() above.

Property: The property header. Does not have to be initialized by the client.

PinId: On entry contains the pin ID, according to the convention described for GetCurrentPin() above.

The Pin Management DeletePin() Function

This function delete static setup data in the registry for the designated pin or pin pair. Note that it has to do with static registry data only. It does not affect a DirectShow pin instance that is already running.

Prototype:

```
STDMETHODIMP IOspreyPinMgt::DeletePin(PKSPROPERTY_OSPREY_PIN_S pPinOp);
```

The KSPROPERTY_OSPREY_PIN_S structure is shown GetCurrentPin() above.

Property: The property header. Does not have to be initialized by the client.

PinId: On entry contains the pin ID on entry. PIN_TYPE_CAPTURE, PIN_TYPE_PREVIEW, or (to delete both pins of a pair) PIN_TYPE_PAIRED. On exit, PinId is unchanged. If PinId.ucInstance is set to 0, the function will have no effect – the pin 0 specifications are not deletable.

The Pin Management Get/SetPinLock() Functions

Prototypes:

```
STDMETHODIMP GetPinLock(PKSPROPERTY_OSPREY_PINLOCK_S pPinLock);
STDMETHODIMP SetPinLock(PKSPROPERTY_OSPREY_PINLOCK_S pPinLock);
```

The KSPROPERTY_OSPREY_PINLOCK_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      ulLock;
}
  KSPROPERTY_OSPREY_PINLOCK_S,
```

```
*PKSPROPERTY_OSPREY_PINLOCK_S;
```

Property: The property header. Does not have to be initialized by the client.

ulLock: For SetPinLock(), set ulLock to 1 to lock the Pin Select control. Set ulLock to 0 to unlock it. Since the lock automatically resets at the end of each property page invocation, it is unlikely that you will ever have to call SetPinLock() with ulLock 0. For GetPinLock(), ulLock does not have to be initialized, and on return contains the current lock status.

Here is a sample, from SwiftCap, of how to use this function in conjunction with SetCurrentPin(), to force all user properties settings to apply to both the preview and capture pins.

```
void CaptureGraph::SetPinMode()
{
  if (m_pPinMgt == NULL)
    return;

  // couple the capture and preview pin set
  KSPROPERTY_OSPREY_PIN_S PinSet;
  PinSet.PinId.ucInstance = theApp.GetPinInstance();
  PinSet.PinId.ucType     = PIN_TYPE_PAIRED;
  m_pPinMgt->SetCurrentPin(&PinSet);

  // lock the "both" setting
  KSPROPERTY_OSPREY_PINLOCK_S PinLock;
  PinLock.ulLock = 1;
  m_pPinMgt->SetPinLock(&PinLock);
}
```

The Crossbar Interface

The Crossbar interface contains the following crossbar and video decoder functions. Note that the structures used in this interface are Microsoft DirectShow structures defined in ksmedia.h.

```
STDMETHODIMP
IOspreyCrossbar::GetCaps (PKSPROPERTY_CROSSBAR_CAPS_S    pCaps);
STDMETHODIMP
IOspreyCrossbar::PinInfo (PKSPROPERTY_CROSSBAR_PININFO_S pPinInfo);
STDMETHODIMP
IOspreyCrossbar::CanRoute(PKSPROPERTY_CROSSBAR_ROUTE_S   pRoute);
STDMETHODIMP
IOspreyCrossbar::GetRoute(PKSPROPERTY_CROSSBAR_ROUTE_S   pRoute);
STDMETHODIMP
IOspreyCrossbar::SetRoute(PKSPROPERTY_CROSSBAR_ROUTE_S   pRoute);

STDMETHODIMP
IOspreyCrossbar::GetDecoderCaps    (PKSPROPERTY_VIDEODECODER_CAPS_S pCaps);
STDMETHODIMP
IOspreyCrossbar::GetDecoderStandard(PKSPROPERTY_VIDEODECODER_S pStandard);
STDMETHODIMP
IOspreyCrossbar::SetDecoderStandard(PKSPROPERTY_VIDEODECODER_S pStandard);
STDMETHODIMP
IOspreyCrossbar::GetDecoderStatus  (PKSPROPERTY_VIDEODECODER_STATUS_S
                                    pStatus);
```

Sample usage:

```
// The property structs are DShow standard, in ksmedia.h:
```

```
#include <ksmedia.h>

CcomPtr  <IBaseFilter    > m_pVidCap;
CComQIPtr<IOspreyCrossbar> m_pCrossbar;

m_pCrossbar = m_pVidCap;

if (m_pCrossbar)
{
  // Try out the crossbar functions:
  KSPROPERTY_CROSSBAR_CAP_S     Caps;
  KSPROPERTY_CROSSBAR_PININFO_S PinInfo;
  KSPROPERTY_CROSSBAR_ROUTE_S   Route;
  HRESULT hr;

  hr = m_pCrossbar->GetCaps(&Caps);

  PinInfo.Direction = KSPIN_DATAFLOW_IN;
  for (ULONG ul = 0; ul < Caps.NumberOfInputs; ul++)
  {
    PinInfo.Index = ul;
    hr = m_pCrossbar->PinInfo(&PinInfo);

    // Save the returned info . . .
  }

  hr = m_pCrossbar->GetRoute(&Route);
  // Simplest possible exercise . . .
  if (Route.IndexInputPin != 0)
  {
    Route.IndexInputPin = 0;
    hr = m_pCrossbar->SetRoute(&Route);
  }

  // Exercise the video decoder functions:
  KSPROPERTY_VIDEODECODER_CAPS_S   DecoderCaps;
  KSPROPERTY_VIDEODECODER_S        Standard;
  KSPROPERTY_VIDEODECODER_STATUS_S Status;

  hr = m_pCrossbar->GetDecoderCaps(&DecoderCaps);

  // Set SECAM video if supported, and not already set:
  if (DecoderCaps.StandardsSupported & KS_AnalogVideo_SECAM_B)
  {
    hr = m_pCrossbar->GetDecoderStandard(&Standard);

    if (!(Standard.Value & KS_AnalogVideo_SECAM_Mask))
    {
      Standard.Value = KS_AnalogVideo_SECAM_B;
      hr = m_pCrossbar->SetDecoderStandard(&Standard);
    }
  }

  // Determine if active video is present:
  if (DecoderCaps.Capabilities &
      KS_VIDEODECODER_FLAGS_CAN_INDICATE_LOCKED)
  {
    hr = m_pCrossbar->GetDecoderStatus(&Status);
    if (Status.SignalLocked)
    {
      ;  // signal is locked
    }
}
```

Note that frequent reference will be made to the "output" pin. This pin is the output of the logical crossbar filter that is implicitly embedded in the Osprey capture driver along with a logical video capture filter. There is just a single crossbar output pin that is connected to the single analog input pin on the capture filter. It is distinct from the multiple outputs of the video capture filter, which may include Capture, Preview, CC (Closed Caption), and VBI (Vertical Blanking Interval) pins.

For further information about the standard crossbar and decoder interfaces, IAMCrossbar and IAMAnalogVideoDecoder, refer to the Microsoft Windows XP DDK documentation.

The Crossbar GetCaps() function

This function implements the functionality of the standard KSPROPERTY_CROSSBAR_CAPS interface. It is a read-only function. The function prototype is as follows:

```
STDMETHODIMP IOspreyCrossbar::GetCaps(PKSPROPERTY_CROSSBAR_CAPS_S pCaps);
```

The KSPROPERTY_CROSSBAR_CAPS_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      NumberOfInputs;
  ULONG      NumberOfOutputs;
}
  KSPROPERTY_CROSSBAR_CAPS_S,
*PKSPROPERTY_CROSSBAR_CAPS_S;
```

Property: The property header. Does not have to be initialized by the client.

NumberOfInputs: The driver returns the total number of video inputs to the device. Most Osprey-100s have four inputs – three composite, one s-video. The Osprey-2X0 devices have two inputs, one composite, one s-video. The Osprey-540 and -560 have four inputs – composite, s-video, SDI, and DV (1394).

NumberOfOutputs: The driver always returns 1 for this parameter.

The Crossbar PinInfo() function

This function implements the functionality of the standard KSPROPERTY_CROSSBAR_PININFO interface. It is a read-only function. The function prototype is as follows:

```
STDMETHODIMP
IOspreyCrossbar::PinInfo(PKSPROPERTY_CROSSBAR_PININFO_S pPinInfo);
```

The KSPROPERTY_CROSSBAR_PININFO_S structure is as follows:

```
typedef struct
{
  KSPROPERTY     Property;
  KSPIN_DATAFLOW Direction;
  ULONG          Index;
  ULONG          PinType;
  ULONG          RelatedPinIndex;
```

```
  KSPIN_MEDIUM    Medium;
}
  KSPROPERTY_CROSSBAR_PININFO_S,
*PKSPROPERTY_CROSSBAR_PININFO_S;
```

Property: The property header. Does not have to be initialized by the client.

Direction: The client sets this to KSPIN_DATAFLOW_IN or KSPIN_DATAFLOW_OUT. The numbering of pins in Index is separate for input and output pins.

Index: The client sets this to the zero-based index of the input or output pin being queried for.

PinType: The driver returns the type of the pin queried for. For the Osprey-100 and –2X0, the two types of video input pins are KS_PhysConn_Video_Composite and KS_PhysConn_Video_Svideo. For the Osprey-540/560 there are also KS_PhysConn_Video_SerialDigital and KS_PhysConn_Video_1394 inputs.

RelatedPinIndex: The Osprey driver always sets this to ~0 (-1).

Medium.Set: The driver sets this to a GUID for the name of the pin.

Medium.Id: The Osprey driver sets this to an index for the device being accessed. The numbering is a zero-based sequence for the devices controlled by this specific driver.

Medium.Flags: The Osprey driver always sets this to 0.

The Crossbar CanRoute() function

This function implements the functionality of the standard KSPROPERTY_CROSSBAR_CANROUTE interface. It is a read-only function.

The CanRoute() function tells you whether a proposed routing from a specified input pin to a specified output pin is possible. It is almost meaningless with the Osprey video driver, because there is only one output pin to the logical crossbar, and all input pins can connect to it. The driver checks that the input and output pins exist on the current device and sets CanRoute to 1 if they do, 0 if they do not.

