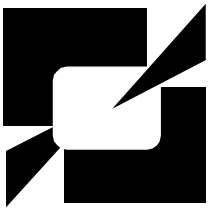
OPI Developer's Guide
Release 2.0

Part No. 40-02025-03



MultiMedia Access Corporation

Osprey Technologies Division

OPI Developer's Guide
Release 2.0

A MultiMedia Access Corporation Company

COPYRIGHT

DISCLAIMER

MMAC reserves the right to change any products herein at any time and without notice. MultiMedia Access Corporation makes no representations or warranties regarding the content of this document, and assumes no responsibility for any errors contained herein.

TRADEMARK ACKNOWLEDGMENT

Osprey-1000, Osprey-100, Osprey-1100, Osprey 1500 and Osprey-150 are trademarks of MultiMedia Access Corporation. Microsoft, Windows NT, Windows 95, Windows 98 and Video for Windows are trademarks or registered trademarks of Microsoft Corporation. Solaris, XIL, and SunVideo Plus are trademarks of Sun Microsystems, Inc. Digital UNIX is a trademark of Compaq Computer Corporation. Any other product names, trademarks, trade names, service marks, or service names owned or registered by any other company and mentioned herein are the property of their respective companies.

MultiMedia Access Corporation
Osprey Technologies Division
600 Airport Boulevard Suite 900
Morrisville North Carolina 27560

Table of Contents

Chapter 1 – Introduction 9

Introduction 9

Chapter 2 – OPI Class Overview 11

Overview 11
OPISystem 11
OPIPipeline 11
OPIStage 11
OPIDevice 11
OPIDeviceInstance 12
Constructing a pipeline 12
Pipeline operations 13

Chapter 3 – Basic OPI Example Program 15

Example 15
Including the header file 16
Initializing the OPISystem 16
Creating the OPIPipeline 17
Setting up the formats for the various stages of the pipeline. 17
Creating the stages of the pipeline 18
Mapping the pipeline 19
Finishing the construction of the pipeline 20
Starting the pipeline 20
Delete the OPIPipeline and OPISystem 20
Source Code Listing: example.cpp 21

Chapter 4 – OPI Streams and Polling 25

Overview 25
OPIStream 26
Creating and setting OPIStreams 27
Using polling to communicate with an OPIPipeline 27

Chapter 5 – OPIWindow and Callbacks 29

OPIWindow 29
Constructing an OPIWindow 29
Callbacks 30
Setting callbacks 30

Priming the pipeline 31
Callback function 31

Chapter 6 – More Details on OPI Concepts 33

OPI Video and Audio Formats 33
OPI Video Formats 34
OPI Audio Formats 36
Input & Output Stream Capabilities 37

Chapter 7 – Hardware Specific Capabilities 39

The Osprey-1x0 family 39
The Osprey-1x00 family 40
Audio and Video hints 42
Video Attributes 43
Codec 43

Chapter 8 - Attribute Control 45

Setting attributes on a known named device 45
Control Panel Example 45
Obtaining the instances of the device 46
Obtaining the attributes for a device instance. 46
Attrlist Example 47
Listing the instances for each device 48
Listing the attributes for a device instance 48

Chapter 9 – OPI Developer's Kit Examples 49

Record Example 49
Setting up the formats for the various stages of the pipeline 50
Creating the branching stages of the pipeline 51
Setting the compression and capture attributes of the device 52
Finishing the construction of the pipeline 52
Loopback Example 53
Getting device specific instance names 54
Forcing a pipeline to use a particular device 54
Setting device specific parameters 55

Appendix A – Class Reference 57

Introduction 57
OPIAttribute 59
Constructor 59
OPIAttributeInfo 60
Constructor 60
Member Functions 61

OPIBuffer 63
Constructor 63
Destructor 63
Member Variables 64
Member Functions 64
OPICaptureDeviceAudio 65
Member Functions 65
OPICaptureDeviceVideo 67
Member Functions 67
OPIColorcube 69
Constructor 70
Destructor 70
Member Functions 70
Static Member Functions: 73
OPICompressionDeviceAudio 74
Member Functions 74
OPICompressionDeviceH261 75
Member Functions 75
OPICompressionDeviceVideo 77
Member Functions 77
OPIDecompressionDeviceAudio 79
Member Functions 79
OPIDevice 80
Constructor 80
Destructor 80
Member Variables 81
Member Functions 81
OPIDeviceInstance 83
Constructor 83
Destructor 83
Member Variables 84
Member Functions 84
OPIFormat 86
Member Variables 86
Member Functions 87
OPIFormatAudio 88
Constructor 88
Member Functions 88
OPIFormatVideo 89
Constructor 89
Member Variables 89
Member Functions: 90

OPINameList 91
Constructor 91
Member Functions 91
OPIOutputDeviceAudio 93
Member Functions 93
OPIPipeline 95
Constructor 96
Destructor 96
Member Functions 96
OPIStream 100
Constructor 100
Destructor 101
Member Functions 101
OPISystem 104
Constructor 104
Destructor 104
Member Variables 105
Member Functions 105
OPIWindow 107
Constructor 107
Destructor 108
Member Functions 108

Appendix B – C Interface 111

OPI C Types 111
OPI C Functions 114

Appendix C – OS Specific Details 125

Introduction 125
Windows 126
Installation 126
Include Files 126
Link Directives 126
OPI Dynamically Loadable Libraries 126
Installed Devices 126
Solaris 127
Installation 127
Include Files 127
Link Directives 127
OPI Dynamically Loadable Libraries 127
Installed Devices 127
Compaq's Digital UNIX 128

Installation 128
Include Files 128
Link Directives 128
OPI Dynamically Loadable Libraries 128
Installed Devices 129

Chapter 1 -Introduction

Introduction

OPI is an application-programming interface for cross-platform multimedia programming using the Osprey multimedia cards. The OPI API provides a mechanism for accessing card specific capabilities in a platform independent way, and serves as the framework for producing platform and hardware independent applications.

OPI is not the only interface for programming the Osprey multimedia cards under a single OS platform. Developers who are implementing applications under a single OS platform are strongly encouraged to use the supported video and audio standard interfaces such as Video for Windows under the Windows platform or XIL under the Solaris platform. These interfaces will provide better portability to other vendor's solutions and are currently more technically matured environments. However, OPI may be the solution for cross-platform multimedia development or the solution for any card functionality that is not supported by the standard interfaces.

The OPI API is a collection of C++ classes. OPI represents a device by an instance of a generic class with derived classes providing for specific capabilities of devices. OPI provides the ability to query devices about the capabilities they provide. Through OPI, programmers conceptually define how to process audio and video data. OPI will map the conceptual view to actual hardware and software modules. The OPI class hierarchy diagram is illustrated in Figure 1-1.

OPI development is ongoing. The Osprey 1x00 and the Osprey 1x0 families of cards are currently supported under Win32 (Windows 95, Windows NT and Windows 98), Solaris 2.5 and Solaris 2.6, and Compaq's Digital UNIX. A C interface to OPI has been added into our current release *(See Appendix B).* Future releases of OPI may include interfaces for other programming languages such as Java. Osprey will make every effort to ensure that future releases of OPI are binary compatible with the current version. However, future releases may require a recompilation of your OPI code to support updated features.

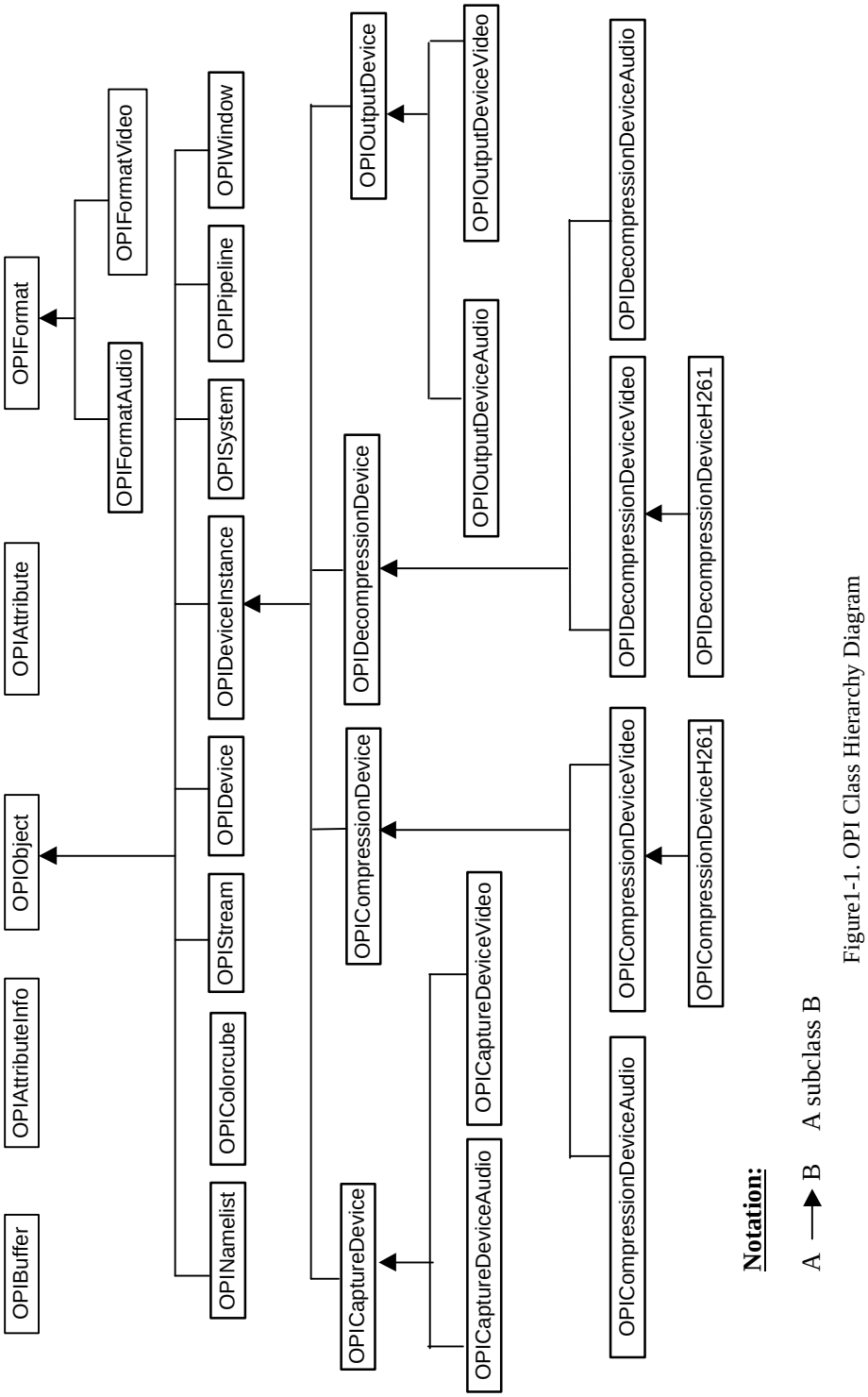


Figure1-1. OPI Class Hierarchy Diagram

Chapter 2 - OPI Class Overview

Overview

In Chapter 3, we will illustrate basic OPI concepts using a simple video capture/display example. This chapter introduces those concepts. For in-depth reference, consult the class reference guide in Appendix A.

OPISystem

Creating an *OPISystem* object initializes OPI. There may be a limitation of a single *OPISystem* per process. Creation of the *OPISystem* determines which devices are present and supported by OPI in this machine.

OPIPipeline

An *OPIPipeline* defines the conceptual processing of data. A user creates a pipeline by defining a series of stages through which the data passes. There may be multiple active pipelines within the *OPISystem*.

OPIStage

An *OPIStage* performs a single operation on the data. It has an input format, an output format, and a capability. The capability defines whether the stage is capturing, compressing, decompressing, or displaying the input data. After mapping the pipeline (defined later), each *OPIStage* also contains a pointer to the *OPIDeviceInstance* that handles this operation. Multiple stages may be executed by a single device that handles the combined operation.

OPIDevice

The *OPIDevice* class represents the capabilities of a device type. There is only one instance of *OPIDevice* per type of device installed on the system.

OPIDeviceInstance

The *OPIDeviceInstance* class provides the interface to each individual device. If there is more than one device of a single type installed on the system, there will be an *OPIDeviceInstance* for each one. For example, if there are two Osprey-1000 cards in a system, there will be one *OPIDevice* object representing the Osprey-1000 device type and two *OPIDeviceInstance* objects representing the two Osprey-1000 cards.

Constructing a pipeline

The construction of a pipeline consists of four basic steps:

1. Define each stage and the relationship between the stages. An input format, an output format, and the type of action define each stage. The relationship between stages is defined by specifying the first stage in the pipeline through the ***firstStage*** routine and then adding stages to the pipeline with the ***addStage*** routine.

2. Map the pipeline to devices by calling the ***mapToDevices*** routine. The pipeline defines the *conceptual* flow of data through the system. The actual data flow may differ, depending on the capabilities of the installed devices. For instance, if a capture is followed by a display, then a device that supports both capabilities will handle both stages (*See Figure 2-1 An OPIPipeline mapping to a single OPIDeviceInstance).* With the same pipeline, a simpler capture device will handle just the capture, and a software device will handle display.

3. Optionally, set up the explicit buffering and set any specific device attributes.

4. Complete the pipeline construction by calling the ***finishConstruction*** routine.

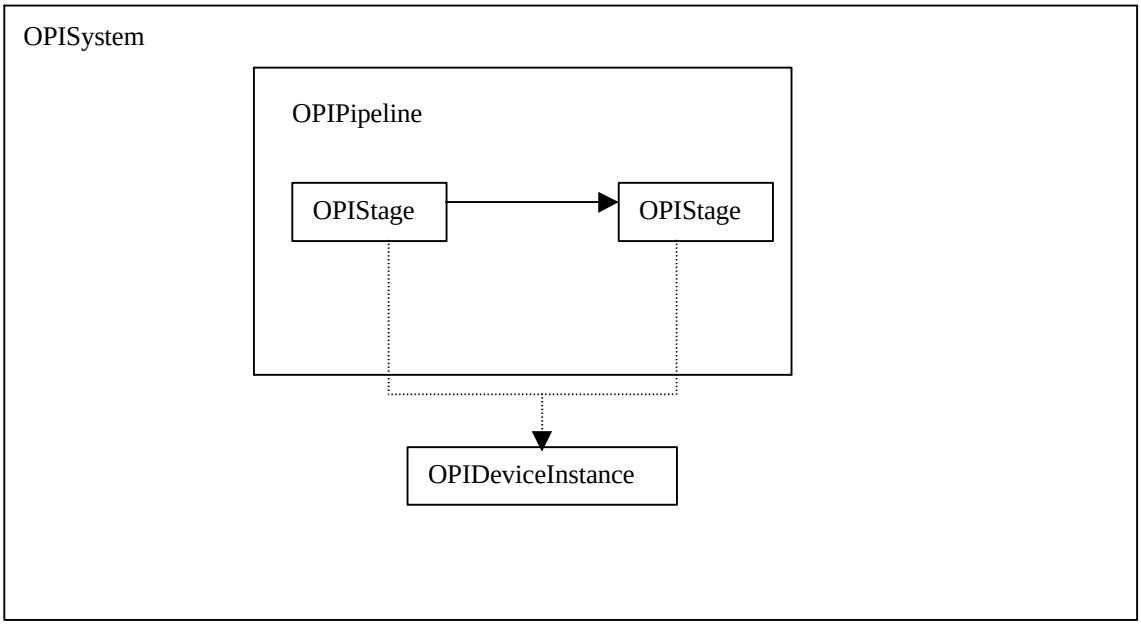


Figure 2-1 An OPIPipeline mapping to a single OPIDeviceInstance

Pipeline operations

Once the pipeline is constructed, one of four operations will act on the pipeline. These operations are ***start***, ***pause***, ***resume*** and ***stop***.

Chapter 3 – Basic OPI Example Program

To introduce you to programming with the OPI library, this chapter will use a basic OPI program called **example** as an illustration. This program demonstrates the use of a pipeline to perform a raw video capture and display. The source code for this program – **example.cpp,** appears at the end of this chapter. Note that this example program will run on an Osprey-1x00 or an Osprey-1x0 card if an appropriate video format is given. For more details on the various video formats supported by the Osprey-1x00 or the Osprey-1x0 families, please refer to *Table 6-1, Video Formats*. For illustration purposes, this example defaults to the output format "RGB565" which is supported by both families. Chapters 4 and 5 discuss detailed examples, and the OPI Developer's Kit contains additional programming examples. This example is installed in the ***examples/rawdisplay*** directory of the ODK installation.

Example

This chapter walks you through the program's code, explaining the steps involved which will be part of most OPI programs. The topics covered are:

- Including the header file.
- Initializing the *OPISystem*.
- Creating the *OPIPipeline*.
- Setting up the format variables for the various stages of the pipeline.
- Creating the stages of the pipeline.
- Mapping the pipeline.
- Finishing the construction of the pipeline.
- Starting the pipeline.
- Deleting the *OPIPipeline* and the *OPISystem*.

Figure 3-1 illustrates the pipeline used in **example**.

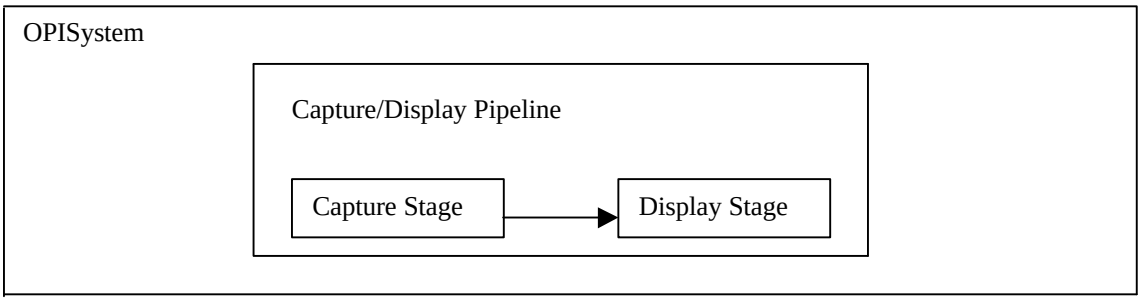


Figure 3-1 Graphical illustration of example

Including the header file

In **example**, you will see a preprocessor directive that includes the header file **opi.h**. All OPI programs must include this header file and, thus, must include the line

```
#include "opi.h"
```

Initializing the OPISystem

All OPI programs must call ***new OPISystem*** () to initialize the library before using other functions from the OPI library. Note that the entire OPI system creation is bracketed within a try and catch block. The *OPIInternalError* exception can be raised if there is something wrong with the OPI system installation (i.e. registry or configuration files are missing or corrupted).

The ***setDebugLevel*** routine sets debugging parameters for the OPI system. For now, the only valid value is 1, which disables certain buffer timeouts. If you are going to be doing any debugging of your program under an interactive debugger, you should call this routine.

```
// Initializing the OPI System
OPISystem* opiSys = NULL;
try
{
      opiSys = new OPISystem();
}
catch(OPIInternalError)
{
      cerr << "An OPI Internal Error has occurred.\n";
      return APP_FAIL;
}
opiSys->setDebugLevel(1);
```

Creating the OPIPipeline

The creation of the OPI pipeline serves as a basic step before the creation of the different operation stages of the pipeline. A pipeline provides the basic construct to allow the user to tell the API what should be done.

```
// Creating the OPI pipeline
OPIPipeline* capturePipeline = NULL;
if((capturePipeline = new OPIPipeline (opiSys)) == NULL)
 {
   cerr << "Could not create pipeline\n";
   delete opiSys;
   return APP_FAIL;
 }
```

Setting up the formats for the various stages of the pipeline.

Each stage is defined by its input format, output format, and capability. The *OPIFormat* class is a base class for *OPIFormatVideo* and *OPIFormatAudio*. While many of the members have default values, the *compression* member must always be set. The *compression* member defines how the rest of the members are interpreted even for uncompressed formats.

In this example, the input format of the capture stage (*externalFmt)* is **"NTSC."** The default output of the capture stage and input of the display stage *(captureFmt)* is **"RGB565."** For uncompressed formats, it is also necessary to specify the width and height. The output of the display stage *(windowFmt)* is set to **"WINDOW"** to indicate that the output will be displayed on a window on the workstation.

The OPI video and audio formats will be discussed in detail in Chapter 6 in the section *OPI Video and Audio Formats* on page 33. To run this program on an Osprey-1x00 or Osprey-1x0 card, simply set the command line video_format parameter to a supported video format or use the default value. Please refer to *Table 6-1, Video Formats* for a list of Osprey1x00 or Osprey1x0 supported video formats.

```
// Set up the input format variables for the first stage of
// the pipeline.
OPIFormatVideo externalFmt;
externalFmt.compression = "NTSC";

// Set up the output format variables for the first stage of
// the pipeline.
OPIFormatVideo captureFmt;
captureFmt.width  = WIDTH;
captureFmt.height = HEIGHT;
captureFmt.depth  = DEPTH;
captureFmt.compression = video_format;

// Set up the window format variables for the window stage of
// the pipeline.
OPIFormatVideo windowFmt;
windowFmt.compression    = "WINDOW";
windowFmt.externalOutput = FALSE;
windowFmt.width  = WIDTH;
windowFmt.height = HEIGHT;
windowFmt.depth  = DEPTH;
```

Creating the stages of the pipeline

We create the first stage of the pipeline by calling the ***firstStage*** routine. We describe this stage by supplying the capability, input and output stream video formats. In this case, the capability is video capture (CAP_VID_CAPTURE). ***firstStage*** returns the created stage. We now add a display stage to the pipeline by calling ***addStage***. The ***addStage*** routine takes the same parameters as ***firstStage*** plus the stage to which we are adding this stage. For the display stage, the capability is CAP_VID_DISPLAY.

```
// Create the capture stage.
OPIStage* captureStage;
captureStage = capturePipeline->firstStage(CAP_VID_CAPTURE,
                                           &externalFmt,
                                           &captureFmt);

// Create the display stage.
OPIStage* displayStage;
displayStage = capturePipeline->addStage(captureStage,
                                         CAP_VID_OUTPUT,
                                         &captureFmt,
                                         &windowFmt);
```

Mapping the pipeline

Once the pipeline stages are created, OPI has to assign the actual hardware and/or software to each of the operations in the pipeline stages. Each stage may have a different *OPIDeviceInstance* mapped to it, or multiple stages may be assigned the same *OPIDeviceInstance*.

```
// Mapping the pipeline.
if(capturePipeline->mapToDevices() == FALSE)
{
  cerr << "Could not map to pipeline.\n";
  delete capturePipeline;
  delete opiSys;
  return APP_FAIL;
}
```

Finishing the construction of the pipeline

The ***finishConstruction*** routine is called to indicate to OPI that we're done constructing the pipeline. At this point, OPI will prepare the devices to start.

```
// Finish the pipeline construction.
   capturePipeline->finishConstruction();
```

Starting the pipeline

When you call the ***start*** routine, the pipeline will begin to run autonomously.

```
// Start the pipeline
   capturePipeline->start();
```

Delete the OPIPipeline and OPISystem

Before exiting, all OPI programs should destroy all OPI objects or data structures that were created explicitly by a *new* operation. It is recommended to stop all pipelines before deleting the pipeline object. Deleting a pipeline automatically deletes all the stages of the pipeline.

```
// Delete the OPI pipeline, OPI system and OPI window.
  capturePipeline->stop();
  delete capturePipeline;
  delete opiSys;
```