The function prototype is as follows:

```
STDMETHODIMP IOspreyCrossbar::CanRoute(PKSPROPERTY_CROSSBAR_ROUTE_S pRoute);
```

The KSPROPERTY_CROSSBAR_ROUTE_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      IndexInputPin;
  ULONG      IndexOutputPin;
  ULONG      CanRoute;
}
  KSPROPERTY_CROSSBAR_ROUTE_S,
*PKSPROPERTY_CROSSBAR_ROUTE_S;
```

Property: The property header. Does not have to be initialized by the client.

IndexInputPin: The client sets this to the zero-based index of the input pin being queried for.

IndexOutputPin: The client sets this to the zero-based index of the output pin being queried for.

CanRoute: The driver returns nonzero if the proposed connection is possible.

The Crossbar GetRoute() function

This read function is strictly speaking an extension of the standard crossbar interface. The standard KSPROPERTY_CROSSBAR_ROUTE interface allows you to set the routing but does not provide a way to find out what routing is currently set in the driver.

The function prototype is as follows:

```
STDMETHODIMP IOspreyCrossbar::GetRoute(PKSPROPERTY_CROSSBAR_ROUTE_S pRoute);
```

The KSPROPERTY_CROSSBAR_ROUTE_S structure is as shown in the CanRoute() description above.

Property: The property header. Does not have to be initialized by the client.

IndexInputPin: The driver sets this to the zero-based index of the current routing.

IndexOutputPin: The driver sets this to 0.

CanRoute: The driver sets this to 1.

The Crossbar SetRoute() function

Thiswrite function implements the functionality of the standard KSPROPERTY_CROSSBAR_ROUTE interface. The function prototype is as follows:

```
STDMETHODIMP IOspreyCrossbar::SetRoute(PKSPROPERTY_CROSSBAR_ROUTE_S pRoute);
```

The KSPROPERTY_CROSSBAR_ROUTE_S structure is as shown in the CanRoute() description above.

Property: The property header. Does not have to be initialized by the client.

IndexInputPin: The client sets this to the zero-based index of the input pin to be routed from.

IndexOutputPin: The client sets this to 0.

CanRoute: If the requested routing is possible, the driver sets this to 1, else to 0.

The Crossbar GetDecoderCaps() function

This function implements the functionality of the standard read-only KSPROPERTY_VIDEODECODER_CAPS interface. The function prototype is as follows:

```
STDMETHODIMP
IOspreyCrossbar::GetDecoderCaps(PKSPROPERTY_VIDEODECODER_CAPS_S pCaps);
```

The KSPROPERTY_VIDEODECODER_CAPS_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      StandardsSupported;
  ULONG      Capabilities;
  ULONG      SettlingTime;
  ULONG      HSyncPerVSync;
}
  KSPROPERTY_VIDEODECODER_CAPS_S,
*PKSPROPERTY_VIDEODECODER_CAPS_S;
```

Property: The property header. Does not have to be initialized by the client.

StandardsSupported: The driver returns the analog video standards that the driver supports, ORed together from the following list that is declared in ksmedia.h:

```
typedef enum
{
    KS_AnalogVideo_None     = 0x00000000,  // This is a digital sensor
    KS_AnalogVideo_NTSC_M   = 0x00000001,  //        75 IRE Setup
    KS_AnalogVideo_NTSC_M_J = 0x00000002,  // Japan,  0 IRE Setup
    KS_AnalogVideo_NTSC_433 = 0x00000004,

    KS_AnalogVideo_PAL_B    = 0x00000010,
    KS_AnalogVideo_PAL_D    = 0x00000020,
    KS_AnalogVideo_PAL_G    = 0x00000040,
    KS_AnalogVideo_PAL_H    = 0x00000080,
    KS_AnalogVideo_PAL_I    = 0x00000100,
    KS_AnalogVideo_PAL_M    = 0x00000200,
    KS_AnalogVideo_PAL_N    = 0x00000400,

    KS_AnalogVideo_PAL_60   = 0x00000800,

    KS_AnalogVideo_SECAM_B  = 0x00001000,
    KS_AnalogVideo_SECAM_D  = 0x00002000,
    KS_AnalogVideo_SECAM_G  = 0x00004000,
    KS_AnalogVideo_SECAM_H  = 0x00008000,
    KS_AnalogVideo_SECAM_K  = 0x00010000,
    KS_AnalogVideo_SECAM_K1 = 0x00020000,
    KS_AnalogVideo_SECAM_L  = 0x00040000,
    KS_AnalogVideo_SECAM_L1 = 0x00080000
}
KS_AnalogVideoStandard;

#define KS_AnalogVideo_NTSC_Mask  0x00000007
#define KS_AnalogVideo_PAL_Mask   0x00000FF0
#define KS_AnalogVideo_SECAM_Mask 0x000FF000
```

The driver does not distinguish between PAL_B, D, G, H, and I. It maps them all to KS_AnalogVideo_PAL_B. If you call SetStandard() with KS_AnalogVideo_PAL_I and then call GetStandard(), KS_AnalogVideo_PAL_B will be returned as the active standard. Similarly, the driver maps all SECAM variants to KS_AnalogVideo_SECAM_B.

Capabilities: The driver returns capabilities the driver supports, ORed together from the following list that is declared in ksmedia.h:

```
typedef enum
{
  KS_VIDEODECODER_FLAGS_CAN_DISABLE_OUTPUT  = 0X0001, //Output can tri-state
```

```
  KS_VIDEODECODER_FLAGS_CAN_USE_VCR_LOCKING = 0X0002, //VCR PLL timings
  KS_VIDEODECODER_FLAGS_CAN_INDICATE_LOCKED = 0X0004, //Can indicate valid
                                                      //signal
}
KS_VIDEODECODER_FLAGS;
```

SettlingTime: The driver returns the settling time for changing input video sources in milliseconds.

HsyncPerVSync: The driver always sets this to 6.

The Crossbar GetDecoderStandard() function

This function implements the functionality of the read portion of the standard KSPROPERTY_VIDEODECODER_STANDARD interface. The function prototype is as follows:

```
STDMETHODIMP
IOspreyCrossbar::GetDecoderStandard(PKSPROPERTY_VIDEODECODER_S pStandard);
```

The KSPROPERTY_VIDEODECODER_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      Value;
}
  KSPROPERTY_VIDEODECODER_S,
*PKSPROPERTY_VIDEODECODER_S;
```

Property: The property header. Does not have to be initialized by the client.

Value: The driver returns the currently set video standard from the KS_AnalogVideoStandard enum shown above in the GetDecoderCaps() description.

The Crossbar SetDecoderStandard() function

This function implements the functionalityof the write portion of the standard KSPROPERTY_VIDEODECODER_STANDARD interface. The function prototype is as follows:

```
STDMETHODIMP
IOspreyCrossbar::SetDecoderStandard(PKSPROPERTY_VIDEODECODER_S pStandard);
```

The KSPROPERTY_VIDEODECODER_S structure is as shown for the previous function.

Property: The property header. Does not have to be initialized by the client.

Value: The client specifes a video standard from the KS_AnalogVideoStandard enum shown above in the GetDecoderCaps() description. If a standard is specified that the driver does not support, no error is returned, but the driver does not change the standard from the current setting.

The Crossbar GetDecoderStatus() function

This function implements the functionality of the standard read-only KSPROPERTY_VIDEODECODER_STATUS interface. The function prototype is as follows:

```
STDMETHODIMP
IOspreyCrossbar::GetDecoderStatus(PKSPROPERTY_VIDEODECODER_STATUS_S
                                  pStatus);
```

The KSPROPERTY_VIDEODECODER_STATUS_S structure is as follows:

```
typedef struct
{
  KSPROPERTY Property;
  ULONG      NumberOfLines;
  ULONG      SignalLocked;
}
  KSPROPERTY_VIDEODECODER_STATUS_S,
*PKSPROPERTY_VIDEODECODER_STATUS_S;
```

Property: The property header. Does not have to be initialized by the client.

NumberOfLines: The driver returns 525 or 625, the number of lines in the currently selected video standard. Note that this value does not indicate the type of signal currently present at the input; you cannot use this value to autodetect the standard of the input signal.

SignalLocked: The driver returns one if the hardware is currently detecting a valid video signal at the selected input; otherwise it returns zero.

The RefSize Interface

The RefSize interface controls video size and placement settings that apply to all pins of the selected device. These include:

- selection of square pixel versus CCIR-601 proportions,
- NTSC video start line,
- NTSC number of video lines.

The RefSize interface includes Get() and Set() functions that both pass the same structure, KSPROPERTY_OSPREY_REFSIZE_S. Prototypes are as follows:

```
STDMETHODIMP IOspreyRefSize::Get(PKSPROPERTY_OSPREY_REFSIZE_S pRefSize);
STDMETHODIMP IOspreyRefSize::Set(PKSPROPERTY_OSPREY_REFSIZE_S pRefSize);
```

Sample usage:

```
CcomPtr  <IBaseFilter   > m_pVidCap;
CComQIPtr<IOspreyRefSize> m_pRefSize;

m_pRefSize = m_pVidCap;

if (m_pRefSize)
{
  KSPROPERTY_OSPREY_REFSIZE_S RefSize;
```

```
  HRESULT hr;

  hr = m_pRefSize->Get(&RefSize);

  if (RefSize.ulCCIR == 0)
  {
    // Enable CCIR-601 reference width
    // (720 instead of square pixel ntsc 640 or pal/secam 768):
    RefSize.ulCCIR = 1;

    hr = m_pRefSize->Set(&RefSize);
  }
}
```

The KSPROPERTY_OSPREY_REFSIZE_S structure is as follows:

```
typedef struct
{
  KSPROPERTY  Property;
  ULONG       ulSSEnabled;    // read-only
  ULONG       ulNtscTopLine;  // read-write
  ULONG       ulNtscLines;    // read-write
  ULONG       ulCCIR;         // read-write
  ULONG       ul625;          // read-only
}
  KSPROPERTY_OSPREY_REFSIZE_S,
*PKSPROPERTY_OSPREY_REFSIZE_S;
```

Property: The property header. Does not have to be initialized by the client.

ulSSEnabled: Read-only. If nonzero, SimulStream is enabled for this device.

ulNtscTopLine: Read/write. Meaningful only when 525-line (NTSC) video standard is selected. May assume the following three values:

- 21: The Closed Caption line, Line 21, is included in the video and is not available for Closed Caption decoding. Attempts to create a CC pin will fail, as will attempts to create a VBI (Vertical Blanking Interval) pin with endline >= 21.
- 22: The Closed Caption line, Line 21, is excluded from the video and is available for decoding. A VBI pin must have endline < 22.
- 23: Line 22 is excluded from the video and can be included in the VBI range by setting the VBI endline to 22. Some broadcast video encodes data on this line.

ulNtscLines: Read/write. May assume just two values:

- 480: This is the normal setting for most applications. ulNtscTopLine can be 21, 22, or 23.
- 485: For specialized applications, all 485 NTSC lines can be captured as video. ulNtscTopLine must be set to 21; Closed Captioning is not available.

ulCCIR: Square / CCIR-601 pixel proportioning indicator. For Osprey-100 and Osprey-2X0, read/write and may equal 0 or 1. For Osprey-540/560, read-only and always reads as 1. When ulCCIR is 0, square pixel horizontal proportioning is set. The reference width is 640 pixels for 525-line video and 768 pixels for 625-line video. When ulCCIR is 1, CCIR-601 pixel proportioning is set, and the reference width is always 720 pixels for both 525-line and 625-line video. The Osprey-540/560 devices always have an unscaled video resolution of 720 pixels per line, although the video can be scaled to any smaller size.

ul625: Read-only. If zero, a 525-line video standard is selected for the device. If one, a 625-line standard is selected. Note that this variable reports the standard selected by software control, not the type of signal present on the video input; the Osprey AVStream driver does not use the CT878 auto standard detect mechanism.

The Crop Interface

The Crop interface controls enabling, sizing and placement of the cropping rectangle. It also allows setting of the default output video size that is visible to clients of the standard video pin properties interface.

Crop settings are maintained on a per-pin basis, and can be set differently for each video pin on each device.

The Crop interface has three functions, Get(), Set(), and GetAlignment().

The Crop Interface – Get() and Set() Functions

The Get() and Set() functions both pass the same structure, KSPROPERTY_OSPREY_REFSIZE_S. Prototypes are as follows:

```
STDMETHODIMP IOspreyCrop::Set(PKSPROPERTY_OSPREY_CROP_S pCrop);
STDMETHODIMP IOspreyCrop::Set(PKSPROPERTY_OSPREY_CROP_S pCrop);
```

Sample usage:

```
CcomPtr  <IBaseFilter> m_pVidCap;
CComQIPtr<IOspreyCrop> m_pCrop;

m_pCrop = m_pVidCap;

if (m_pCrop)
{
  KSPROPERTY_OSPREY_CROP_S CropSettings;
  HRESULT hr;