Source Code Listing: example.cpp

```
/* $Id: example.cpp 1.6 1998/07/10 18:48:33 brettb Exp $ */
/*****************************************************************
 ** File: File Long Name ($RCSfile: example.cpp $)
 **
 ** ($Revision: 1.6 $)
 **
 ** Abstract:
 **   This example program demonstrates video display.
 **
 **   A capture pipeline is created and is then started.
 **   The pipeline captures video and displays this video
 **   in a window.
 **
 **   To execute this program, please enter: example [option]
 **   Legal Option   |   Description   |  Defaults
 **   ------------------------------------------------------
 **       -c         |  video format   |  RGB565
 **
 ** Copyright (c) 1998, MultiMedia Access Corp
 **                     Osprey Tech Division
 *****************************************************************/

#include <iostream.h>
#include <string.h>
#include "opi.h"

#define APP_OK      0
#define APP_FAIL   -1
#define WIDTH     320
#define HEIGHT    240
#define DEPTH      16


/*****************************************************************
 ** Function: Raw Display (main)
 **
 ** Abstract:
 **   Initialize OPI, and display video to a window.
 **
 ** Error Conditions: None
 **
 *****************************************************************/
int main(int argc, char** argv)
{
  int nframes = 100;
  char* video_format = "RGB565";

  if (argc >= 2) {
```

```
    if (strcmp( argv[1], "-c") == 0)
      video_format = strdup( argv[2]);
    else {
      cout << "Please enter rawdisplay -c [video format] "<< endl;
      return APP_FAIL;
    }
  }

  // Initializing the OPI System
  OPISystem* opiSys = NULL;
  try
  {
    opiSys = new OPISystem();
    opiSys->setDebugLevel( 1);
  }
  catch( OPIInternalError)
  {
    cerr << "An OPI Internal Error has occurred.\n";
    return APP_FAIL;
  }

  // Creating the OPI pipeline
  OPIPipeline* capturePipeline = NULL;
  if((capturePipeline = new OPIPipeline( opiSys)) == NULL)
  {
    cerr << "Could not create pipeline\n";
    delete capturePipeline;
    delete opiSys;
    return APP_FAIL;
  }

  // Set up the input format variables for the first stage of
  // the pipeline.
  OPIFormatVideo externalFmt;
  externalFmt.compression = "NTSC";

  // Set up the output format variables for the first stage of
  // the pipeline.
  OPIFormatVideo captureFmt;
  captureFmt.width  = WIDTH;
  captureFmt.height = HEIGHT;
  captureFmt.depth  = DEPTH;
  captureFmt.compression = video_format;

  // Set up the window format variables for the window stage of
  // the pipeline.
  OPIFormatVideo windowFmt;
  windowFmt.compression    = "WINDOW";
  windowFmt.externalOutput = FALSE;
  windowFmt.width  = WIDTH;
  windowFmt.height = HEIGHT;
```

```
  windowFmt.depth  = DEPTH;

  // Create the capture stage.
  OPIStage* captureStage;
  captureStage = capturePipeline->firstStage(CAP_VID_CAPTURE,
                                             &externalFmt,
                                             &captureFmt);

  // Create the display stage.
  OPIStage* displayStage;
  displayStage = capturePipeline->addStage(captureStage,
                                           CAP_VID_OUTPUT,
                                           &captureFmt,
                                           &windowFmt);

  // Mapping the pipeline.
  if(capturePipeline->mapToDevices() == FALSE)
  {
    cerr << "Could not map to pipeline.\n";
    delete capturePipeline;
    delete opiSys;
    return APP_FAIL;
  }

  // Finish the pipeline construction.
  capturePipeline->finishConstruction();

  // Start the pipeline
  capturePipeline->start();

  // Putting a delay immediately after starting the frame capture
  // to let the user view the captured frames on the display.
  // This is just a simple example. In an actual application,
  // the developer may implement a "QUIT" Button here to wait
  // for a user's response to terminate the application.
  OPISleep(33*nframes);

  // Delete the OPI pipeline, OPI system and OPI window.
  capturePipeline->stop();
  delete capturePipeline;
  delete opiSys;
  return APP_OK;
}
```

Chapter 4 - OPI Streams and Polling

Overview

To introduce more OPI concepts, this chapter will use an OPI program called **vidcapdsp** as an illustration. This program demonstrates the use of two pipelines to perform a raw video capture, save the data to a file, and display. The program will also demonstrate how to share buffers between pipelines. This example program will run on an Osprey-1x00 or an Osprey-1x0 card if the appropriate video format is given. For more details on the various video formats supported by the Osprey-1x00 or the Osprey-1x0 families, please refer to Chapter 6, Table 6-1, Video Formats). For illustration purposes, this example uses the output format "RGB565" which is supported by both families. This example is in the ***examples/vidcapdsp*** directory of the ODK installation.

The program is graphically illustrated below:

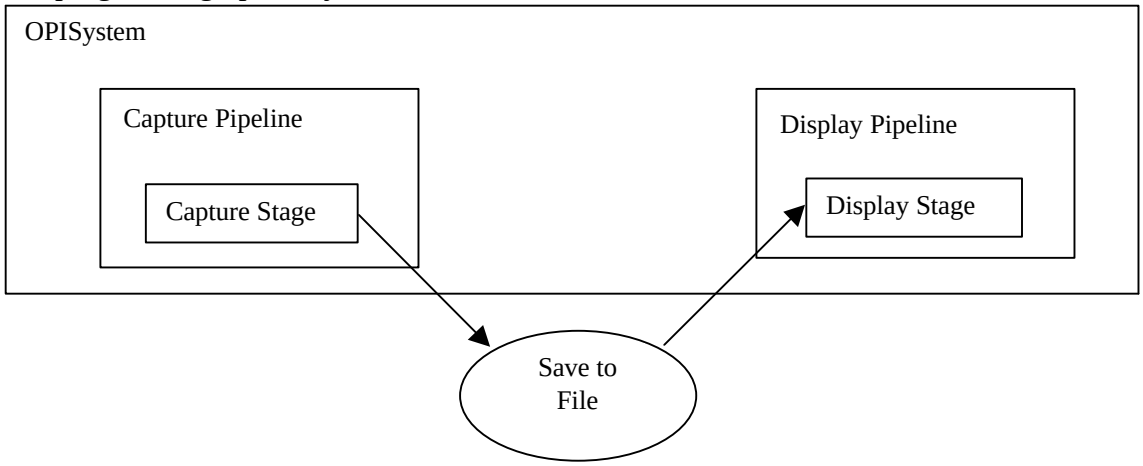


Figure 4-1 Graphic Illustration of vidcapdsp

OPIStream

The *OPIStream* class manages the streaming of data between OPI and the application. A stage at the beginning of a pipeline that does not have an external input (camera, microphone, etc.) will be given data through an *OPIStream*. A stage at the end of a pipeline that does not feed an external output (window, monitor, speakers, etc.) will supply data to the application through an *OPIStream*.

OPIStreams have one of three types. An ***INPUT*** stream supplies data to a stage and acts as the input to a pipeline. An ***OUTPUT*** stream receives data from a stage and acts as the output of a pipeline. The ***PIPE*** stream connects two stages of a pipeline. ***PIPE*** streams are usually created implicitly by OPI.

An *OPIStream* consists of a set of *OPIBuffers* and two queues –a full queue and a free queue. When an *OPIStream* is created, all buffers begin on the free queue. For ***INPUT*** streams, the application must get a buffer off the free queue, place data in the buffer, and then add the buffer to the full queue. OPI will grab the buffer from the full queue, process the data, and then return the buffer to the free queue. For ***OUTPUT*** streams, the opposite occurs. The application must grab a full buffer, process it, and return it to the free queue.

An application interacts with an *OPIStream* through polling or callbacks. An application may occasionally poll OPI to see if there is an available buffer, or the application may set up callbacks through which OPI returns the available buffer. In either case, the application should quickly process the buffers and return the buffer to the stream. If OPI runs out of free buffers, the pipeline may stall and stop providing data.

An *OPIStream* may be implicitly created by OPI or explicitly created by the application. For explicitly created *OPIStreams*, the application first creates an *OPIStream* by calling the *OPIStream* constructor and then attaches the *OPIStream* to an *OPIPipeline*. Attaching the *OPIStream* to an *OPIPipeline* must take place after ***mapToDevices()*** has been called for the *OPIPipeline*, but before ***finishConstruction()*** has been called. During ***finishConstruction()***, OPI will implicitly create any required *OPIStreams* that were not explicitly created.

Creating and setting OPIStreams

After mapping a pipeline, but before finishing construction of a pipeline, you may explicitly define the *OPIStream* to be used to communicate with the pipeline. You may wish to create your own *OPIStream* to control the number of buffers or the size of buffers in the *OPIStream*. The constructor for *OPIStream* requires three parameters: the stream type, the number of buffers and the buffer size. In this example, we will create a stream with zero buffers, because we are going to use buffers from the capture video stream. After creating an *OPIStream*, we attach it to the pipeline through ***setInStream()***.

```
// Map the pipeline
if (dspPipeline->mapToDevices() == FALSE)
{
  cerr << "Unable to map display pipeline" << endl;
  return -1;
}

// Create input stream with no buffers since we will be passing
// buffers directly from the captured video stream to this pipeline
dspVidStream = new OPIStream(INSTREAM, 0, 0);
dspPipeline->setInStream(dspVidStream);

// Finishing the display pipeline construction.
dspPipeline->finishConstruction();
```

Using polling to communicate with an OPIPipeline

In this example, we demonstrate communication with an *OPIStream* through polling. We will poll OPI for available buffers from the capture pipeline. Once we have a buffer, we will add it to the display pipeline. In addition, we must monitor the display pipeline for buffers that have been processed, and place the empty buffers back on the capture pipeline. For this example, we will contain the polling in a *while* loop that will terminate after a given number of buffers have been obtained from the capture pipeline.

The polling loop begins by checking for buffers that have been processed by the display pipeline. The function ***getBuffer()*** will obtain a buffer from the free queue of a stream, or return NULL if there are no free buffers. If there are any free buffers, they are returned to the capture pipeline.

```
  // See if the display pipeline is finished with any buffers.
  // if so, return them to the capture pipeline.
   while ((buf=dspVidStream->getBuffer()) != NULL)
   {
     capVidStream->returnBuffer(buf);
   }
```

Now poll for data from the capture pipeline. To request data from a pipeline, call the ***getData()*** member of the output stream of the pipeline. If a buffer is available, the buffer is taken from the OPI stream and a pointer to the buffer is returned by ***getData()***. If no buffer is available, ***getData()*** will return NULL and the application should try again later.

```
    // Wait for capture buffer to be available
    if ((buf=capVidStream->getData()) == NULL)
    {
      OPISleep(5);
      continue;
    }
```

At this point, we have the data and process it. In this example, we write it to disk.

Finally, we add the buffer to the display pipeline through ***addData()***.

```
// Add buffer to display stream
    dspVidStream->addData(buf);
```

Chapter 5 - OPIWindow and Callbacks

This chapter will use the OPI program, **playraw,** as an illustration. This program demonstrates the use of callbacks to interact with a pipeline and explains the *OPIWindow* class in more detail. This example program will run on any system with OPI installed. For illustration purposes, this example displays video captured in "RGB565" format, which is the capture format used in the previous example. This example is in the ***examples/playraw*** directory of the ODK installation.

OPIWindow

The window defined as the output of the last stage in the previous examples was an instance of an *OPIWindow*. The *OPIWindow* class provides a simple cross-platform method of displaying video in a window on a workstation. When constructing an *OPIWindow*, the application may request OPI to create a new window or to use an existing window already created by the workstation's window manager (X-Windows or Microsoft Windows).

When using the *OPIWindow* class, OPI will attempt to convert uncompressed video to the video format in use on your workstation. For example, suppose you create a display pipeline that has "RGB565" as the input format and "WINDOW" as the output format. If your video display card supports 8-bit color (256 colors), then OPI will convert the "RGB565" video to 8-bit video before displaying on your workstation. If OPI is unable to do the color conversion, then the pipeline mapping will fail.

Some platforms support hardware drawing directly to the video display card (such as DirectDraw on Microsoft Windows). OPI may require that video be displayed in an *OPIWindow* in order to use hardware drawing.

Constructing an OPIWindow

An *OPIWindow* may be used as the output of a CAP_VID_OUTPUT stage. In this example, we explicitly create a new *OPIWindow* by defining the position, size and title of the window.

```
  // Creating the OPI window for the image display.
  OPIWindow* window = NULL;
  window = new OPIWindow(0,0,
                   windowFmt.width,windowFmt.height,
                   "Output");
  if (window  == NULL)
  {
    cerr << "Could not create OPI window\n" ;
    return APP_ERROR;
  }
```

The *OPIWindow* is created at screen coordinates (0,0) has size (windowFmt.width,windowFmt.height) and title of "Output." The translation of the window position and size parameters is dependent on the window manager.

After creating the window, assign the window pointer to the ***param*** member of the *OPIFormatVideo* class defining the output format.

```
windowFmt.param = (void *)window;
```

Callbacks

An OPIStream has two possible callbacks - ***dataDoneCallback*** and ***dataReadyCallback***. The ***dataDoneCallback*** is generated when a buffer has been processed by OPI. This callback is normally used for ***INPUT*** streams. The ***dataReadyCallback*** is generated when OPI has placed data in a buffer. This callback is normally used for ***OUTPUT*** streams.

Setting callbacks

This example uses a ***dataDoneCallback*** to know when the ***INPUT*** stream is finished processing the data. The set callback call must be made after the call to ***mapToDevices*** and before the call to ***finishConstruction***.

```
  // Setting up the callback for the input stream.
  OPIStream* inVidStream = NULL;
  inVidStream = pipeline->getInStream();
  inVidStream->setDataDoneCallback(&DoneBuffer, (void*)this);
```

When setting a callback, you must pass a pointer to the callback function. Callback functions must be static members of a class or non member functions. When setting a callback, you may also pass a ***userData*** pointer that will be passed to the callback function. Frequently, it is helpful for the ***userData*** pointer to be a pointer to a class variable.

Priming the pipeline

In our example, the next step is adding buffers to the pipeline. We obtain all available buffers, fill the buffers with data, and then add them to the pipeline. Once we start the pipeline, and after OPI has processed the data in the buffer, OPI will generate a ***dataDoneCallback*** and return the buffer.

```
//  Prime the pipeline by filling the initial buffers
//  Once the pipeline is started, we can refill buffers
//  in the callback
primedBuffers = 0;
OPIBuffer *vidBuf;
while ((vidBuf = inVidStream->getBuffer()) != NULL)
{
  vidBuf->fsize = fread(vidBuf->data,
                  1, vidBuf->bufferSize(),
                  fd);
  if(vidBuf->fsize  == 0)
  {
     cerr << "No data in input file during priming.\n";
     break;
  }
  primedBuffers++;
  inVidStream->addData(vidBuf);
}
```

After the pipeline is primed, we can call ***start***.

Callback function

When calling the callback function, OPI passes a pointer to the buffer that is now free or full (depending on the callback type). It also passes the ***userData*** pointer given when the callback was set up. The callback function must process the buffer and then return the buffer to the stream. The processing in a callback should take place quickly to insure that OPI does not run out of buffers to use. In this example, we show a ***dataDoneCallback*** generated by an ***INPUT*** stream when a buffer has been processed by OPI. If there is more data in the input file, we read the data into the buffer and place the buffer on the full queue. If there is no data, we place the buffer on the free queue.

```
/*************************************************************
 ** Method:  Done Buffer  (CPlayRaw::DoneBuffer)
 **
 ** Abstract:
 **   Callback to copy more data to buffer and
 **   add buffer to stream
 **
 ** Error Conditions: None
 **
 *************************************************************/
```

When defining a callback function, use the ***CALLBACK*** modifier to insure that the function is properly defined for the operating system.

```
void CALLBACK CPlayRaw::DoneBuffer(OPIBuffer* buffer,
                           void* userData)
{
  CPlayRaw* playPtr = (CPlayRaw*) userData;

  // Read more data from input file into buffer.  If there
  // is no more data, indicate that we have one fewer primed
  // buffers and return buffer to free queue.
  buffer->fsize = fread(buffer->data, 1,
                  buffer->bufferSize(), playPtr->fd);
  if (buffer == 0)
  {
    playPtr->primedBuffers--;
    playPtr->pipeline->getInStream()->returnBuffer(buffer);
  }
  else {
    // Add buffer to full queue
    playPtr->pipeline->getInStream()->addData(buffer);
  }
}
```

Chapter 6 –More Details on OPI Concepts

Chapter 6 has more detail on OPI concepts not covered previously including formats (audio and video) and stream capabilities.

OPI Video and Audio Formats

OPI video and audio formats describe the input/output stream data format that forms the connection between the pipeline stages. These formats are grouped into three types:

- **External**
  The external video or audio format type is used as an input format type to a capture stage or an output format type from an output stage.

- **Uncompressed**
  The uncompressed video or audio format type is used as an input format type to a compression/output stage or an output format type from a decompression stage.

- **Compressed**
  The compressed video or audio format type is used as an input format to a decompression stage or an output format type from a compression stage.

Figure 6-1 shows a graphical representation of the OPI format types.

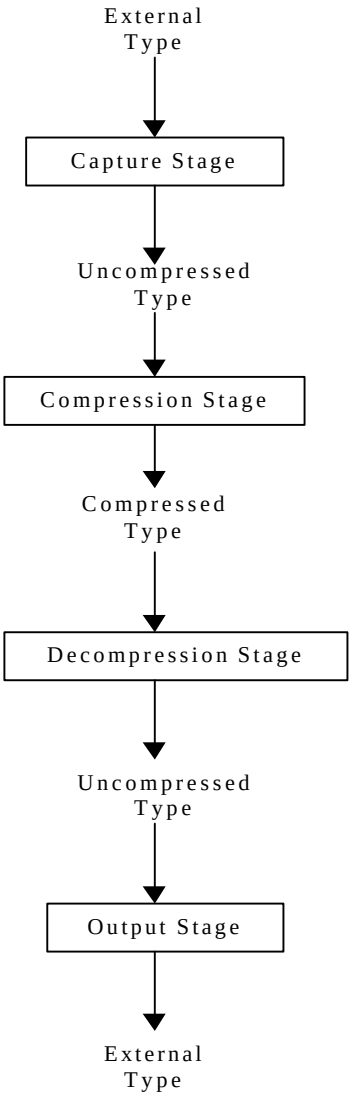


Figure 6-1 Graphical Illustration of OPI Formats

OPI Video Formats

When setting up the external video format type for the initial stage of the pipeline (i.e. capture stage), the only attributes that need to be initialized are the *compression* and the *externalInput* parameters.

When setting decompressed/compressed video format type variables (i.e. decompression/compression stage), the *compression*, *externalInput*, *width*, *height* and *depth* attributes needs to be defined.

Table 6-1, Video Formats lists the various formats associated with each supported device.

Video Formats	Type	Supported By:
NTSC	External	Osprey-1x0 /Osprey-1x00.
PAL	External	Osprey-1x0 /Osprey-1x00.
WINDOW	External	Osprey-1x0 / Osprey-1x00.
RGB555	Uncompressed	Future releases
RGB565	Uncompressed	Osprey-1x0 / Osprey-1x00.
YUV411	Uncompressed	Future releases.
YUV422	Uncompressed	Osprey-1x0 / Osprey-1x00
GRAY8	Uncompressed	Osprey-1x0.
RGB24	Uncompressed	Osprey-1x0.
RGB32	Uncompressed	Osprey-1x0.
IP64	Compressed	Osprey-1x00.

Table 6-1, Video Formats

OPI Audio Formats

Each audio format listed in Table 6-2 is supported by the Osprey-1x00 family. Uncompressed audio formats may have either 1 or 2 channels (mono or stereo). The valid sample settings for uncompressed audio formats are 8000, 11025, 16000, 22050, 32000 and 44100 Hz. All uncompressed audio formats use 8-bit samples except for PCM16, which uses 16-bit samples. The bitrate for an uncompressed audio format is equal to num_channels × sample rate × bits_per_sample. The bitrates for G.722 and G.728 are 64kbps and 16kbps respectively. G.723 supports two bitrates, 5.3kbps and 6.3kbps. When setting the external, uncompressed or compressed audio format type, the ***compression*** attribute is the only attribute that needs to be defined. The code to implement this is shown below.

```
// Set up the recorded format variables
// for the first stage of the pipeline.
OPIFormatAudio captureFmt;
captureFmt.compression   = "ANALOG";

// Set up the decompressed format variables
// for the first stage of the pipeline.
OPIFormatAudio decompressedFmt;
decompressedFmt.compression   = "PCM";

// Set up the compressed format variables
OPIFormatAudio compressedFmt;
compressedFmt.compression = "G728";
```

Audio Formats	Type	Supported by	Comments
ANALOG	External	Osprey-1x00 only	
PCM	Uncompressed	Osprey-1x00 only	Same as PCM8
PCM8	Uncompressed	Osprey-1x00 only	Same as PCM
PCM16	Uncompressed	Osprey-1x00 only	
uLaw	Uncompressed	Osprey-1x00 only	Also known as G.711 uLaw
Alaw	Uncompressed	Osprey-1x00 only	Also known as G.711 Alaw
G.722	Compressed	Osprey-1x00 only	
G.723	Compressed	Osprey-1x00 only	
G.728	Compressed	Osprey-1x00 only	

Table 6-2 Audio Formats

Input & Output Stream Capabilities

A pipeline stage is defined by an input and output format, and the capability of the input and output streams. The types of input and output stream capabilities are as follows:

Capture capability. This has an external input and stream output. It is used for video or audio capture cards.

Possible values are:

CAP_CAPTURE,
CAP_VID_CAPTURE,
CAP_AUD_CAPTURE,
CAP_OTHER_CAPTURE.

Compression capability. This has stream input and output. It takes an uncompressed stream and compresses it to a desired format.

Possible values are:

CAP_COMPRESSION,
CAP_VID_COMPRESSION,
CAP_AUD_COMPRESSION,
CAP_OTHER_COMPRESSION.

Decompression capability. This has stream input and output. It also takes a compressed stream and decompresses it to a raw format.

Possible values are:

CAP_DECOMPRESSION,
CAP_VID_DECOMPRESSION,
CAP_AUD_DECOMPRESSION,
CAP_OTHER_DECOMPRESSION.

Output capability. Has stream input and external output. It takes the stream data and places it on some output device like a monitor or speakers.

Possible values are:

CAP_OUTPUT,
CAP_VID_OUTPUT,
CAP_AUD_OUTPUT,
CAP_OTHER_OUTPUT.

Chapter 7 – Hardware Specific Capabilities

This chapter discusses the specific hardware capabilities of the Osprey-1x0 and the Osprey-1x00 family of cards. Developers are encouraged to stick to generic capabilities whenever feasible. This will allow their programs to run unmodified on new hardware as we deliver it. However, one of the goals of OPI is to provide low-level access to the hardware to improve performance and to exploit the full capabilities of the card. Thus, each specific card may have card specific capabilities as well as the generic ones supported by all OPI supported cards.

The Osprey-1x0 family

The Osprey-1x0 family consists of the Osprey-100 and Osprey-150. The Osprey-100 card is a PCI card that runs on the Windows platform. The Osprey-150 is a PCI card that runs on Sun's Solaris and Compaq's Digital UNIX platforms. Both the Osprey-100 and Osprey-150 supply uncompressed video streams that are used with today's exciting multimedia applications. The Osprey-100 is not supported in our current release but it will be supported in future releases. In the current release, the Osprey-150 has the following attributes:

Attribute	Type	Min	Max	Effect
"Video Brightness"	INT	-128	127	input video brightness
"Video Contrast"	INT	0	511	input video contrast
"Video Hue"	INT	-128	127	input video hue
"Video Gain U"	INT	0	511	input U gain
"Video Gain V"	INT	0	511	input V gain
"Video Port"	INT	0	2	0 = Svideo, 1 = Composite1, 2 = Composite2
"Video Format"	INT	0	2	0=NTSC, 1=PAL

The Osprey-1x00 family

The Osprey-1x00 family is a family of video and audio compression/decompression cards. They all use the 8x8 VCP chip for video compression/decompression and the Analog Devices DSP 2181 for audio compression/decompression. The family includes:

- **Osprey-1000**

  The Osprey-1000 is a PCI card that runs on Windows platforms.

- **Osprey-1100**

  The Osprey-1100 is an SBUS card that runs on the Solaris platform.

- **Osprey-1500**

  The Osprey-1500 is a PCI card that runs on UNIX platforms.

- **SunVideo Plus**

  SunVideo Plus is a SUN OEM product. This PCI card runs on Solaris platforms. It differs from the other cards in the Osprey-1x00 line in that it supports two composite video inputs, instead of 1 composite input and one composite output.

In the current release, the Osprey-1x00 family has the following attributes:

Attribute	Type	Min	Max	Effect
"Audio Input Select"	INT	0	1	0 = Line In, 1= Microphone In
"Audio Volume Left"	INT	128	255	left speaker volume
"Audio Volume Right"	INT	128	255	right speaker volume
"Audio Gain Left"	INT	0	255	left gain
"Audio Gain Right"	INT	0	255	right gain
"Audio Echo Cancellation"	INT	0	1	AEC = 1 is on
"Audio Mixing"	INT	128	255	audio mixing
"Video Port"	INT	0	2	0 = Svideo, 1 = Composite1, 2 = Composite2[1]
"Video Brightness"	INT	-128	127	input video brightness
"Video Hue"	INT	-128	127	input video hue
"Video Contrast"	INT	0	511	input video contrast
"Video Gain U"	INT	0	511	input U gain
"Video Gain V"	INT	0	511	input V gain
"Video Out"	INT	0	1	1=video out port, not on-screen video[2]
"Video Format"	INT	0	2	0=PAL, 1=NTSC, 2=AUTO[3]

[1] Only applicable to SunVideo Plus.

[2] Not applicable to SunVideo Plus .

[3] Auto Detect is not applicable to Osprey-1000.

Attribute	Type	Min	Max	Effect
"Hardware Draw"	INT	0	1	Use DirectDraw if set.[4]
"Video hint"	String			See below for details.
"Audio hint"	String			See below for details.

Audio and Video hints

The firmware for the Osprey-1x00 is downloaded to the card when ***finishConstruction*** is called for the first pipeline that uses the card. The audio and video firmware must be downloaded simultaneously. Therefore, if you are creating both an audio and a video pipeline for an Osprey-1x00, the first pipeline to be constructed must contain a "hint" as to what format the other pipeline will be using. You must set the "Audio hint" or "Video attribute for the device before calling ***finishConstruction*** on the first pipeline for that device. The way to implement the setting of the audio hint before the call to ***finishConstruction*** is shown below.

```
OPISystem *msys;
OPIDeviceInstance *inst;
…
OPIPipeline audioPipeline = new OPIPipeline(msys);

… <define and map pipeline>

OPIAttribute hint("G723");
inst->setAttribute("Audio hint" hint);

audioPipeline->finishConstruction();
```

[4] Direct Draw is only applicable on the NT platform.

Video Attributes

As a general rule of thumb, you should set the "Video Out" and "Video Format" attributes before calling ***finishConstruction*** on any pipeline. Once ***finishConstruction*** is called, firmware is downloaded to the Osprey-1x00 and those attributes cannot be changed.