  // Get current settings for capture pin zero:
  CropSettings.PinId.ucType     = PIN_TYPE_CAPTURE;
  CropSettings.PinId.ucInstance = 0;

  hr = m_pCrop->Get(&CropSettings);

  // Enable cropping with current settings:
  CropSettings.ulCropEnable = 1;

  hr = m_pCrop->Set(&CropSettings);
}
```

The KSPROPERTY_CROP_S structure is as follows:

```
typedef struct
{
  KSPROPERTY  Property;
  PIN_ID      PinId;
```

```
    ULONG       ulRefHt;
    ULONG       ulRefWd;
    ULONG       ulCropTop;
    ULONG       ulCropBottom;
    ULONG       ulCropLeft;
    ULONG       ulCropRight;
    ULONG       ulCropEnable;
    ULONG       ulAutoMode;       // 0==custom 1==full/cif/qcif
    ULONG       ulAutoSize;       // 0==full 1==cif 2==qcif
    ULONG       ulDefaultHeight;  // working default height
    ULONG       ulDefaultWidth;   // working default width
    ULONG       ulDefaultHeightX; // custom  default height
    ULONG       ulDefaultWidthX;  // custom  default width
    ULONG       ulColorFmt;       // color format for offset and granularity
}
  KSPROPERTY_OSPREY_CROP_S,
*PKSPROPERTY_OSPREY_CROP_S;
```

When you call Get() for this property, only one member, PinId, must be filled in.

When you call Set() for this property, all members must be valid except for Property, ulRefHt, and ulRefWd. Normally ulColorFmt should be set to CFMT_INVALID, as explained below.

Property: The property header. Does not have to be initialized by the client.

PinId.ucType: Set by client to either PIN_TYPE_CAPTURE, PIN_TYPE_PREVIEW, or PIN_TYPE_PAIRED. In the _PAIRED case, a Get() call is the same as a call with PIN_TYPE_CAPTURE; and a Set() is the same as a call with PIN_TYPE_CAPTURE followed by a call with PIN_TYPE_PREVIEW.

PinId.ucInstance: Set by client. Always set to 0 if SimulStream is not enabled. Set to any number in the range 0x00..0xFE if SimulStream is enabled.

ulRefHt:
ulRefWd: These are read-only parameters – that is, they are returned by Get() but are ignored by Set(). They define the maximum full frame size that the crop top, bottom, left, and right are based on. They vary according to the current video standard and pixel aspect ratio. For example:

- For standard NTSC square pixel video, the reference size is 640x480.
- For 625-line PAL square pixel video, the reference size is 768x576.
- For NTSC CCIR-601 video, the reference size is 720x480.
- For PAL or SECAM CCIR-601 video, the reference size is 720x576.

For the Osprey-100 and Osprey2X0 devices, all of the above sizes are supported. For the Osprey-540/560, the reference width is always CCIR-601, that is, 720 pixels for both 525-line and 625-line video standards.

ulCropEnable: Zero for cropping disabled, one for cropping enabled. When cropping is disabled, the crop boundaries are not used in defining the active video area; the entire video field is used.

ulCropLeft:
ulCropTop:
ulCropRight:
ulCropBottom: These are the cropping boundaries. They are relative to ulRefHt and ulRefWd. For example, if the crop rectangle is 0, 0, 640, 480, and the reference size is 640x480, then the video is effectively uncropped. If the reference size is 768x576, then a 0, 0, 640, 480 specification excludes the bottom and right edges of the available video from the crop.

It is recommended that when calling Set() you always set these variables to valid settings even when ulCropEnable is 0. The driver will maintain the settings and will recover them when ulCropEnable is set to 1 in the future. The simplest method to maintain these variables correctly is shown in the code sample above – precede each Set() with a Get().

ulDefaultHeight:
ulDefaultWidth: These two variables define the default output size of the video. It is important to understand exactly what the "default output size" is used for in the DirectShow framework. It is not necessarily the actual output size that will appear when the graph is run. Rather, it is simply the size at the top of the size list in the standard pin properties dialog. These variables allow you to provide a customized size choice to the user in this dialog, which otherwise shows only some standard sizes chosen by DirectShow. Your application has the option to not use the default size choice, and to instead call SetFormat() with a video size determined in some other way.

Note that the default output size – as well as the final output size specified by SetFormat() – may not exceed the crop size. For example, if the reference size is 640x480 and the crop rectangle is 0, 0, 320, 240, then the largest output size allowed is 0, 0, 320, 240. An output size of 640x480 would force the driver to upscale video to a size greater than 1X – an operation which it does not support. If you give the driver these setting the graph will run but the video will be uncropped. Note, the statement is that the *driver* will not upscale video. A DirectShow renderer attached to the driver will upscale the video if you ask it to.

The default height and width do not have to be in the same proportions as the reference size or crop rectangle. For example, if the reference size is 640x480, the crop rectangle is 0, 0, 640, 240, and the output size is 320x240, the video will be 1X vertically, but 1/2X horizontally.

ulAutoMode:
ulAutoSize: These two variables are stored by the AVStream driver but are not directly used by it. They are used by clients to control setting of standard sizes. They are used in a particular way by o200avs.ax, which should be adhered to by other clients. That usage is as follows:

If ulAutoMode is 0, then the client is setting ulDefaultHeight and ulDefaultWidth in a freeform manner, to a size that may be nonstandard. In o200avs.ax, the default height and width edit boxes are enabled, and the three radio buttons for standard sizes are disabled.

If ulAutoMode is 1, then the client is using ulAutoSize to set a standard size – 1X, 1/2X, or 1/4X – of the video size defined by ulCropEnable, ulRefHt, ulRefWd, and ulCropTop / Bottom / Left / Right. For example:

- If ulCropEnable is 0 and the reference size is 640x480, then ulAutoSize set to DEFAULT_CIF means that the client is setting the default size to 320x240.
- If ulCropEnable is 1, the reference size is 720x480, and the crop rectangle is 0, 0, 720, 240, then ulAutoSize set to DEFAULT_CIF means the default size is 360x120.

The #defines for the ulAutoSize member are as follows:

```
#define DEFAULT_FULL    0
#define DEFAULT_CIF     1
#define DEFAULT_QCIF    2
```

ulDefaultHeightX:
ulDefaultWidthX: These variables are stored by the driver but are not directly used by it. They are used by clients in conjunction with ulAutoMode and ulAutosize to assist setting of standard sizes. The intended usage is that if ulAutoMode is 0, then the default-X size is the same as the default size; but if ulAutoMode is 1, then the default-X size is whatever custom size was most recently set, while the default size is 1X, 1/2X, or 1/4X of the input video size. In other words, the

default-X values prevent a painstakingly set custom size from getting lost as soon an autosize is selected.

ulColorFmt: This variable is used in conjunction with GetAlignment() function discussed below. This variable is stored by the driver for the convenience of clients but is not directly used by it. Its use is strictly optional. If you do not want to use it, then when calling the Set() function either do not change it from the setting in the most recent Get() call, or set it to CFMT_INVALID. CFMT_INVALID inhibits the driver from storing a value, so that you will not interfere with other clients such as the o200avs.ax property sheets.

The (obscure) function of ulColorFmt is to store the color format setting that video offset and granularity information that a client is using is for. It can be used with the GetAlignment() functions described next to get the offset and granularity requirements for a color format of interest to the user.

The Crop Interface – GetAlignment() Function

This is a read-only function. The prototype is as follows:

```
STDMETHODIMP IOspreyCrop::GetAlignment(
  PKSPROPERTY_OSPREY_CROP_ALIGNMENT_S pCropAlignment
);
```

Sample usage is as follows:

```
//... see previous sample

if (m_pCrop)
{
  KSPROPERTY_OSPREY_CROP_ALIGNMENT_S CropAlignment;
  HRESULT hr;

  // Get yvu9 granularity:
  CropAlignment.ulColorFmt = CFMT_YVU9;

  hr = m_pCrop->GetAlignment(&CropAlignment);

  // Mask down video height and width to meet granularity requirements.
  m_ulHeight &= ~CropAlignment.ulGranV;
  m_ulWidth  &= ~CropAlignment.ulGranH;
}
```

The KSPROPERTY_CROP_ALIGNMENT_S structure is as follows:

```
typedef struct
{
  KSPROPERTY  Property;
  ULONG       ulColorFmt;   // color format for offset and granularity
  ULONG       ulOffsV;      // vertical   crop offset for this color format
  ULONG       ulOffsH;      // horizontal crop offset for this color format
  ULONG       ulGranV;      // vertical   granularity for this color format
  ULONG       ulGranH;      // horizontal granularity for this color format
  ULONG       ulMinV;       // smallest crop and output height allowed
  ULONG       ulMinH;       // smallest crop and output width  allowed}
  KSPROPERTY_OSPREY_CROP_ALIGNMENT_S,
*PKSPROPERTY_OSPREY_CROP_ALIGNMENT_S;
```

Property: The property header. Does not have to be initialized by the client.

ulColorFmt: Initialize to the color format for which offset and granularity information will be returned. The #defines for ulColorFmt are given below.

```
#define CFMT_NONE     0
#define CFMT_RGB32    1
#define CFMT_RGB24    2
#define CFMT_RGB16    3   // not used
#define CFMT_RGB15    4
#define CFMT_RGB8     5   // not used
#define CFMT_GREY8    6   // not used
#define CFMT_YUY2     7   // 4:2:2 packed
#define CFMT_UYVY     8   // 4:2:2 packed
#define CFMT_YVYU     9   // 4:2:2 packed - not used
#define CFMT_BTYUV    10  // 4:1:1 packed - not used
#define CFMT_I420     11
#define CFMT_YVU9     12
#define CFMT_ANY      13
#define CFMT_INVALID  ~0  // write-inhibits ulColorFmt in pCrop->Set()
#define CFMT_NMODES   14
```

If ulColorFmt is set to CFMT_ANY, the driver will return the worst-case granularity. In practice the worst case is YVU9.

ulGranV:
ulGranH: The function returns granularity values for the horizontal and vertical directions for that color format. Crop sizes and default output size parameters must adhere to the granularity requirements expressed in ulGranV and ulGranH.

The values returned are in mask format – that is, the allowed size modulo minus 1. For example, for YVU9 the horizontal granularity modulo is 16, so ulGranH is 15. The code fragment above shows how to use these parameters to mask down a proposed video size to values that are legal for the proposed format.

ulOffsV:
ulOffsH: The offset parameters are not really used – they are always returned as 0 – that is, the top left corner of the video is modulo-1 and can be at any position on the video reference plane. This characteristic is not expected to change for Osprey products based on the CT878-type devices.

ulMinV:
ulMinH: The smallest output and crop size that can be captured. In the 3.1 release the minimum size is 48 wide by 36 high, but this could change for future releases.

The Logo Interface

The Logo interface controls selecting, enabling, sizing, placement, and display mode of logos.

Logo settings are maintained on a per-pin basis, and can be set differently for each video pin on each device.

The Logo interface has two functions, Get() and Set(), that both use the KSPROPERTY_OSPREY_LOGO_S structure. The prototypes are as follows:

```
STDMETHODIMP IOspreyLogo::
Get(PKSPROPERTY_OSPREY_LOGO_S pLogoControl);
STDMETHODIMP IOspreyLogo::
Set(PKSPROPERTY_OSPREY_LOGO_S pLogoControl, BOOL bSaveSpec = TRUE);
STDMETHODIMP IOspreyLogo::
SetPosition(PKSPROPERTY_OSPREY_LOGO_OP_S pLogoOp, BOOL bSaveSpec = TRUE);
```

Sample usage:

```
CcomPtr  <IBaseFilter> m_pVidCap;
CComQIPtr<IOspreyLogo> m_pLogo;

m_pLogo = m_pVidCap;

if (m_pLogo)
{
  KSPROPERTY_OSPREY_LOGO_S LogoSettings;
  HRESULT hr;

  // Get current logo settings for capture pin zero:
  LogoSettings.PinId.ucType     = PIN_TYPE_CAPTURE;
  LogoSettings.PinId.ucInstance = 0;

  hr = m_pLogo->Get(&LogoSettings);

  // Enable logo with current settings:
  LogoSettings.ulFlags |= LOGO_ENABLE;

  hr = m_pLogo->Set(&LogoSettings);

  // . . .
  // Some time later, reposition the logo.
  LogoSettings.ulTop  = 10;
  LogoSettings.ulLeft = 10;
  hr = m_pLogo->SetPosition((PKSPROPERTY_OSPREY_LOGO_OP_S)&LogoSettings);
}
```

The KSPROPERTY_OSPREY_LOGO_S structure is as follows:

```
#define LOGO_MAXPATH 260  // make sure size is consistent user / kernel

typedef struct
{
  KSPROPERTY  Property;
  PIN_ID      PinId;
  ULONG ulOp;               // op code for current command
  ULONG ulFlags;            // flags
  ULONG ulRefId;            // reserved
  ULONG ulLinkId;           // reserved
```

```
  ULONG ulTop;              // logo top  in ulVideoHt units
  ULONG ulLeft;             // logo left in ulvideoWd units
  ULONG ulVideoHt;          // height of video for 1x scaling
  ULONG ulVideoWd;          // width  of video for 1x scaling
  ULONG ulHeight;           // current height of logo
  ULONG ulWidth;            // current width  of logo
  ULONG ulHeight1x;         // height of logo from .bmp file
  ULONG ulWidth1x;          // width  of logo from .bmp file
  ULONG ulKeyColor;         // key color, 0x00rrggbb, working
  ULONG ulKeyColorCustom;   // key color, 0x00rrggbb, custom
  ULONG ulDuration;         // display for this many frames, or 0
  WCHAR awFName[LOGO_MAXPATH];  // path to .bmp file - note WCHARs
}
  KSPROPERTY_OSPREY_LOGO_S,
*PKSPROPERTY_OSPREY_LOGO_S;
```

The KSPROPERTY_OSPREY_LOGO_OP_S structure is as follows:

```
typedef struct
{
  KSPROPERTY  Property;
  PIN_ID      PinId;
  ULONG ulOp;               // op code for current command
  ULONG ulFlags;            // flags
  ULONG ulRefId;            // reserved
  ULONG ulLinkId;           // reserved
  ULONG ulTop;              // logo top  in ulVideoHt units
  ULONG ulLeft;             // logo left in ulvideoWd units
}
  KSPROPERTY_OSPREY_LOGO_OP_S,
*PKSPROPERTY_OSPREY_LOGO_OP_S;
```

Since KSPROPERTY_OSPREY_LOGO_OP_S consists of the first 8 members of the KSPROPERTY_OSPREY_LOGO_S structure, in the same order, you can cast the longer structure to the shorter structure as follows:

```
hr = m_pLogo->SetPosition((PKSPROPERTY_OSPREY_LOGO_OP_S)&LogoSettings);
```

The following member descriptions apply to both structures:

Property: The property header. Does not have to be initialized by the client.

PinId.ucType: Set to either PIN_TYPE_CAPTURE, PIN_TYPE_PREVIEW, or PIN_TYPE_PAIRED. In the _PAIRED case, a Get() call is the same as a call with PIN_TYPE_CAPTURE; and a Set() is the same as a call with PIN_TYPE_CAPTURE followed by a call with PIN_TYPE_PREVIEW.

PinId.ucInstance: Always set to 0 if SimulStream is not enabled. Set to any number in the range 0x00..0xFE if SimulStream is enabled.

ulOp: The op code for the current operation. The client does not have to initialize this member; the Get(), Set(), or SetPosition() function sets it.

ulFlags: The #defines of the ulFlags bits are as follows:

```
#define LOGO_ENABLE     0x01  // display logo
#define LOGO_KEYCOLOR   0x02  // use keycolor
#define LOGO_EMBOSSED   0x04  // embossed style
#define LOGO_SS_EVAL    0x40  // simulstream eval mode in effect (read only)
```

- LOGO_ENABLE: Read/write. When this bit is set, the logo is enabled.

- LOGO_KEYCOLOR: Read/write. When this bit is set, portions of the logo bitmap that match ulKeyColor are not displayed, so that the underlying video is seen instead.

- LOGO_EMBOSSED: Read/write. Enables a transparency or translucency effect. When this bit is set, the logo's RGB color value for each bit is averaged with the RGB color value of the corresponding video bit. "Embossing" is applied after the keycolor is applied.

- LOGO_SS_EVAL: Read only. SimulStream evaluation mode is in effect, and the SimulStream eval logo is being displayed on capture and preview video. The eval logo .BMP file must be used, and LOGO_ENABLE is forced on. The logo can be downsized to half size, no smaller. Transparency ("embossing") and keycolor effects may be used.

ulRefId:
ulLinkId: Not currently used. The client does not have to initialize them before calling Get(), Set(), or SetPosition().

ulTop:
ulLeft: The pixel position on the video of the top left corner of the logo.

ulVideoHt:
ulVideoWd: The pixel width and height of the video frame that ulTop, ulLeft, ulHeight, and ulWidth use as their reference size. ulVideoHt and ulVideoWd may or may not be the video size that is currently running – rather, they are the video size at the time that the logo position and size were most recently set.

ulHeight:
ulWidth: The current working height and width in pixels of the logo, including scaling effects that may have been applied.

ulHeight1x:
ulWidth1x: The unscaled height and width in pixels of the logo bitmap, taken directly from the .BMP file.

ulKeyColor: Read/write. When LOGO_KEYCOLOR in ulFlags is set, portions of the logo bitmap that match ulKeyColor are not displayed, so that the underlying video is seen instead. ulKeyColor is encoded as follows:

```
#define BGR(b,g,r) \
  ((COLORREF)(((BYTE)(r)|((WORD)((BYTE)(g))<<8))|(((DWORD)(BYTE)(b))<<16)))
```

ulKeyColorCustom: Read/write. This parameter is stored by the driver but is not directly used by it. It allows clients to select from a list of standard keycolors while saving a special nonstandard keycolor as well. Encoding is the same as ulkeyColor.

ulDuration: Not currently used.

awFName[]: Read/write. The path and name of a logo file in 24-bit .BMP format. Note that it must be in WCHAR format.

The Logo Interface – Get() Function

```
STDMETHODIMP IOspreyLogo::Get(PKSPROPERTY_OSPREY_LOGO_S pLogoControl);
```

The Get() function returns data saved in the registry for the logo specification of the specified pin. The only member of the structure that needs to be initialized before the call is PinId. If no specification has been defined, the function returns default data in which the LOGO_ENABLE bit is clear and awFName[0] == 0.

The Logo Interface – Set() Function

```
STDMETHODIMP IOspreyLogo::
  Set(PKSPROPERTY_OSPREY_LOGO_S pLogoControl, BOOL bSaveSpec = TRUE);
```

The Set() function sets the complete logo specification for the specified pin. If the specified pin is instantiated and running, changes will be immediately visible on the video. All members of the structure must contain valid data except for Property, ulOp, ulRefId, ulLinkId, and ulDuration.

If ulTop and ulLeft would result in the logo being partially or entirely off the video field as specified in ulVideoHt and ulVideoWd, they will be adjusted so the logo is at the bottom and/or right edge of the video field.

If ulHeight or ulWidth exceeds ulVideoHt or ulVideoWd, the Set() function fails.

The second parameter of Set() is bSaveSpec, which defaults to TRUE. When bSaveSpec is TRUE (or unspecified), your settings are written to the registry. If you specify bSaveSpec FALSE, your settings are not written to the registry and are used only for the currently running pin identified by PinId. If that pin is not instantiated and running, Set() with bSaveSpec FALSE has no effect.

The Logo Interface – SetPosition() Function

```
STDMETHODIMP IOspreyLogo::
  SetPosition(PKSPROPERTY_OSPREY_LOGO_OP_S pLogoOp, BOOL bSaveSpec = TRUE);
```

The SetPosition() function sets position of the logo specified in ulTop and ulLeft. The rest of the logo specification is unchanged. The only members of pLogoOp that must be initialized are PinId, ulTop, and ulLeft.

If ulTop and ulLeft would result in the logo being partially or entirely off the video field as specified in ulVideoHt and ulVideoWd, they will be adjusted so the logo is at the bottom and/or right edge of the video field.

The second parameter of Set() is bSaveSpec, which defaults to TRUE. When bSaveSpec is TRUE (or unspecified), your settings for ulTop and ulLeft (only) are written to the registry. If you specify bSaveSpec FALSE, your settings are not written to the registry and are used only for the currently running pin identified by PinId. If that pin is not instantiated and running, SetPosition() with bSaveSpec FALSE has no effect.

Although Set() can also do a reposition-only operation, you should always use SetPosition() for this purpose. Set() always reloads the logo bitmap from file, and scales and translates it as required. SetPosition() skips these operations and so is much faster.

The Device Configuration Interface

The Device Interface consists of controls and information services that apply to specific video devices. If multiple Osprey capture cards are installed and run under this driver, each card has its own independent device settings.

The functions in this interface are as follows:

- GetConfig()
- SetConfig()
- GetDvcInfo()
- GetRegister()
- SetRegister()
- GetVitc()
- SetVitc()
- GetHDelayAdj()
- SetHDelayAdj()

The Device Configuration Interface – GetConfig() and SetConfig() Functions

Prototypes are as follows:

```
STDMETHODIMP
IOspreyDevice::GetConfig(PKSPROPERTY_OSPREY_DEVICE_CONFIG_S pDeviceConfig);
STDMETHODIMP
IOspreyDevice::SetConfig(PKSPROPERTY_OSPREY_DEVICE_CONFIG_S pDeviceConfig);
```

Sample usage:

```
CcomPtr  <IBaseFilter>   m_pVidCap;
CComQIPtr<IOspreyDevice> m_pDevice;

m_pDevice = m_pVidCap;

if (m_pDevice)
{
  KSPROPERTY_OSPREY_DEVICE_CONFIG_S DeviceSettings;
  HRESULT hr;

  // Get current device settings:
  hr = m_ pDevice->GetConfig(&DeviceSettings);

  // Turn on software deinterlacing.
  DeviceSettings.ulCfg |= DVC_SWDEILACE;

  // Put the revised setting to the driver.
  hr = m_ pDevice->Setconfig(&DeviceSettings);
}
```

The KSPROPERTY_OSPREY_DEVICE_CONFIG_S structure:

```
typedef struct
{
```

```
  KSPROPERTY  Property;
  ULONG       ulCfg;
}
  KSPROPERTY_OSPREY_DEVICE_CONFIG_S,
*PKSPROPERTY_OSPREY_DEVICE_CONFIG_S;
```

Property: The property header. Does not have to be initialized by the client.

ulCfg: The ulCfg bits are as follows. The items marked with '*' are meaningful for the Osprey-500/540/560 only.