A single instance of an Osprey-1x00 cannot handle pipelines with different video attributes. For example, you cannot map a pipeline that captures NTSC to the same device instance as a pipeline that outputs PAL. You also cannot map a pipeline that generates a video out signal (attribute "Video Out" = 1) to the same device instance as a pipeline that sends video to an *OPIWindow* or data buffers.

When a video pipeline is constructed, the appropriate video attributes are changed to match the pipeline. For example, if a decompression pipeline ends in a stage with "PAL" as the output, the attribute "Video Out" is set to 1 (i.e. video out port) and "Video Format" is set to 0 (i.e. PAL) during ***mapToDevices*** routine call. An error will occur if a pipeline construction attempts to change attributes that have been fixed because of a previous firmware download.

The Osprey-1x00 card can generate a signal on its video out jack only if it decompresses data in hardware. Currently in OPI, the only way to generate a video out signal is through decompressing H261. The Osprey-1x00 cannot display uncompressed data through the video out jack. For example, a pipeline with RGB565 as input and NTSC as its output format will not map.

Codec

A codec is a software or hardware coder/decoder that encodes an analog stream(video or audio) into a compressed digital format, then decodes and decompresses the digital data back into analog data.

The Osprey-1x00 card is capable of running both a video and an audio codec simultaneously. This same device instance *cannot* run two video codecs at the same time or two audio codecs at the same time. Users cannot use the same device instance to compress H261 video format and decompress H263 video format (when H263 becomes available) simultaneously. Neither can they use a device instance to simultaneously

compress G.722 audio format and decompress G.723 audio format. The same restriction also applies to uncompressed formats. For example, a device instance cannot simultaneously capture in YUV422 video format while decompressing an H261 video format. If a user has two device instances of the Osprey-1x00 card in the system, the user may use the first device instance to run a video codec and use the next device instance to run a different codec. Table 7-1 shows the available codecs that are currently supported by the Osprey-1x00 hardware.

Uncompressed Video	**Compressed Video**	**Uncompressed Audio**	**Compressed Audio**
YUV422	H261 (IP64)	PCM8	G722
RGB565		PCM16	G723
		G711	G728

Table 7-1 Codecs supported by the Osprey-1x00 hardware.

Chapter 8 - Attribute Control

Devices can have specific attributes that are controlled through the device attribute interface. Examples of such attributes might be brightness or which video port to use.

Setting attributes on a known named device

The simplest way to set an attribute for a device is if the name of the device, the device instance, and the attribute are all known. The following code fragment illustrates setting the "Video Brightness" attribute on the "o1c0" instance of the Osprey-1x0.

```
OPISystem* sys = new OPISystem();
OPIDeviceInstance* di = sys->getControlInstance("o1c", "o1c0");
di->setAttribute("Video Brightness", (long)brightness);
```

Usually, control panel code should be written to work directly on an *OPIDeviceInstance* in a pipeline, or should query for the supported instances in the system.

In addition to the *control panel* and *attrlist* examples below, there is a UNIX command line interface program *cpanel*, which illustrates many of the same concepts.

Control Panel Example

The control panel and attrlist examples illustrate attribute control. This control panel application is specific to the Osprey 1000, while attrlist is germane to all OPI devices. The code excerpts are from Windows NT control panel examples. The OPI specific portions will be the same for Solaris or DEC, although the specifics of coding the Graphical User Interface differ. The topics that will be covered for the control panel example are:

- opening the OPI System.
- obtaining the instances of the device.
- obtaining the attributes for a device instance.

Obtaining the instances of the device

To obtain the Osprey 1000 device, use the ***getDeviceByName*** call. To obtain the name of the device instances, use the *OPIDevice* class member variable ***instances***. To obtain the pointer to the device instance, use the *OPIDevice* class member function ***getInstanceByName***.

```
OPIDevice* dp;
OPIDeviceInstance* dp_inst;

…(Begin extracted code)
dp = opi->getDeviceByName(“o1k”);
if (dp == NULL)
{
   cerr << “No o1k device installed\n”;

}
// Getting the number of instances of the device
num_of_instances = dp->instances.getNumItems();
…(End extracted code)

for (int j = 0; j < num_of_instances; j++)
{
       (Begin extracted code)

   // Get the name and the ptr to the device instance.
   inst_name = dp->instances.getItemName(j);
   dp_inst = dp->getInstanceByName(inst_name);
      …(End extracted code)
}
```

Obtaining the attributes for a device instance.

To obtain an attribute with a known name, use ***getAttribute***.

```
OPIAttribute attribute;
device_instance->getAttribute(“name”, &attribute);
```

To determine the type of the attribute, use ***getAttributeInfo***. The possible types of an attribute can be an integer, float, string or a generic pointer.

```
OPIAttributeInfo info;
device_instance->getAttributeInfo("name", &info);
```

To get a list of attributes that are associated with a particular device instance, use the *OPIDeviceInstance* class member variable ***availableAttributes***.

```
int num_of_attr;
char* name_of_attr;
OPIAttribute max_value;
OPIAttribute min_value;
…
// Getting the number of attributes on the device instance
num_of_attr = dp_inst->availableAttributes.getNumItems();

for (int I = 0; I < num_of_attr; i++)
{
  // Getting the name of the attribute
  name_of_attr = dp_inst->availableAttributes.getItemName(i);

  // Getting the min and max value set for the attribute
  dp_inst->getAttributeInfo(name_of_attr, &info);
  max_value = info->getMax();
  min_value = info->getMin();

}
```

Attrlist Example

This Windows example displays a list of device instances and their associated attributes. It uses a C++ tree class to display all attributes, their value, min, and max. The topics that will be covered are:

- listing the instances for each device.
- listing the attributes for a device instance.

Listing the instances for each device

To get a list of device instances, use the *OPIDevice* class member variable ***instances.*** For hardware devices, this example will list the number of device instances that are currently on your system. For software devices, this example will list the number of software instances that are currently available on your system.

```
for (j=0; j < dp->instances.getNumItems(); j++)
{
      tree.InsertItem(dp->instances.getItemName(j), device);
}
```

Listing the attributes for a device instance

To get a list of attributes for a device instance, use the *OPIDeviceInstance* class member variable ***availableAttributes***.

```
OPINameList* nl = &inst->availableAttributes;
int I;
for (I=0; i < nl->getNumItems(); I++)
{
      int att = attributes.InsertItem(0, nl->getItemName(i));
}
```

Chapter 9 - OPI Developer's Kit Examples

This chapter contains discussions and code excerpts for additional OPI examples that are included in the OPI Developer's Kit.

Record Example

The record example captures raw video, displays an uncompressed image from the capture device to a window, and converts an uncompressed image (RGB565) into a compressed image, (H.261, also known as IP64) which is then saved to a file. The source code for this example is included in the ODK. Figure 9-1 shows a graphical illustration of the record example found in the **example/record** subdirectory.

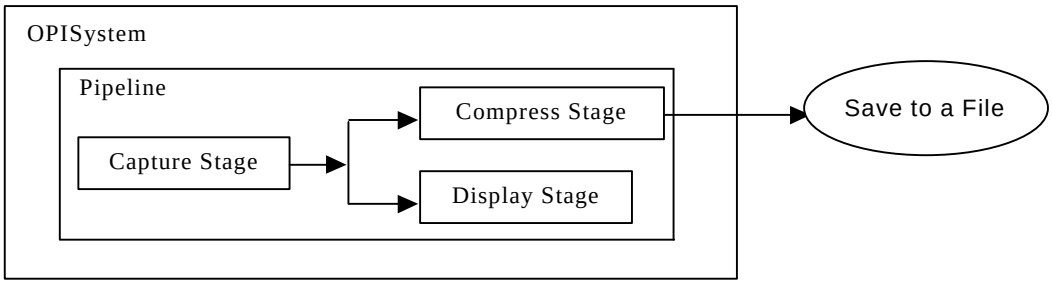


Figure 9-1 Graphical Illustration of the Record Example

The topics that will be covered for the record example are:

- creating a branching pipeline.
- setting up the compression and capture attributes of the device.

Setting up the formats for the various stages of the pipeline

This sets up the recorded, uncompressed, compressed, and window format variables of the desired stream data. The *recordedFmt.compression* type is set to "NTSC." "NTSC" is an external video format type used as an input format to a capture stage. The *uncompressedFmt.compression* type is set to "RGB565," an uncompressed video format type that serves as an output format from the capture stage and an input to the compression stage. The *compressedFmt.compression* type is set to "IP64" which is a compressed video format type that serves as the output for the compression stage. The *windowFmt.compression* type is set to "WINDOW" as it serves as the output format for the window stage.

```
// Set up the recorded format variables
// for the first stage of the pipeline.
OPIFormatVideo recordedFmt;
recordedFmt.compression   = "NTSC";
recordedFmt.externalInput = TRUE;

// Set up the uncompressed format variables
// for the first stage of the pipeline.
OPIFormatVideo uncompressedFmt;
uncompressedFmt.width         = 160;
uncompressedFmt.height        = 120;
uncompressedFmt.depth         = 16;
uncompressedFmt.compression   = "RGB565";
uncompressedFmt.externalInput = TRUE;

// Set up the compressed format variables
// for the second stage of the pipeline.
OPIFormatVideo compressedFmt;
compressedFmt.compression   = "IP64";
compressedFmt.width         = 160;
compressedFmt.height        = 120;
compressedFmt.depth         = 16;
compressedFmt.externalInput = TRUE;

// Setting the window format variables that are needed for the
// creation of the opi window for the local view while
// video is being captured to the file.
OPIFormatVideo windowFmt;
windowFmt.compression    = "WINDOW";
windowFmt.externalOutput = FALSE;
windowFmt.width          = 160;
windowFmt.height         = 120;
windowFmt.depth          = 16;
```

Creating the branching stages of the pipeline

Begin by creating the first stage of the pipeline for the capture operation. Next, add the second stage of the pipeline to handle video compression, and add another branching stage to handle the uncompressed video display to a window.

Create a branch in a pipeline by calling ***addStage*** twice with the same stage as the first parameter.

```
// Create the first stage of the pipeline.
OPIStage* captureStage;
captureStage = pipeline->firstStage(CAP_VID_CAPTURE, &recordedFmt,
                                                    &decompressedFmt);

// Creating the 2nd stage of the pipeline.
OPIStage* videoCompressionStage;
videoCompressionStage = pipeline->addStage(captureStage,
           CAP_VID_COMPRESSION, &decompressedFmt, &compressedFmt);

// Creating a branch of pipeline
// for the viewing of the captured video.
pipeline->addStage(captureStage,CAP_VID_OUTPUT, &decompressedFmt,
                                                          &windowFmt);
```

Setting the compression and capture attributes of the device

Once the pipeline maps to the devices successfully, you can get the handle of the device that implements each stage of the pipeline by calling the routine ***getDeviceFromHandle***. To set some device specific attributes, or to set explicit buffering, the set routine calls have to be made before the ***finishConstruction*** routine call.

```
// Getting the specific compression device and the capture device.
// Setting some of the attributes of the compression device and
// capture device.
OPICompressionDeviceH261 *cdH261;
cdH261 = (OPICompressionDeviceH261*)
             pipeline->getDeviceFromHandle(videoCompressionStage);
cdH261->setQuality(15);
cdH261->setBitRate(256);
cdH261->setBlockRefreshPeriod(3000);

OPICaptureDeviceVideo *cap;
cap = (OPICaptureDeviceVideo*)
          pipeline->getDeviceFromHandle(captureStage);
cap->setFrameRate(50);    // in milliseconds per frame
cap->setOverlayRate(100); // set the frame rate for local view.
```

Finishing the construction of the pipeline

After completing any changes to the devices, the ***finishConstruction*** routine is called to indicate that the API is now free to set up any implicit buffers and is ready for a ***start*** routine call.

```
// Finish the pipeline construction
pipeline->finishConstruction();
```

Loopback Example

In the loopback example, we capture & compress using the first instance of the Osprey-1000 card (i.e. o1k0), and decompress & display using the last instance of the Osprey-1000 card (i.e. o1k (n-1)). For this example, it could have been implemented as two separate pipelines, but by implementing it as a single pipeline, there is no need for the program to deal with the transfer of data from one pipeline to the other. Note that if you have one board installed, the program will use the same board for both the capture & compression, and the decompression & display operations. The source code is included in the ODK in the directory ***examples/loopback***. Figure 9-2 shows a graphical illustration of the loopback example.

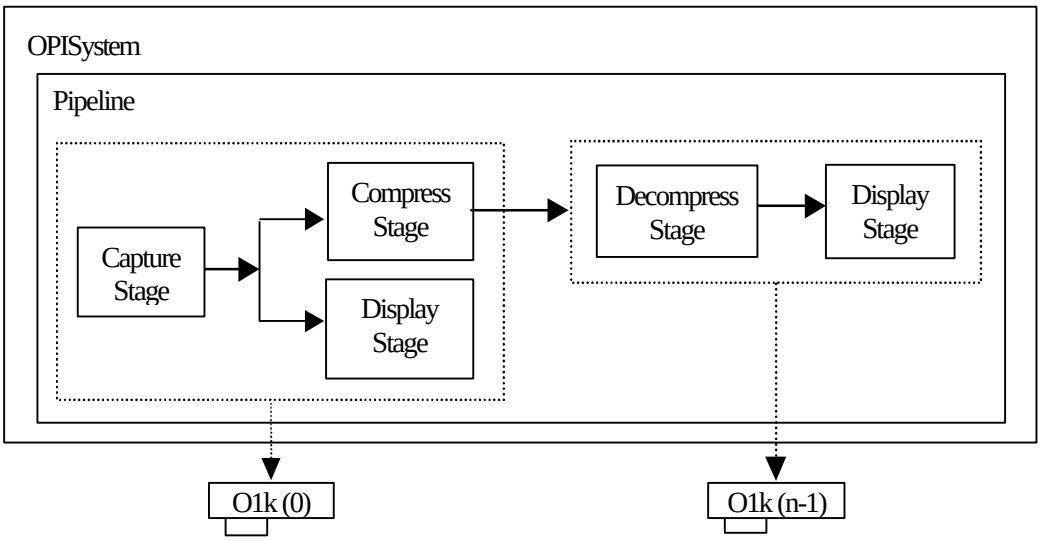


Figure 9-2 Graphical Illustration of the loopback example

The topics that will be covered for the loopback example are:

- getting device specific instance names.
- forcing a pipeline to use a particular device.
- setting device specific parameters.

Getting device specific instance names

In this excerpt, we determine the name of the last o1k device in the system.

```
  OPIDevice* dp;
  dp = opi->getDeviceByName("o1k");
  if(dp==NULL)
    {
      cerr << "No o1k device installed\n";
      return APP_ERROR;
    }

  // Getting the number of instances of the device
  num_of_instances = dp->instances.getNumItems();

  // Get the pointer to the last instance of the device.
  OPIDeviceInstance* inst =
      dp->getInstanceByIndex(num_of_instances - 1);

  // Get the name of the last instance of the device.
  char* selected_device = inst->getName();
```

Forcing a pipeline to use a particular device

A particular stage of a pipeline can be forced to use a particular device by assigning the device name as part of the input or output format. The *captureFmt* variable ***deviceName*** is set to the first instance of the Osprey-1000 card (i.e. o1k0), and the *decompressedFmt* variable ***deviceName*** is set to the last instance of the Osprey-1000 card (i.e., o1k1).

```
decompressedFmt.deviceName = selected_device;
```

Setting device specific parameters

Note that although there are multiple devices assigned to the pipeline, it is still possible to retrieve any particular device by passing the ***OPIStage*** handle to the ***getDeviceFromHandle*** routine call. After retrieving the device handle, you can set the capabilities of the device before calling the ***finishConstruction*** routine call.

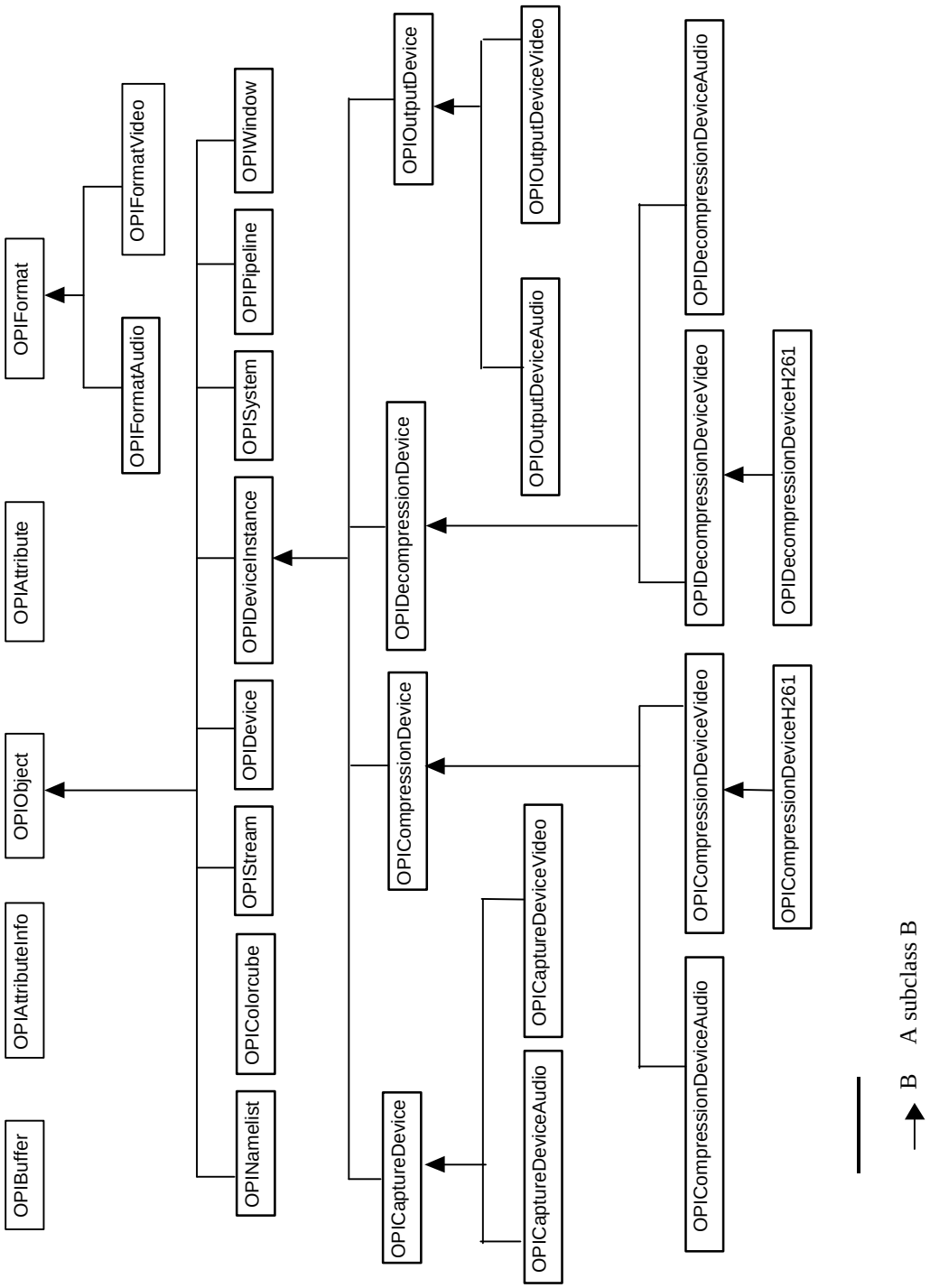
```
// Getting the specific compression device and setting some of the
// capabilities of the compression device.
OPICompressionDeviceH261 *cdH261;
cdH261 = (OPICompressionDeviceH261 *)
    pipeline->getDeviceFromHandle(videoCompressionStage);
cdH261->setQuality(31);
cdH261->setBitRate(1024);
cdH261->setBlockRefreshPeriod(3000);

// Getting the specific capture device and setting the framerate
// and the overlay rate.
OPICaptureDeviceVideo *cap = OPICaptureDeviceVideo *)
    pipeline->getDeviceFromHandle(videoCompressionStage);
cap->setFrameRate(33);    // frame rate is measured in msec per frame
cap->setOverlayRate(1000);   // Set frame rate for local video
```

Appendix A - Class Reference

Introduction

This appendix contains all the class definitions that are available to you in the OPI C++ class library. Each class shown is listed alphabetically with a description of the class, a short description of its methods, the parameters, and return codes.


OPIBuffer
OPIAttributeInfo
OPIObject
OPIAttribute
OPIFormat
OPIFormatVideo
OPIFormatAudio
OPINamelist
OPIColorcube
OPIStream
OPIDevice
OPIDeviceInstance
OPISystem
OPIPipeline
OPIWindow
OPICaptureDevice
OPICaptureDeviceAudio
OPICaptureDeviceVideo
OPICompressionDevice
OPICompressionDeviceAudio
OPICompressionDeviceVideo
OPICompressionDeviceH261
OPIDecompressionDevice
OPIDecompressionDeviceAudio
OPIDecompressionDeviceVideo
OPIDecompressionDeviceH261
OPIOutputDevice
OPIOutputDeviceAudio
OPIOutputDeviceVideo
A subclass B

OPIAttribute

class OPIAttribute

The ***OPIAttribute*** class provides a way to pass integers, floats, strings, or generic pointers to or from a device.

Constructor

OPIAttribute ()
Creates a new OPIAttribute object.

OPIAttribute (*const char** s)
Creates a new OPIAttribute object and initializes it to the specified string value.

Parameters**:**
s –string type.

OPIAttribute (*const void** p)
Creates a new OPIAttribute object and initializes it to the specified pointer value.

Parameters**:**
p –pointer type.

OPIAttribute (*const long* i)
Creates a new OPIAttribute object and initializes it to the specified long integer value.

Parameters**:**
i –long integer type.

OPIAttribute (*const double* d)
Creates a new OPIAttribute object and initializes it to the specified double value.

Parameters**:**
d –double type.

OPIAttributeInfo

class OPIAttributeInfo

Member Functions:

getAttributeType
getGettable
getMax
getMin
getSettable
getType

The ***OPIAttributeInfo*** class provides information about an attribute, including whether the attribute has read and/or write access, what type the device expects, and optionally minimum and maximum values.

Constructor

OPIAttributeInfo (*OPIBool* get, *OPIBool* set, *OPIAttributeType* typep,
OPIAttribute maxp = *OPIAttribute*((long)0),
OPIAttribute minp = *OPIAttribute*((long)0))

Creates a new ***OPIAttributeInfo*** object and initializes it to the specified value.

Parameters**:**

get – *OPIBool* value.
Either TRUE or FALSE.
set – *OPIBool* value.
Either TRUE or FALSE.
typep – type of attribute.
Possible values: PTR_ATTRIBUTE
STRING_ATTRIBUTE
DOUBLE_ATTRIBUTE
INT_ATTRIBUTE
minp – minimum value (defaults to zero).
maxp – maximum value (defaults to zero).

Member Functions

OPIAttributeType
getAttributeType ()
Get the type of attribute.

Returns**:**
type.

OPIBool
getGettable ()
Get read access of the attribute.

Returns**:**
TRUE if the attribute has read access. FALSE otherwise.

OPIAttribute
getMax ()
Get the maximum values (for integer attributes only).

Returns:
max.

OPIAttribute
getMin ()
Get the minimum values (for integer attributes only).

Returns:
min.

OPIBool
getSettable ()

Gets write access of the attribute.

Returns:

TRUE if the attribute has write access, FALSE otherwise.

OPIAttributeType
getType ()

As in the **getAttributeType,** this gets the type of the attribute.

Returns:

type.

OPIBuffer

class OPIBuffer

Member Variables:

data
fsize
userData

Member Functions:

bufferSize
setSize

The ***OPIBuffer*** class creates a buffer to hold the input/output data stream to devices. This class is used by the *OPIStream* class. The size of the data in the buffer is obtained by accessing the *fsize* data member. To create a buffer that points to a user defined section of memory, create a 0 byte buffer, and then set *data* and *fsize* appropriately.

Constructor

OPIBuffer (*int* numBytes = 0)

Creates a new ***OPIBuffer*** object. By default, it creates a buffer with zero numBytes.

Parameters:

numBytes – Number of bytes.

Destructor

~**OPIBuffer** ()

Member Variables

*OPIBytes** **data**
Pointer to the actual data.

long **fsize**
Size of the data in the buffer.

*void** **userdata**
Can be set by a user for data associated with a buffer.

Member Functions

long
bufferSize ()
Returns the size of the actual buffer.

Returns:
Size of the actual buffer allocated.

void
setSize (*long* s)
Set the size of the actual buffer.

Parameters**:**
s - size of the actual buffer.

OPICaptureDeviceAudio

class OPICaptureDeviceAudio
inherits OPICaptureDevice.

Member Functions:
getChannels
getSampleRate
setChannels
setSampleRate

The ***OPICaptureDeviceAudio*** class provides generic capabilities for all audio capture devices. The sample rate and channels can be accessed via its member functions.

Member Functions

int
getChannels ()
Get the channels of the device.

Returns:
channels.

void
setChannels (*int* channels)
Set the channels of the device.

Parameters:
channels - Possible values:
1 - mono
2 - stereo

int
getSampleRate ()
Get the sample rate of the device.

Returns:
sample rate.

void
setSampleRate (*int* hz)
Set the sample rate of the device.