```
#define DVC_EVEN_ODD          0x00000001  // rw - even field before odd
#define DVC_SWDEILACE         0x00000002  // rw - sw deilace of capture video
#define DVC_SS_HI_Q           0x00000004  // rw - ss high quality
#define DVC_SS_ENABLED        0x00000008  // ro - ss enabled - full or eval
#define DVC_HWDEILACE         0x00000010  // rw - hw deilace of preview video
#define DVC_SS_MULTI          0x00000020  // rw - multi cap pins per process
#define DVC_SS_KEYED          0x00000040  // ro - full ss is keyed
#define DVC_CC_AVI            0x00000080  // rw - cc timing is avi-compatible

#define DVC_SS_EVAL_NOT       0x00000000  // ro - eval not installed
#define DVC_SS_EVAL_BEFORE    0x00000100  // ro - eval installed, before perd
#define DVC_SS_EVAL_EXPIRED   0x00000200  // ro - eval installed, expired
#define DVC_SS_EVAL_OK        0x00000300  // ro - eval installed, in period
#define DVC_SS_EVAL_USR_ENBL  0x00000400  // rw - eval ss enabled by user
#define DVC_SS_USR_INH         0x00000800  // rw - real ss inhibited by user

#define DVC_PCI_MASK          0x00003000  //      pci compatibility mode mask
#define DVC_PCI_430FX         0x00001000  // rw - pci 430fx   compat. mode
#define DVC_PCI_VIA_SYS       0x00002000  // rw - pci via/sys compat. Mode

#define DVC_540_VBI_ENABLE    0x00004000  // rw - enable 540 vbi pin for wme9

#define DVC_CLR_SDI           0x00010000  // rw - color bypass on SDI port*
#define DVC_CLR_DV            0x00020000  // rw - color bypass on DV port*

#define DVC_SS_TS_WORKAROUND  0x00040000  // (experimental only)
#define DVC_SS_MIN_CPU        0x00080000  // rw – interp scaled copy suppress
#define DVC_500_HWDEILACE     0x00100000  // rw - o500 hardware deinterlace*
#define DVC_PROGRESSIVE       0x00200000  // rw - o5XX progressive scan*
#define DVC_SS_CLIENT_COPY    0x00400000  // (not used)
```

These controls are guaranteed to fully take effect only when all pins on the device are stopped and deleted, and then recreated. That is, the device needs to pass through the state of having zero pins defined by any client application.

- DVC_EVEN_ODD: This control is mainly useful for capturing analog video from digital sources that synthesize progressive video into odd and even interlaced fields. In this case a frame can logically consist of an odd field and the even field following it; or, an even field and the odd field following in. The default is the odd-then-even order; if this bit is set, the order becomes even-then-odd.

- DVC_SWDEILACE: This bit controls software deinterlacing of capture pin video. By default, software deinterlacing is disabled; setting this bit enables it.

- DVC_SS_HI_Q: This bit is meaningful only when SimulStream is installed for the device. Setting this bit tells the driver to assume that multiple video pins are going to be running, and to configure itself accordingly. This mode ensures that there will be no visible glitches ore transitions when subsequent pins are started. It may result in higher cpu usage when only one pin is running, however. When DVC_SS_HI_Q is on, and SimulStream is installed for

the device, software deinterlacing is always turned on, regardless of the settings of the DVC_DEINTERLACE bit.

- DVC_SS_ENABLED: This read-only bit is set if SimulStream is enabled for the device.

- DVC_HWDEILACE: This bit controls hardware deinterlacing of preview pin video. By default, hardware deinterlacing is disabled; setting this bit enables it. More specifically, this control tells the driver to deliver video in VIDEOINFOHEADER2 rather than VIDEOINFOHEADER format. The actual deinterlacing is performed by the the display adapter hardware, if it has that capability.

- DVC_SS_MULTI: With SimulStream installed, by default, although each device can have multiple video capture clients, each individual application is limited to only one video capture pin. The reason is to ensure maximum compatibility with standard applications. If this bit is set, this restriction is removed, and one application can use multiple video capture pins.

- DVC_SS_KEYED: This read-only bit is set if a full license key for SimulStream is installed for the device, as opposed to an evaluation license.

- DVC_CC_AVI: This bit when set puts the CC capture timing in AVI compatibility mode. Refer to the 3.1 driver users' guide, Closed Caption topic, for an explanation of what this control does.

The following four #defines are for four states of two bits of ulCfg. They are read-only.

- DVC_SS_EVAL_NOT: A SimulStream evaluation license is not installed for this device.

- DVC_SS_EVAL_BEFORE: A SimulStream evaluation license is installed for this device, but the start date of the evaluation period has not been reached yet.

- DVC_SS_EVAL_EXPIRED: A SimulStream evaluation license is installed for this device, but the end date of the evaluation period has been passed.

- DVC_SS_EVAL_OK: A SimulStream evaluation license is installed for this device and you are in the evaluation period.

The following SimulStream evaluation bits are read-write:

- DVC_SS_EVAL_USR_ENBL: This bit when set enables SimulStream evaluation mode if the DVC_SS_EVAL_OK condition holds. The main reason for turning the SimulStream evaluation off is to shut off the evaluation logo.

- DVC_SS_USR_INH: This bit when set disables SimulStream when a full SimulStream key is installed. It is independent from DVC_SS_EVAL_USR_ENBL and does not affect evaluation mode.

- DVC_PCI_MASK:
- DVC_PCI_430FX:
- DVC_PCI_VIA_SYS: These bits control the hardware device's PCI bus configuration. When the bits in DVC_PCI_MASK are both 0, the hardware is in "normal" mode. When either DVC_PCI_430FX or DVC_PCI_VIA_SYS is set, the hardware is put in a compatibility mode for respectively 430FX and Via/Sys PCI bridges. Evidence that one of these settings is needed would be inordinant drops of audio and/or video data. Results are undefined if both bits are set at once.

- DVC_540_VBI_ENABLE: This read-write bit applies to the Osprey-540/560 only. The Osprey-540/560 does not support capture of raw VBI data. However, Window Media Encoder 9 will not correctly enumerate the Closed Caption pin that the Osprey-540/560 does provide, unless a VBI pin is exposed for enumeration. When this bit is set the driver exposes a non-functioning but enumeratable VBI pin.

- DVC_CLR_SDI:
- DVC_CLR_DV: These read-write bits apply to the Osprey-500/540/560 only. They control color processing on the SDI and DV (1394) video inputs, respectively. Then they are set, the driver's brightness, contrast, and saturation controls are bypassed and those controls are disabled.

- DVC_SS_MIN_CPU: Read-write, has no effect unless SimulStream is enabled. When set, directs the driver to perform simple rather than interpolated scaled copying of video. Saves CPU time but reduces quality of video.

- DVC_500_HWDEILACE: Read-write. For Osprey-500 only. When set, enables hardware deinterlacing on the Osprey-500 board.

- DVC_PROGRESSIVE: Read-write. For Osprey-500/540/560. If set, incoming video is interpreted as composed of single progressive fields rather than two fields that must be interleaved.

The Device Configuration Interface – GetDvcInfo() Function

The prototype of this read-only function is as follows:

```
STDMETHODIMP
IOspreyDevice::GetDvcInfo(PKSPROPERTY_OSPREY_DVC_INFO_S pDvcInfo);
```

Sample usage:

```
CcomPtr  <IBaseFilter>   m_pVidCap;
CComQIPtr<IOspreyDevice> m_pDevice;

m_pDevice = m_pVidCap;

if (m_pDevice)
{
  KSPROPERTY_OSPREY_DVC_INFO_S DeviceInfo;
  HRESULT hr;

  // Get info for the current device:
  hr = m_ pDevice->GetDvcInfo(&DeviceInfo);
}
```

The KSPROPERTY_OSPREY_DEVICE_CONFIG_S structure:

```
#define DVC_INFO_MAX_NAME  32

typedef struct
{
  KSPROPERTY    Property;       // property header
  KSCOMPONENTID KsComponentId;  // ks component id data
```

```
  ULONG         ulVersion;      // driver version, same as version interface
  ULONG         ulFlags;        // flags
  ULONG         ulProductClass; // 0 = ct878
  ULONG         ulType;         // board type id (o200=0x1A)
  ULONG         ulSerNum;       // board serial number
  ULONG         ulBusNum;       // pci bus number
  ULONG         ulSlotNum;      // pci slot number
  ULONG         ulFunction;     // pci function number (video=0, audio=1)
  ULONG         ulDeviceId;     // device number in awDeviceName
  WCHAR         awDeviceName[DVC_INFO_MAX_NAME];
                                // L"Osprey-230 video device 1", etc
}
  KSPROPERTY_OSPREY_DVC_INFO_S,
*PKSPROPERTY_OSPREY_DVC_INFO_S;

// ulFlags, other bits must be 0:
#define DVC_INFO_SS_KEYED  1
```

Property: The property header. Does not have to be initialized by the client.

KsComponentId: Component id data in KSCOMPONENTID format. Refer to Microsoft DirectShow documentation.

ulProductClass: Included for compatibility with future products. Zero for all Osprey ct878-based products, including the -100, -200,-210, -220, -230, and -540.

ulVersion: A sequence number indicating the driver version. This number as returned by the driver is changed whenever the custom properties interface changes.

ulFlags: One bit of this variable, DVC_INFO_SS_KEYLED, is defined. It is read-only. If this bit is set, it indicates that a valid SimulStream key is currently installed for this device. It is the same as the DVC_SS_KEYED bit returned by the GetConfig() function.

ulType: The board type of the device. The main types supported by the Osprey-MM AVStream driver are as follows:

- o100 = 0x00 – 0x17
- o200 = 0x1A
- o210 = 0x60 – 0x6F
- o220 = 0x70 – 0x7F
- o230 = 0xA0 – 0xAF

The types supported by the Osprey-540/560 driver are as follows:

- o540 = 0x50 – 0x5F
- o560 = 0xB0 – 0xBF

ulSerNum: The serial number of the board in binary format.

ulBusNum: The number of the PCI bus on which the device is installed.

ulSlotNum: The number of the PCI slot on ulBusNum on which the device is installed.

ulFunction: PCI function number, 0 for the video device, 1 for the audio device.

ulDeviceNum: The instance number of this board type, same as the number in awDeviceName.

awDeviceName: A WCHAR string expressing the board type, and instance number of that board type.

The Device Configuration Interface – GetRegister(), SetRegister() Functions

These functions provide direct low-level access to the control registers of the video section of the CT878 device. Write operations may interact in undesirable ways with the driver's access to the CT878 registers, and are therefore at-your-own-risk and unsupported.

Obtain documentation of the Ct878 registers from conexant.com. The document is the "Fusion 878A" data sheet.

Prototypes are as follows:

```
STDMETHODIMP
IOspreyDevice::GetRegister(PKSPROPERTY_OSPREY_REGISTER_S pRegister);
STDMETHODIMP
IOspreyDevice::SetRegister(PKSPROPERTY_OSPREY_REGISTER_S pRegister);
```

Sample usage:

```
CcomPtr  <IBaseFilter>   m_pVidCap;
CComQIPtr<IOspreyDevice> m_pDevice;

m_pDevice = m_pVidCap;

if (m_pDevice)
{
  KSPROPERTY_OSPREY_Register_S Register;
  HRESULT hr;

  // Read the 878 video INT_STAT register:
  Register.ulAddr = 0x100;
  hr = m_ pDevice->GetRegister(&Register);

  // Set contrast to its default.
  // Write lower byte:
  Register.ulAddr = 0x30;  // CONTRAST_LO
  Register.ulData = 0xD8;  // default contrast parameter
  Register.ulMask = 0x00;  // simple write of all bits
  hr = m_ pDevice->SetRegister(&Register);