Parameters:
hz - Sample Rate Settings.

OPICaptureDeviceVideo

class OPICaptureDeviceVideo
inherits OPICaptureDevice.

Member Functions:
getFrameRate
getOverlayRate
setFrameRate
setOverlayRate

The ***OPICaptureDeviceVideo*** class provides generic capabilities for all video capture devices. The frame and overlay rate can be accessed via its member functions.

Member Functions

int
getFrameRate ()
Get the frame rate of the actual captured image.

Returns:
milliseconds per frame.

int
getOverlayRate ()
Get the overlay rate of the local video view of the captured image.

Returns:
milliseconds per frame.

void
setFrameRate (*int* msecPerFrame)
Set the desired frame rate capture.

Parameters:
msecPerFrame – Milliseconds per frame.

void
setOverlayRate (*int* msecPerFrame)
Set the overlay rate of the local video view of the captured image.

Parameters:
msecPerFrame – Milliseconds per frame.

OPIColorcube

class OPIColorcube
inherits OPIObject.

Member Functions:
getColors
getColorspace
getLimits
getMult
getOffset
getSize
setLimits

Static Member Functions:
getDefaultColorcube

An ***OPIColorcube*** can be used to set or get values for an X Colormap (for X systems), or to set the values used in an RGB8 capture (for X or Windows). Colorcubes are defined by 3 bands, the starting colorspace, and the offset (i.e. RGB 5:8:5, offset 10, would be an RGB colorcube with 8 values of G for each 5 of R and 5 of B). The colorcube will start at offset <*offset*>.

Each band can have a limit range (i.e. 10 - 230, instead of 0-255). This allows the values to be allocated across a smaller range, providing better quality in the mid-range.

The **getColors** call only returns the valid values. For instance, if you have an offset of 10, then band0[0] would be the value assigned to pixel index 10 for red.

Constructor

OPIColorcube (*ColorSpace* cs, *int* offset, *int* mult0, *int* mult1, *int* mult2)

Creates an *OPIColorcube* and initializes it to the specified colorspace, offset and size.

Parameters**:**

cs - *ColorSpace* –Colorcube space.
Possible values: YUV, RGB, GRAY

offset –the offset of the first index in the colorcube

mult0 , mult1, mult2 –the number of values per band. I.e., for an 8:5:5 RGB colorcube, the mult values are 8,5, and 5.

Destructor

~OPIColorcube ()

Member Functions

*unsigned char**
getColors (*int* i)

Get the colors of a single band of the colorcube.

Parameters:

i –the index of the band to get. (0= RED, 1=GREEN, 2=BLUE).

Returns**:**

An array of the color values for a particular band.

Colorspace
getColorspace ()

Get the colorspace of the colorcube.

Returns**:**

colorspace.

void
getLimits (*int* band, *int* &min, *int* &max)

Get the minimum and maximum range values for a particular band.

Parameters:

band – The band to which the limit is being applied.

Returns:

min, max – the minimum and maximum values for the band.

void
getMult (*int* &mult0, *int* &mult1, *int* &mult2)
Gets the dimensions of the colorcube.

Returns:
mult0, mult1, mult2: the dimensions of the colorcube.

int
getOffset ()
Returns the offset.

Returns**:**
an integer value.

int
getSize ()
Get the size of colorcube.

Returns**:**
size of colorcube.

void
setLimits (*int* band, *int* min, *int* max)
Set the minimum and maximum range values for a particular band.

Parameters:
band - the band to which the limit is being applied.
min, max - the minimum and maximum values for the band.

Static Member Functions:

OPIColorcube
GetDefaultColorcube (*ColorSpace* cs)

Get the standard system defined colorcube. There is one of these for each colorspace.

Parameters**:**

ColorSpace - Colorcube space.

Possible values: YUV, RGB, GRAY.

Returns**:**

an OPIColorcube.

OPICompressionDeviceAudio

OPICompressionDeviceAudio

inherits OPICompressionDevice.

Member Functions:

getSampleBlockSize

The ***OPICompressionDeviceAudio*** class provides generic capabilities for all audio compression devices. The sample block size can be assessed via its member function.

Member Functions

int

getSampleBlockSize ()

Get the sample block size of the audio data. Note that audio data has to be sent in blocks. For audio formats G.728 and G.723, the block size are 10 bytes and 20-24 bytes respectively.

Returns:

sample block size.

OPICompressionDeviceH261

class OPICompressionDeviceH261
inherits OPICompressionDeviceVideo.

Member Functions:

getBlockRefreshPeriod
getPIP
getPIPLocation
setBlockRefreshPeriod
setPIP
setPIPLocation

The ***OPICompressionDeviceH261*** class describes a H261 (IP64) compression device. Both Picture-in-Picture location and status can be accessed via their member functions. The refresh period can also be accessed via its member functions.

Member Functions

int
getBlockRefreshPeriod ()
Get the Block Refresh Period of the device.

Returns:
Block refresh period in milliseconds.

OPIBool
getPIP ()
Check to see if PIP is set.

Returns:
TRUE if PIP is set, FALSE otherwise.

int
getPIPLocation ()
Get the PIP location of the device.

Returns:
PIP Location.
Possible return values:
0 –lower left hand corner
1 –upper left hand corner
2 –lower right hand corner
3 –upper right hand corner

void
setBlockRefreshPeriod (*int* milliseconds)
Setting the Block Refresh Period of the device.
BlockRefreshPeriod refers to the time between each forced update of a macroblock of the image.

Parameters:
milliseconds –the block refresh rate.

void
setPIP (*OPIBool* PIP)
Set PIP for the device.

Parameters**:**
PIP –TRUE if you wish to set PIP, FALSE otherwise.

void
setPIPLocation (*int* location)
Set the PIP location of the device.

Parameters:
Location – integer value.
Possible values:
0 –lower left hand corner
1 –upper left hand corner
2 –lower right hand corner
3 –upper right hand corner

OPICompressionDeviceVideo

class OPICompressionDeviceVideo
inherits OPICompressionDevice.

Member Functions:
getBitRate
getQuality
setBitRate
setQuality

The ***OPICompressionDeviceVideo*** class provides generic capabilities for all video compression devices. The quality and bit rate can be accessed via its member functions.

Member Functions

int
getBitRate ()
Get the bit rate of the device.

Returns:
bit rate.

int
getQuality ()
Get the quality rate of the device. Quality ranges from 1 to 31, where 1 is the highest quality.

Returns:
quality rate.

void
setBitRate (*int* bitspersecond)
Set the bit rate of the device.

Parameters:
bitspersecond - the bit rate.

void
setQuality (*int* q)
Set the quality rate of the device. Quality ranges from 1 to 31, where 1 is the highest quality.

Parameters:
q - the quality rate.

OPIDecompressionDeviceAudio

class OPIDecompressionDeviceAudio
inherits OPIDecompressionDevice.

Member Functions:
getSampleBlockSize

The ***OPIDecompressionDeviceAudio*** class describes the generic capabilities for all audio decompression devices. The sample block size can be accessed via its member functions.

Member Functions

int
getSampleBlockSize ()
Get the sample block size of the audio data. Note that audio data has to be sent in blocks. For audio formats G.728 and G.723, the block size are 10 bytes and 20-24 bytes respectively.

Returns**:**
sample block size.

OPIDevice

class OPIDevice
inherits OPIObject.

Member Variables:
instances

Member Functions:
getAudioCapabilities
getInstanceByName
getInstanceByNdx
getName
getVideoCapabilities

The ***OPIDevice*** class describes the basic capabilities of an OPI device, which may be a hardware or software implementation. It provides a member variable to allow the user to get access to the list of all attributes that are associated with the device. The video or audio capabilities, the name of the device and the pointer or name of the device instance can be accessed via its member functions.

Constructor

OPIDevice ()
Creates a new *OPIDevice* object.

Destructor

~OPIDevice ()

Member Variables

OPINameList **instances**

instances is a list of all attributes that are associated with a particular device.

Member Functions

long

getAudioCapabilities ()

Get the audio capabilities. *(See section* Input & Output Stream Capabilities *on page 37).*

Returns:

audio capabilities.

*OPIDeviceInstance**

getInstanceByName (*const char** name)

Get the pointer to the device instance given the name of the device instance.

Parameters:

name –name of the device instance.

Returns:

pointer to the device instance.

*OPIDeviceInstance**

getInstanceByNdx (*int* index)

Get the pointer to the device instance given the index location of the device instance.

Parameters:

index –index location of the device instance.

Returns:

pointer to the device instance.

*char**
getName ()
Get the name of the device.

Returns:
name of the device.

long
getVideoCapabilities ()
Get the video capabilities. *(See* Input & Output Stream Capabilities *on page 37.)*

Returns:
video capabilities.

OPIDeviceInstance

class OPIDeviceInstance
inherits OPIObject.

Member Variables:
availableAttributes
inFmt
outFmt

Member Functions:
getAttribute
getAttributeInfo
getDevice
getName
setAttribute

The ***OPIDeviceInstance*** class describes an instance of an *OPIDevice*. It provides member variables to allow the user to query the input and output formats for the device instance, and gets a list of available attributes through the *availableAttributes* member variable. The name of the device, attribute settings and attribute information can be accessed via its member functions.

Constructor

While the constructors are public, this type is created by *OPISystem*, not directly by a user program.

Destructor

~OPIDeviceInstance()

Member Variables

OPINamelist **availableAttributes**
The *availableAttributes OPINameList* provides a list of available attributes.

*OPIFormat** **inFmt**
Input Format.

*OPIFormat** **outFmt**
Output Format.

Member Functions

OPIBool
getAttribute (*char** name, *OPIAttribute** value)
Get the values of the attribute "name."

Returns:
TRUE if successful, FALSE otherwise.

OPIBool
getAttributeInfo (*char** name, *OPIAttributeInfo *** value)
Gets information about the attribute, including type and optionally, minimum and maximum values. It could be an integer, float, string or a generic pointer.

Parameters:
name –name of the attribute
value –assigned the *OPIAttributeInfo* structure.

Returns:
TRUE if successful, FALSE otherwise.

*OPIDevice**
getDevice ()
Get the device type associated with this instance.

Returns:
OPIDevice.

*char**
getName ( )
Get the name of the device instance.

Returns:
name of the device.

OPIBool
setAttribute (*char** name, *OPIAttribute* value)
Set the attribute of the device.

Parameters:
name –name of the device.
value –value to be set.

Returns:
TRUE if successful, FALSE otherwise.

OPIFormat

class OPIFormat

Member Variables:

compression
deviceName
externalInput
externalOutput
param

Member Functions:

dup

The ***OPIFormat*** class defines general formats of the stream data that is desired for the pipeline stages (i.e. audio, video, or other). It provides member variables to allow the user to initialize the external input/output, the type of compression, param value and the device name.

Member Variables

*char** **compression**
Details about compression can be found in the section *OPI Video and Audio Formats* on page 33.

*char** **deviceName**
Name of the device to use.

OPIBool **externalInput**
External input. Either TRUE or FALSE.

OPIBool **externalOutput**
External output. Either TRUE or FALSE.

*void** **param**

This parameter passes extra information to *compression* types that need it. The following table lists current *compression* types that can receive param information.

Compression	Param Value
"WINDOW"	OPIWindow Handle
"RGB8"	OPIColorcube

Member Functions

virtual *OPIFormat**
dup () = 0

duplicates the *OPIFormat* variables.

OPIFormatAudio

class OPIFormatAudio
inherits OPIFormat.

Member Functions:
dup

This ***OPIFormatAudio*** class defines the audio format of the stream data that is desired for the pipeline stage. It provides a member function to clone an existing instance of the ***OPIFormatAudio*** object. *(See Section* OPI Audio Formats *on page 36 for more information on supported formats).*

Constructor

OPIFormatAudio ()
Creates a new ***OPIFormatAudio*** object.

Member Functions

*OPIFormat******
dup ()
Clone an existing instance of ***OPIFormatAudio***.

Returns**:**
OPIFormatInstance.

OPIFormatVideo

class OPIFormatVideo
inherits OPIFormat.

Member Variables:
depth
height
width

Member Functions:
dup

This ***OPIFormatVideo*** class defines the video format of the stream data that is desired for the pipeline stage. For a video format type, it has attributes like width, height and depth. Cloning an existing instance of the ***OPIFormatVideo*** object can be accessed via its member function. *(See the section* OPI Video Formats *on page 34 for more information.)*

Constructor

OPIFormatVideo ()
Creates a new ***OPIFormatVideo*** object.

Member Variables

int **depth**
Depth of the image in bits.

int **height**
Height of the image in pixels.

int **width**
Width of the image in pixels.

Member Functions:

*OPIFormat**
dup ()

Clone an existing instance of ***OPIFormatVideo***.

Returns**:**

OPIFormat.