  // Write upper bit (to both odd and even control registers):
  Register.ulMask = 0x04;  // r-m-w, modify bit 2 only
  Register.ulData = 0x00;  // clear bit 2
  Register.ulAddr = 0x2C;  // E_CONTROL
  hr = m_ pDevice->SetRegister(&Register);
  Register.ulAddr = 0x2C;  // O_CONTROL
  hr = m_ pDevice->SetRegister(&Register);
}
```

The KSPROPERTY_OSPREY_REGISTER_S structure:

```
typedef struct
```

```
{
  KSPROPERTY  Property;
  ULONG       ulAddr;
  ULONG       ulData;
  ULONG       ulMask;
}
  KSPROPERTY_OSPREY_REGISTER_S,
*PKSPROPERTY_OSPREY_REGISTER_S;
```

Property: The property header. Does not have to be initialized by the client.

ulAddr: The address offset of the byte or dword register to be read or written.

ulData:
ulMask:

- For GetRegister(), these members do not have to be initialized. On return, ulData contains the byte or dword data at ulAddr.
- For SetRegister(), both ulData and ulMask must be initialized. If ulMask is 0, a simple byte or dword write is performed rather than a slower read-modify-write. If ulMask is nonzero, a read-modify-write operation is performed that modifies with ulData only the nonzero bits of ulMask.

The Device Configuration Interface – GetVitc(), SetVitc() Functions

These functions control the Osprey custom vertical interval timecode (VITC) interface. Refer to the users' guides for the devices for full details of the VITC implementation. Refer also to the descriptions of TCApp and TCOverlay later on in this document.

Briefly, the interface extracts VITCs from the raw VBI data and watermarks each outgoing video frame with the 8 substantive data bytes encoded in the VITC line. The control functions documented here allow you to turn the interface on or off, specify the field and line to extract VITC from, or alternatively to direct the driver to automatically find and lock onto the VITC line.

The two control functions are as follows:

```
STDMETHODIMP IOspreyDevice::GetVitc(PKSPROPERTY_OSPREY_VITC_S pVitc);
STDMETHODIMP IOspreyDevice::SetVitc(PKSPROPERTY_OSPREY_VITC_S pVitc);
```

Sample code could be as follows:

```
CcomPtr  <IBaseFilter>   m_pVidCap;
CComQIPtr<IOspreyDevice> m_pDevice;

m_pDevice = m_pVidCap;
if (m_pDevice)
{
  KSPROPERTY_OSPREY_VITC_S Vitc;
  HRESULT hr;
  hr = m_pDevice->GetVitc(&Vitc);
  if (SUCCEEDED(hr))
  {
    Vitc.Vitc.ucEnable = 1;  // enable vitc capture
    Vitc.Vitc.ucAuto   = 1;  // enable auto vitc line search
    hr = m_pDevice->SetVitc(&Vitc);
  }
  m_pDevice->Release();
```

```
}
```

The data structures are as follows:

```
typedef union
{
  struct
  {
    UCHAR ucLine;       // rw - the line  requested
    UCHAR ucField;      // rw - the field requested 0 or 1
    UCHAR ucAuto;       // rw - autolock enabled
    UCHAR ucEnabled;    // rw - vitc marking enabled
    UCHAR ucMinLine;    // ro - ntsc 10, pal 6
    UCHAR ucMaxLine;    // ro - ntsc 20, pal 22
    UCHAR ucAutoLine;   // ro - the line      most recently locked to, or 0
    UCHAR ucAutoField;  // ro - the field 1|2 most recently locked to, or 0
  };
  ULONG aul[2];
}
  OSPREY_VITC,
*POSPREY_VITC;

typedef struct
{
  KSPROPERTY  Property;
  OSPREY_VITC Vitc;
}
  KSPROPERTY_OSPREY_VITC_S,
*PKSPROPERTY_OSPREY_VITC_S;
```

If ucEnable is 1, the VITC watermarking is enabled. Since VITC search, extraction, and watermarking all consume significant driver cycles, it should be turned off if it is not being used.

If ucAuto is 1, then the driver will search for the VITC line. If there are multiple VITC lines it will lock on to the first it finds in a search in ascending line order within an indeterminate field order. If this locked-to VITC line drops out, the driver will reenter search mode and attempt to resync with this or another line. ucAuto is not applicable to the Osprey-540/560, which does not implement the search and lock capability. These devices ignore the ucAuto setting, and require that ucLine and ucField be set.

ulLine is the VITC line requested. It can be in the range 10..20 for NTSC or 6..22 for PAL/SECAM. ucField is the field requested, 0 or 1. If ucAuto is specified and the device is an Ospey-100 or -2X0, ucLine and ucField are maintained by the driver for future reference but are not used.

ucMinLine and ucMaxLine are read-only variables that give the allowed range of ucLine for the current video standard. These are the same as the line range of the vertical blanking interval.

ucAutoField and ucAutoLine are currently not implemented and always return 0.

The Device Configuration Interface – GetHDelayAdj(), SetHDelayAdj() Functions

These functions are used to interactively adjust video horizontal delay.

Prototypes are as follows:

```
STDMETHODIMP IOspreyDevice::
GetHDelayAdj(PKSPROPERTY_OSPREY_DEVICE_HDELAYADJ_S pHDelayAdj);
STDMETHODIMP IOspreyDevice::
SetHDelayAdj (PKSPROPERTY_OSPREY_DEVICE_HDELAYADJ_S pHDelayAdj);
```

Sample usage:

```
CcomPtr  <IBaseFilter>   m_pVidCap;
CComQIPtr<IOspreyDevice> m_pDevice;

m_pDevice = m_pVidCap;

if (m_pDevice)
{
  KSPROPERTY_OSPREY_DEVICE_HDELAYADJ_S HDelayAdj;
  HRESULT hr;

  hr = m_ pDevice->GetHDelayAdj(&HDelayAdj);

  if (HdelayAdj.lHDelayAdj < 8)
    HdelayAdj.lHDelayAdj++;

  hr = m_ pDevice->SetHDelayAdj(&HDelayAdj);
}
```

The data structure is as follows. The useful adjustment range is about +/- 8. Current builds of the driver clamp the range to +/- 16, but this number is subject to finalization.

```
typedef struct
{
  KSPROPERTY  Property;
  LONG        lHDelayAdj;
}
  KSPROPERTY_OSPREY_DEVICE_HDELAYADJ_S,
*PKSPROPERTY_OSPREY_DEVICE_HDELAYADJ_S;
```

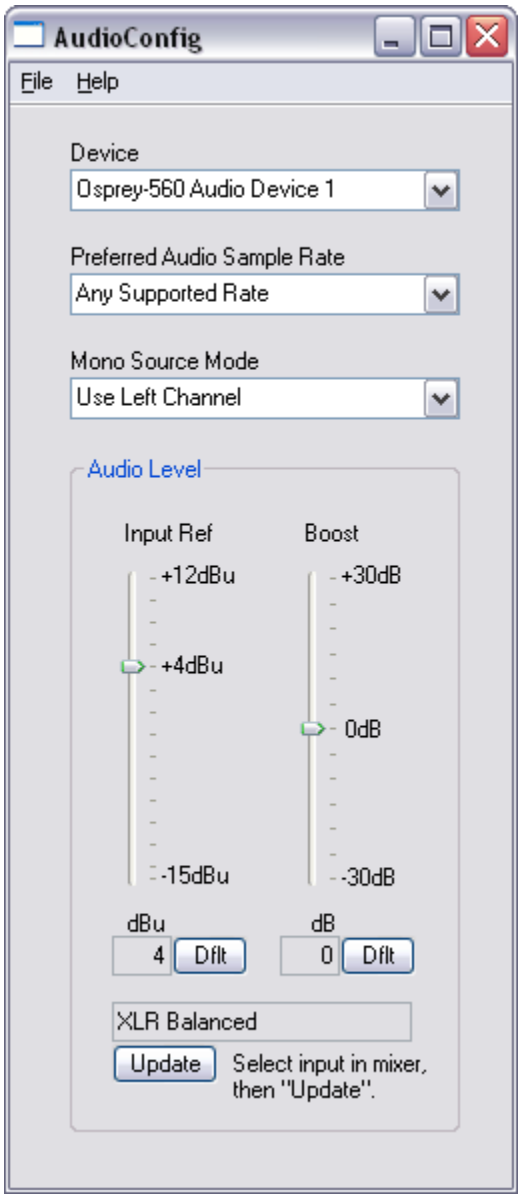
Sample Applications

Four of these samples – VidControl, CropApp, LogoApp, and CCApp – utilize a common class named OspreyCaptureDevice that encapsulates the custom properties as convenient function calls. These four samples also share a common method for enumerating devices, and listing and selecting them.

In addition to the modules described here, the SDK includes the source for SwiftCap, Osprey's AVStream-oriented capture application, and for AudioConfig, a tool for setting up audio. The functionality of these is described in the AVStream driver Users' Guide. The source is provided on an unsupported, as-is basis. SwiftCap utilizes Osprey custom interfaces that are exemplified more simply and clearly in the sample applets. AudioConfig does not utilize any Osprey custom interfaces.

Audio Configuration

AudioConfig provides supplementary audio controls that are not available via the standard system properties. These controls apply only to devices in the Osprey-2XX series. Use the Device menu list at the top of the applet's window to select which device you are controlling.



Preferred Audio Sample Rate

The audio sample rate is the rate at which the hardware samples the incoming audio, which may differ from the sample rate delivered to the client application.

The choice are to allow “Any Supported Rate”, or to force the sampling rate to be 32 kHz, 44.1 kHz, or 48 kHz. If “Any Supported Rate” is selected, all three rates, 32, 44.1, and 48 kHz, are available for selection by the Microsoft kmixer driver. Kmixer, however, does not necessarily select the optimum hardware rate for a given software rate. It may specify a 44.1 kHz hardware rate when a supplying 16 kHz software rate to the application, for example. In this case it would be better to set the Preferred Audio Sample Rate 32 kHz, so that downsampling is exactly 2:1.

When the audio input is SDI, the only sample rate that the Osprey hardware supports is 48 kHz. The driver will override your setting here.

When the audio input is DV1394, you have to use this control to match your audio sample rate to the actual rate of the incoming data. The two most common formats are 48 kHz / 16-bit, and 32 kHz / 12-bit. You may have to listen to a captured sample of your audio to determine whether the sample rate is set correctly – if the pitch is incorrect, try the other setting.

Mono Source Mode

This control determines which audio channel will be the source when monaural audio is selected.

- If set to Use Left Channel, the mono channel contains audio from the left input.
- If set to Use Right Channel, the mono channel contains audio from the right input.
- If set to Average Left and Right, the mono channel contains the average of the two inputs.

Note that in order to get mono audio you have to select mono mode in your application. For example, in SwiftCap you have to select mono 8-bit or mono 16-bit in the Capture Settings -> Audio -> Format control group.

Because of the way DirectShow chooses audio modes, mono mode only works correctly when the sample rate you select is an exact match for an available hardware sample rate. That is, the sample rate must be 32, 44.1, or 48 kHz in most cases, and may be further restricted when SDI or DV1394 is selected (Refer to the explanation of Preferred Audio Sample Rate, above.).

Audio Level

This control sets the hardware Input Reference level and software-based Boost factor. The settings are separate for each input of each device, and are applied to whichever input is selected in the current application or in the system mixer. The settings displayed do NOT automatically update when you change inputs in the application or mixer – click the “update” button to refresh the settings.

The Input Reference level is meaningful only on the analog unbalanced and balanced inputs; when a digital input is selected this control is disabled. The default level is chosen such that the expected amplitude of a full volume input signal will have adequate headroom without clipping. If you do experience clipping, or are working with very low-level signals, you can adjust this level. On this control, a higher reference level results in lower gain, so the quietest setting is at the top of the scale. The units, default value and range are different for unbalanced and balanced signals. Click the “Dflt” button to restore the default value.

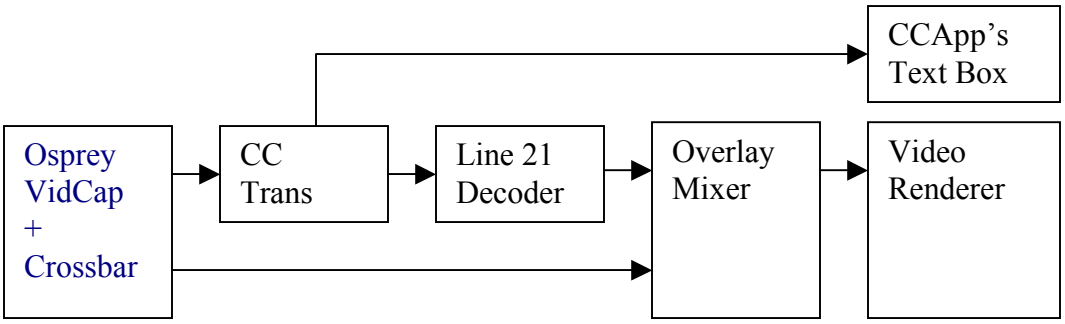
The Boost setting can be set individually for each input. It supplements the system mixer volume controls by providing a very wide adjustment range. You can use it to calibrate or normalize input levels across multiple inputs; or to accomodate microphones or other non-line inputs that have nonstandard signal levels.

CCApp

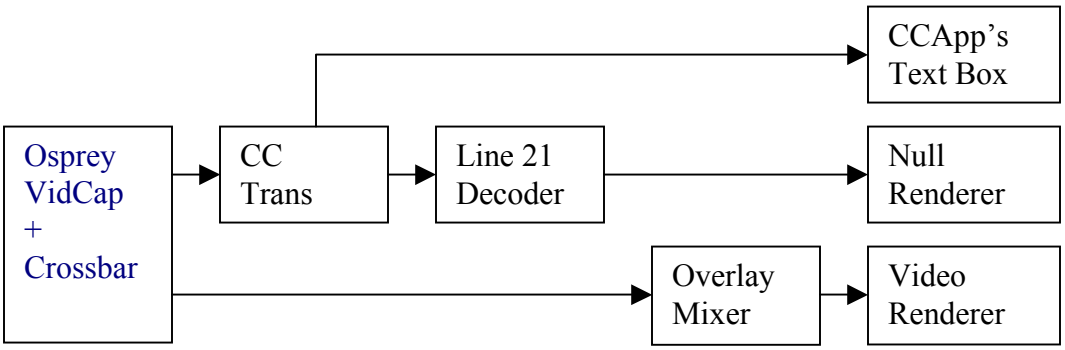
The Osprey AVStream driver performs raw capture of NTSC closed caption character pairs. The character pairs are emitted as a raw, uninterpreted, DirectShow-standard CC Category stream.

CCApp renders Closed Captioning on video, and also displays it as line mode text in a separate text box. The View options permit viewing of no video, video without captions, or video with captions; and allow the text box to be shown or removed.

The graph constructed in CCApp's BuildGraph() routine is illustrated below. The upper pin of "Osprey VidCap" is a standard CC pin that emits raw CC character pairs. "CCTrans" is the CCTransFilter custom filter that is part of the SDK. CCTrans picks off the character pairs and returns them to the application ("Text Box") via a callback, as well as passing them on to the line 21 decoder. "Line 21 Decoder" is a DirectShow filter that decodes CC character pairs and formats them into an overlayable video frame. CCTrans does not transform the datain any way, so Line 21 Decoder can be connected directly to the driver's CC pin if the CCTrans function is not needed. "Overlay Mixer" multiplexes the CC field with preview video from the capture driver, and "Video Renderer" displays the result.



When CC-on-video is turned off by the user, the CC pairs must still have a destination of some sort in the graph, so the graph is modified as follows:



CCApp also illustrates use of IAMAnalogVideoDecoder to obtain the current setting for the video standard, NTSC, PAL, or SECAM. CCApp's CC functions are enabled only for NTSC video.

CCApp renders video on the preview pin. It includes code to handle the "video2" descriptor format used by Osprey preview video.

CCFilter.ax (CCTransFilter)

CCApp uses this filter to obtain Closed Caption character pairs for display in a text box. This filter provides an interface that the application uses to set a callback function that the filter calls with Closed Caption data. The filter also passes the CC data on unchanged to downstream filters. The description of CCApp shows how CCFilter is used in a graph.

In reality, it should be possible to use the generic DirectShow ISampleGrabber filter to perform this and similar tasks. This technique is illustrated for VBI data in the VbiGraph sample shown below. CCFilter is a useful starting point, though, for applications where ISampleGrabber is not adequate.

CropApp

CropApp both illustrates for developers use of the IOspreyCrop interface, and provides a way for both developers and non-developers to set up crops visually and interactively. Its functionality is similar to the driver's Size and Crop property page, but it has the added dimension of graphical placement of the cropping rectangle on live video.

The functions of the left-hand controls are as follows:

- If multiple Osprey devices are in the system, you can select the device of interest from the dropdown list at the top of the control groups. Click "Device Properties…" to access controls that are not explicitly addressed by CropApp.

  All operations affect both the Capture and Preview pin on the device. If SimulStream is enabled, CropApp is hardwired to set up pin pair 0 only – to set up other pins you will have to go to the driver's Size and Crop property page.

- The Reference Image group shows the video height and width that are the reference size for cropping operations. For example, if the reference size is 640x480 and the cropping spec is 640x480, then the video is effectively uncropped. This group also states the basis for this reference size – that is, whether the video standard is NTSC (640x480 or 720x480) or PAL/SECAM (768x576 or 720x576). If CCIR601 sizing is selected, the video width is 720 for NTSC, PAL, and SECAM.

- The Cropping Parameters group is where the current cropping parameters are shown. When the Enable button is not checked, the entire video field is shown, with the crop as an overlayed rectangle. You can modify the crop in three ways:

  - By editing the X, Y, Width, and Height boxes.
  - With the two sets of arrows adjacent to these boxes.
  - By dragging the center, edges, or corners of the crop rectangle on the video.

- The Default Output Size group sets a default size that applications may choose. Use the slider to set the approximate size you want, and then if necessary use the [<] and [>] buttons to fine tune the setting.

  The sizes available in CropApp will always retain a 1:1 height:width proportion. If you want to stretch the video to other proportions, use the driver's Size and Crop property page, or SwiftCap's crop dialog.

  Not all applications use the driver's default output size or present it as a choice; you may have to manually enter the settings calculated by CroppApp into the application.

- The Granularity group allows you to determine the allowed sizing increments for the selected video format. For example, if you select YVU9 in the drop list, you will see that the video widths allowed in this format are modulo-16, that is, 320, 336, 352, etc., and the video heights allowed are modulo-4 – 240, 244, 248, etc. All editing of the crop size will snap to the nearest allowed size.

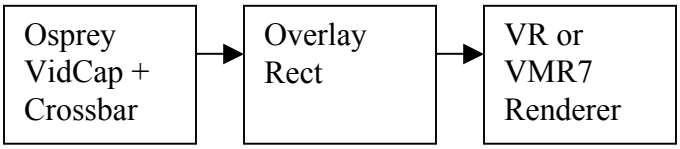
  There are no restrictions on placement of the left and top of the video – for example, in YVU9 the width must be 320, 336, etc, but the left side can be 0, 1, 2, etc.

  Selecting a format here causes CropApp to use that format for its own rendering, but it does not cause that same color format to be selected in your application. In only ensures that your crop size will work with that color format when it is used.

If you exit CropApp with the crop Enabled, that is, the center checkbox in Cropping Parameters checked, the crop parameters will be set in the driver for any other application to use. If you exit CropApp with the crop disabled, the crop parameters will be set for other applications to use, but cropping will not be enabled until it is turned on as a separate step.

Crop uses the custom transform filter OverlayRect.ax to draw the crop rectangle on the video. The source for this filter is under CoverlayRectTransFilter in the SDK source tree.

The graph for interactive mode is as follows:



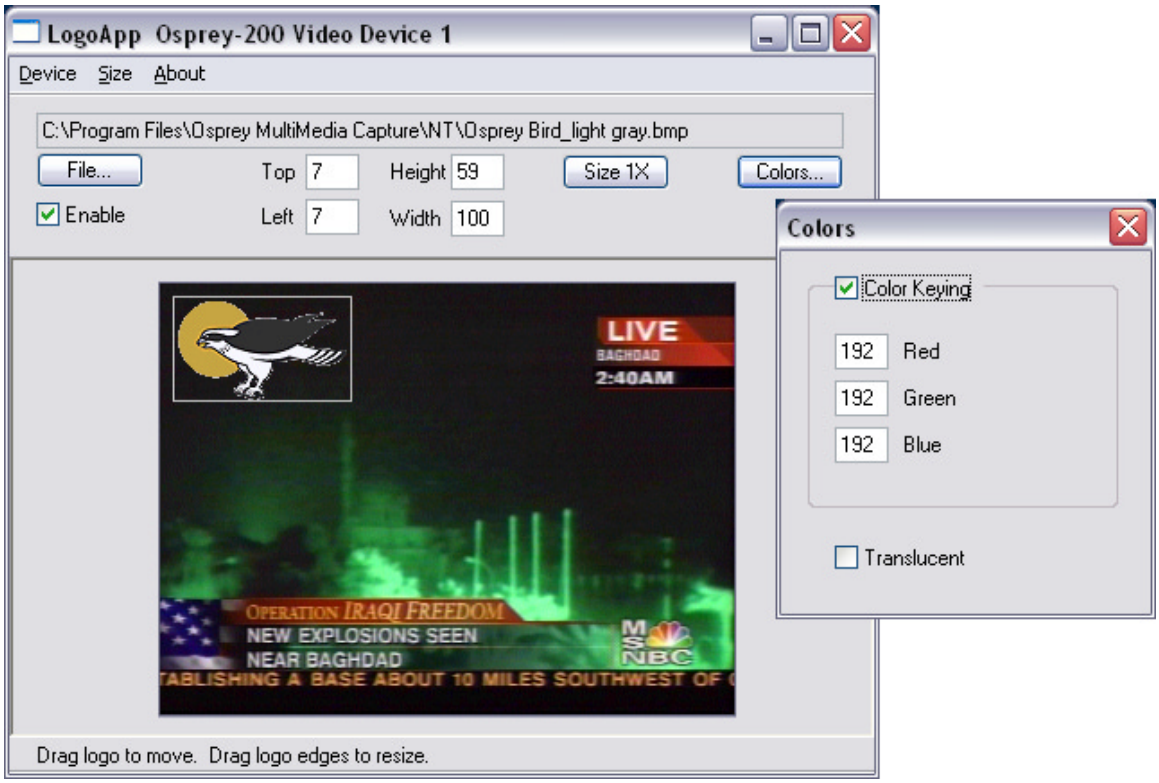
OverlayRect.ax (COverlayRectTransFilter)

This filter is used by CropApp to draw a rectangular outline of the crop region on live video. The code is good starter code for applications that need to draw on or modify video after capture. It

implements an interface named IOverlayRect to set the position of the rectangle and to turn drawing on and off.

One obscure point that it handles correctly is the possible difference between video width and video stride that may happen when the video sink is a Video Mixing Renderer (VMR7 or VMR9). This correction is in place for YUY2 and UYVY formats, the only places where this issue is known to occur.

LogoApp

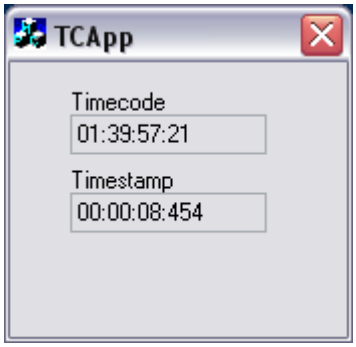


LogoApp demonstrates the IOspreyLogo custom property. The functionality is similar to that of the driver's Logo property page, but you can place and size the logo by dragging its center, sides, or corners directly on live video.

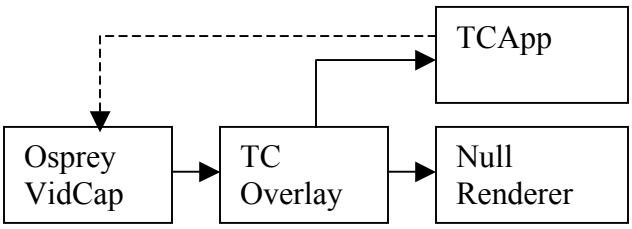
The control bar across the top allows you to Enable/Disable logo display and select the 24-bit .BMP logo file. You can size and place the logo graphically on the video, or you can directly edit the top/left/height/width boxes. Use the Size 1X button to snap the logo to its original size. The Colors… button brings up a dialog to enable color keying, select the key color, and enable translucent mode.

The Size menu allows you to select full or half-size video. The Device menu displays a list of enabled devices and provides access to the selected device's property pages. LogoApp sets the same logo spec for both the Capture and Preview pin. If SimulStream is enabled, LogoApp will set only Pin Pair 0. To set up other pin instances, or to set the capture and preview pins differently, use the driver's Logo property page.

TCApp



TCApp is a demo of Osprey vertical interval timecode (VITC) extraction. The graph is as follows:



The Osprey driver extracts VITC data from the VBI region of the incoming video, and watermarks outgoing video frames with this data. The TCOverlay filter in turn extracts timecodes from the watermarks and returns them along with the timestamps of each frame to TCApp via a custom interface. You could replace the Null Renderer with another video handler, and optionally render timecodes onto the video.

TCApp also includes a control interface to the driver (the dotted line). VITC watermarking has to be turned on in the driver, and because it entails some overhead you will want to keep it turned off unless you are actually using it. TCApp enables watermarking and requests the driver to search for and lock on to the VBI line that contains VITC data. It restores the previous control state at the end of the session. The autosearch feature only works with Osprey-100 and –2X0 devices, not with the Osprey-540/560. For these devices use the driver's Device -> Extras… dialog to manually set the VITC field and line.

TCApp in its present form always connects to the first video capture device that DirectShow enumerates.

TCOverlay.ax

This filter extracts Vertical Interval Timecode (VITC) data that is watermarked into video frames in a proprietary Osprey format. It can deliver the timecode data to applications, and/or can render it onto the video as a text string. When timecode is enabled in the driver, timecodes are "watermarked" onto the low bits of the first 10 words of each video frame. The exact format that is decoded is shown in the Transform() routine in timecodeoverlay.cpp.

To enable timecode watermarking in the driver, go to the driver's Device property page and click the Extras… button. You can also operate this control interface programmatically in a custom app, using the Device Configuration Interface. Use of the interface is illustrated by TCApp. With the Osprey-100 and –2X0 devices you can direct the driver either to search for and lock on to the VITC line, or to look only at a specific field and line that you specify. With the Osprey-540/560 there is no autosearch feature and you have to always set the field and line manually. The controls are per-device, but all capture and preview pins of the device receive the watermarked data.

TCOverlay's application interface consists of three functions plus a callback. The three functions are:

```
STDMETHODIMP SetTCCallback(TCCallbackType pTCCallback, BOOL bDisplay);
STDMETHODIMP GetTCPosn(PBOOL pbDisplay, PULONG pulLine, PULONG pulCol);
STDMETHODIMP SetTCPosn( BOOL  bDisplay,  ULONG  ulLine,  ULONG  ulCol);
```

In SetTCCallback(), pTCCallback is the callback function that is called once each frame. pTCCallback NULL cancels the callback. bDisplay turns rendering of timecodes on the video on or off.

GetTCPosn() and SetTCPos() show and set whether timecodes are rendered, and if so, where on the video. The line variable may be 0, 1, or 2 for top, center, or bottom; the column may be 0, 1, or 2 for left, center or right.

The prototype of the callback function is:

```
void CALLBACK TCCallback(REFERENCE_TIME* pTimeStart, UCHAR* pucVitc);
```

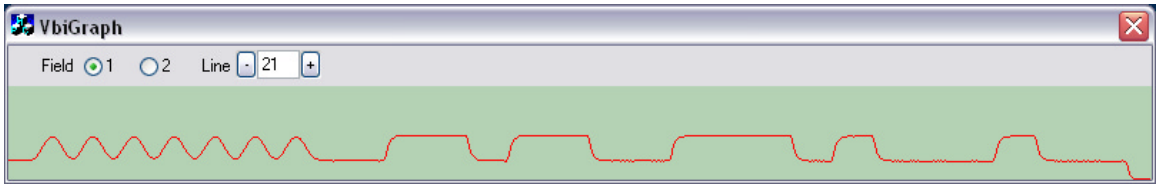
pTimeStart is the LONGLONG timestamp of the watermarked video frame. pucVitc is the 8-byte raw timecode, minus framing and crc bits. The eight actual timecode digits are in the lower bits of the eight bytes; the upper bits contain additional information. The format of the digits is HHMMSSFF, with byte 0 containing the low frame digit and byte 7 containing the upper hour digit. The mask to extract the digits is 3F7F7F3F, in the same order. Refer to the source code of TCApp and/or TCOverlay for more detail on the encoding.

For a full description of all bits of the VITC format, refer to *Timecode: a user's guide – 3rd ed.*, John Ratcliff, Focal Press, 1999.

When the driver is installed, TCOverlay.ax is registered as a WDM Streaming VBI Codec named Osprey Timecode Filter. To use it, use GraphEdit (or a custom application, perhaps based on TCApp, described above); add the timecode filter and attach it to any capture or preview pin of the Osprey AVStream driver; and place a video renderer or other standard capture destination after it. If you are only

To unregister the filter, type on the command line *regsvr32 /u tcoverlay.ax*. To reregister it, *regsvr32 tcoverlay.ax*.

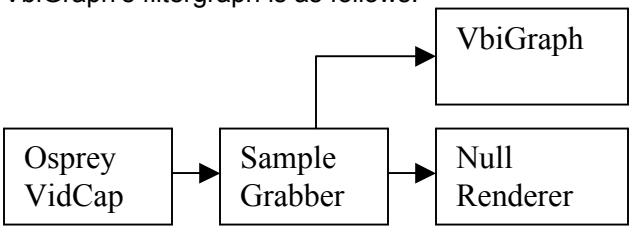
VbiGraph



VbiGraph is a demo of DirectShow VBI streaming. No Osprey custom properties are used. The applet shows how to use the ISampleGrabber interface to capture kernel streaming video samples into user mode space.

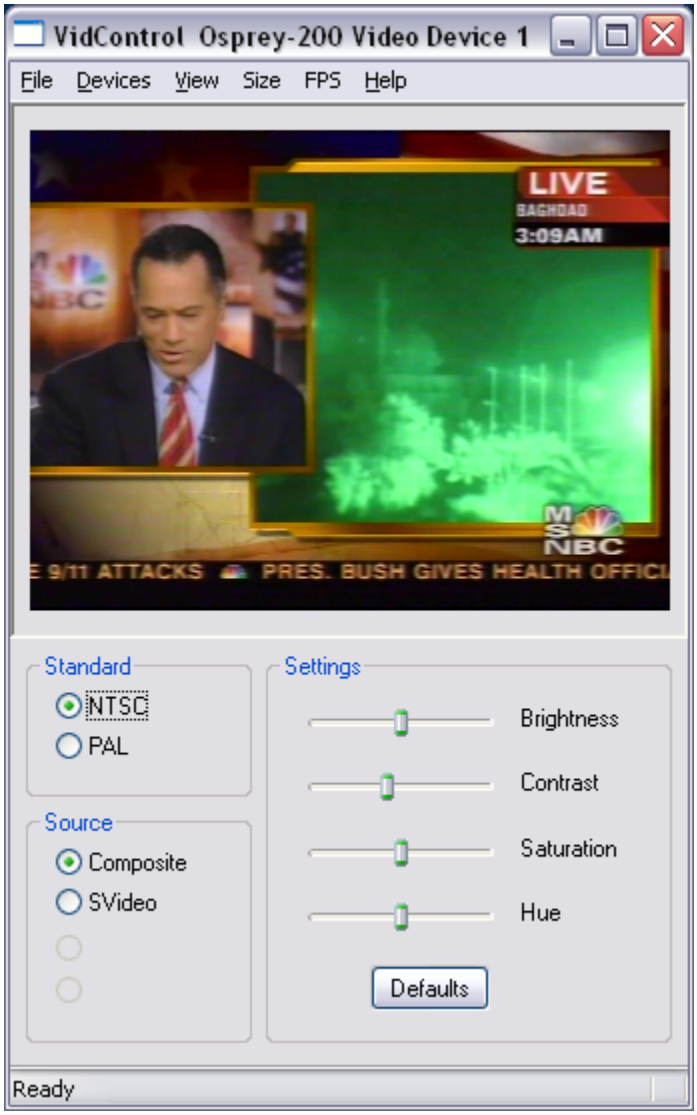
VbiGraph as currently written always connects to the first video capture device enumerated. It will detect whether 525- or 625-line video is currently playing and configure itself accordingly. The controls let you display any VBI line from either video field. Since the Osprey-540 and Osprey-560 do not have VBI pins, VbiGraph does not work with them.

VbiGraph's filtergraph is as follows:



The video capture driver's output pin is of type VBI. The sample grabber is a standard DirectShow filter that delivers VBI bytes to a callback in VbiGraph. The Null Renderer provides a logical destination for the VBI stream.

VidControl



VidControl shows how to control the video source, standard, and settings of an Osprey AVStream Capture Device.

VidControl illustrates:

- the DirectShow IAMVideoProcAmp interface
- the DirectShow IAMAnalogVideoDecoder interface
- the Osprey custom IOspreyCrossbar interface

Two discretionary choices are made in the OspreyCaptureDevice methods that this applet uses:

- The selector for NTSC or PAL video standard uses the DirectShow standard IAMAnalogVideoDecoder interface. It could instead have used the custom IOspreyCrossbar interface. Many applications will prefer the better portability of the IAMAnalogVideoDecoder implementation.

- The selector for video source or input uses the Osprey custom IOspreyCrossbar interface. The other choice would have been to use the DirectShow IAMCrossbar interface. Using IOspreyCrossbar is a bit simpler for applications that are targeted for Osprey devices.

Access the driver's IAMCrossbar interface as follows. Refer to Microsoft's DirectX documentation for details on this interface.

```
// ICaptureGraphBuilder2 pBuilder...
CComPtr<IAMCrossbar> pCrossbar;
hr = pBuilder->FindInterface(
  &LOOK_UPSTREAM_ONLY,
  NULL,
  p_VidCap,
  IID_IAMCrossbar,
  (void**)&pCrossbar
);
```

VidControl renders video on the preview pin. It includes code to handle the “video2” descriptor format used by Osprey preview video.