OPINameList

class OPINameList
inherits OPIObject.

Member Functions
getItemName
getNumItems
nameNdx

The ***OPINameList*** class is a general-purpose list data structure that is used by several classes to list the name/value pairs. For instance, **OPIDevice** uses the ***OPINamelist*** to list all attributes associated with that particular device.

Constructor

OPINameList ()
Creates a new ***OPINameList*** object.

Member Functions

*char**
getItemName (*int* i)
Get the item name associated with i.

Parameters:
i - item index.
Returns:
item name.

int
getNumItems ()
Get the number of items in the ***OPINamelist***.

Returns:
The number of items.

int
nameNdx (*const char** name)
Get the index of the item.

Parameters:
name - item name.

Returns:
item index, -1 if not in the list.

OPIOutputDeviceAudio

class OPIOutputDeviceAudio
inherits OPIOutputDevice.

Member Functions:
getChannels
getSampleRate
setChannels
setSampleRate

The ***OPIOutputDeviceAudio*** class describes the generic capabilities of all audio output devices. The sample rate and channels can be accessed via its member functions.

Member Functions

int
getChannels ()
Get the channels of the device.

Returns:
channels.

int
getSampleRate ()
Get the sample rate of the device.

Returns:
sample rate.

void
setChannels (*int* channels)
Set the channels of the device.

Parameters:
channels - Possible values:
1 - mono
2 - stereo

void
setSampleRate (*int* hz)
Set the sample rate of the device.

Parameters:
hz - Sample Rate Settings.

OPIPipeline

class OPIPipeline
inherits OPIObject.

Member Functions:

addStage
finishConstruction
firstStage
getDeviceFromHandle
getInStream
getOutStream
mapToDevices
pause
resume
setInStream
setOutStream
setThreadPriority
start
stop

The ***OPIPipeline*** class provides the basic construct to create the various pipeline stages of operations. It handles the streaming of data by connecting the input of one stream to the output of another. Pipelines must be constructed in start to finish order using the ***firstStage*** and ***addStage*** routines. To get the pointer to the device, get/set the input/output streams, and other major pipeline operations like start and stop can be accessed via its member functions.

Constructor

OPIPipeline (*OPISystem** msys)

Creates an *OPIPipeline* object.

Parameters**:**

msys - *OPISystem*

Destructor

~OPIPipeline ()

Member Functions

*OPIStage**

addStage (*OPIStage** sh, *int* capability, *OPIFormat** inFmt, *OPIFormat** outFmt)

Add a new stage to a pipeline. By calling **addStage** multiple times with the same value for sh, you can create a branch in the pipeline.

Parameters:

sh - Stage handle.

capability - there are 4 types of capabilities supported by the cards or software modules *(See section* Input & Output Stream Capabilities *on page 37).*

inFmt - input format.

outFmt - output format

Returns:

a stage handle.

OPIBool
finishConstruction ()

Finish the pipeline construction to indicate that the API is free to set up any implicit buffers and ready for a start call.

Returns:

TRUE if successful, FALSE otherwise.

*OPIStage**
firstStage (*int* capability, *OPIFormat** inFmt, *OPIFormat** outFmt)

Create the first stage of the pipeline by giving it the capability, input format and output format.

Parameters:

See ***addStage*** for details on individual parameters.

Returns:

the input stream.

*OPIDeviceInstance**
getDeviceFromHandle (*OPIStage** sh)

Get the handle of the specific device that will execute this stage of the device pipeline. This handle can be used to set up any explicit buffering and to change device specific attributes. This call is only valid after the ***mapToDevices*** call has returned successfully.

Parameters:

sh - an *OPIStage*.

Returns:

A device instance.

*OPIStream**
getInStream ()

Gets the input stream.

Returns:

the input stream.

*OPIStream**
getOutStream ()
Gets the output stream.

Returns:
the output stream.

OPIBool
mapToDevices ()
Map the pipeline stages to the device by assigning the actual hardware and software to each of the required actions. This routine is called after the various stages of the pipelines have been set up.

Returns:
TRUE if successful, FALSE otherwise.

void
pause ()
Pause the pipeline.

void
resume ()
Resume the pipeline.

void
setInStream (OPIStream* stream)
Set the input stream. This can be done explicitly by the user, or if left null, will be handled by the system.

Parameters:
Stream – input stream.

void
setOutStream (OPIStream* stream)
Set the output stream. This can be done explicitly by the user, or if left null, will be handled by the system.

Parameters:
Stream – output stream.

void
start ()
Start the pipeline.

void
stop ()
Stop the pipeline.

void
setThreadPriority (int priority)
Set the priority of the thread. At the time of this release, this member function is only valid for Windows NT applications.

Parameters:
priority – priority level.

OPIStream

class OPIStream
inherits OPIObject.

Member Functions:
addData
getBuffer
getData
isInternal
returnBuffer
setBuffer
setDataDoneCallback
setDataReadyCallback

The ***OPIStream*** class describes the most basic element in the OPI API. It is used to provide data to the input or output of a device instance. It can also be used as the conduit between two device instances within a pipeline. Data streaming handling can be done through either polling or callbacks. This class provides member functions to allow the user to set up the buffer data area, get the buffer, get the data streams and set up the callbacks for the input/output routines. Note that streams are unidirectional, but may be either an **INSTREAM**, an **OUTSTREAM**, or a **PIPE**.

Constructor

OPIStream
(*OPIStreamType* stype,
long num_buffers,
long buffer_size,
OPIBool internal = TRUE,
OPIBool parse = TRUE)

Creates an ***OPIStream*** object and initializes it to the specified stype, buffers, buffer_size, internal and parse.

Parameters**:**

stype - a stream type.

enum **OPIStreamType**

{INSTREAM = 1, OUTSTREAM = 2, PIPE = 3};

num_buffers - number of buffers.

buffer_size - size of buffer.

internal - defaults to TRUE. If set to FALSE, developer must create their own buffers by calling the ***setBuffer*** routine.

parse - defaults to TRUE. This variable does nothing at the time of this release. It will be supported in future releases.

Destructor

~OPIStream()

Member Functions

OPIBool
addData (*OPIBuffer** buffer)

Give the buffer to the actor.

Returns:

TRUE if successful, FALSE otherwise.

*OPIBuffer**
getBuffer ()

Get a buffer to put data in.

Returns:

OPIBuffer, if available, null otherwise.

*OPIBuffer**
getData ()
Get data as available buffer.

Returns:
OPIBuffer, if available, null otherwise.

OPIBool
isInternal ()

Returns:
Returns TRUE if the library created the buffers, FALSE if the user created the buffers.

void
returnBuffer (*OPIBuffer** buffer)
Done with this buffer.

Parameters:
buffer - the buffer to return.

void
setBuffer (OPIByte* data, *long* numBytes)
Sets a buffer's data area.
If using external buffers, it is the user's responsibility to call the ***setBuffer*** () routine *nbuffer* number of times to set up the proper data pointers.

Parameters:
data - buffer data.
numBytes - size of the buffer.

void
setDataDoneCallback (*dataDoneCallback* cb, void* userData)

Set ***dataDoneCallback*** for the input routines. This callback is called when the pipeline finishes with a buffer.

Parameters:

cb –done callback function.
userData –This value will be passed to the callback routine.

void
setDataReadyCallback (*dataReadyCallback* cb, void* userData)

Set ***dataReadyCallback*** for the output routines. This callback is called when data is ready to be pulled off a stream.

Parameters:

cb –ready callback function.
userData –This value will be passed to the callback routine.

OPISystem

class OPISystem
inherits OPIObject.

Member Variables:
devices

Member Functions:
getControlInstance
getDeviceByName
setDebugLevel

An instance of the ***OPISystem*** class has to be created prior to any OPI API calls. This class initializes the OPI library. Note that there may be a limitation of only one instance of the ***OPISystem*** class in a process.

Constructor

OPISystem ()
Creates a new ***OPISystem*** object.

Destructor

~OPISystem ()

Member Variables

OPINameList **devices**

OPINameList **devices** can be used to get a list of all devices in the system

Member Functions

*OPIDeviceInstance**

getControlInstance (*const char** device_name, *const char** instance_name)

Get access to a device instance based on the device and instance name. This device instance can only be used to get and set attributes.

Parameters:

device_name - the name of the device that is installed in your system.
instance_name - the instance name of the device.

Returns:

OPIDeviceInstance pointer of the device instance.

*OPIDevice**

getDeviceByName (*const char** device_name)

Get access to device.

Parameters:

device_name - the name of the device that is installed in your system.

Returns:

OPIDevice pointer of the device.

void
setDebugLevel (*int* level)

Control the amount of debug information provided by OPI System. Currently, the only effect this has is disabling detection of streaming timeouts if passed a non-zero value.

Parameters:

level - a non-zero integer value.

OPIWindow

class OPIWindow
inherits OPIObject.

Member Functions:
getHandle
isInternal
resize
getColorcube
setColorcube

The ***OPIWindow*** class describes a display window managed by OPI library. In order to display uncompressed video to a window, the end of the pipeline must be an ***OPIWindow*** object.

Constructor

OPIWindow (*int* x, *int* y, *int* width, *int* height, *char** name)
Construct an ***OPIWindow*** and initialize it to the specified x-coordinates, specified y-coordinates, width, height and title name.

Parameters**:**
x –x coordinates.
y –y coordinates.
width –window width.
height –window height.
name –window title.

OPIWindow (*void** window)

OPIWindow (*struct HWND__ *window)

The above two constructors do the same thing. They construct an ***OPIWindow*** object from an already existing display window.

Parameters**:**

window - For Windows applications *HWND* is declared as *type struct HWND__ *,*
Non-Windows applications declare *HWND* as *type void*.*

Destructor

~OPIWindow()

Member Functions

HWND
getHandle ()

Gets the pointer to the handle.

Returns**:**

HWND - In the case of Solaris, an **HWND** is a pointer to the following structure:

```
typedef struct {
        Display *xdisplay;
        Window xwindow;
}WND;
```

In the case of Win32 programming, an **HWND** is an MFC HWND or Win32 window.

OPIBool
isInternal ()

Returns:
TRUE if the window was created by the OPI library, otherwise, FALSE.

void
resize ()
Request the window to be resized.

*OPIColorcube**
getColorcube ()
To get the colorcube that OPI has assigned to a window.

Returns:
The colorcube that OPI has assigned to a window.

void
setColorcube (*OPIColorcube** cc)
Set the desired colorcube for an 8bit Xwindow and ignore the default colorcube.

Parameters:
cc - desired colorcube.

Appendix B -C Interface

The OPI library includes a C interface. This interface is simply a wrapper to the C++ interface, so the C++ interface is preferred. To use the C interface, a program must include the "opi_c.h" file.

OPI C Types

The C++ classes are represented in C through opaque C types. The name is the C++ class name with "c" prepended. The mapping of C types to analogous C++ classes is provided in Tables B-1 & B-2 below. For more information on individual C++ classes consult Appendix A.

C types	***C++ classes***
c_dataReadyCallBack	dataReadyCallBack
cOPIAttribute	OPIAttribute
cOPIAttributeInfo	OPIAttributeInfo
cOPIBuffer	OPIBuffer
cOPICaptureDevice	OPICaptureDevice
cOPICaptureDeviceAudio	OPICaptureDeviceAudio
cOPICaptureDeviceVideo	OPICaptureDeviceVideo
cOPIColorcube	OPIColorcube
cOPICompressionDevice	OPICompressionDevice
cOPICompressionDeviceAudio	OPICompressionDeviceAudio
cOPICompressionDeviceH261	OPICompressionDeviceH261
cOPICompressionDeviceVideo	OPICompressionDeviceVideo
cOPIDecompressionDeviceAudio	OPIDecompressionDeviceAudio
cOPIDevice	OPIDevice
cOPIDeviceInstance	OPIDeviceInstance
cOPIFormat	OPIFormat
cOPIFormatAudio	OPIFormatAudio
cOPIFormatVideo	OPIFormatVideo
cOPINameList	OPINameList
cOPIOutputDeviceAudio	OPIOutputDeviceAudio
cOPIPipeline	OPIPipeline
cOPIStage	OPIStage
cOPIStream	OPIStream
cOPISystem	OPISystem
cOPIWindow	OPIWindow

Table B-1 Mapping of C types to C++ classes.

C++ classes	***C types***
dataReadyCallBack	c_dataReadyCallBack
OPIAttribute	cOPIAttribute
OPIAttributeInfo	cOPIAttributeInfo
OPIBuffer	cOPIBuffer
OPICaptureDevice	cOPICaptureDevice
OPICaptureDeviceAudio	cOPICaptureDeviceAudio
OPICaptureDeviceVideo	cOPICaptureDeviceVideo
OPIColorcube	cOPIColorcube
OPICompressionDevice	cOPICompressionDevice
OPICompressionDeviceAudio	cOPICompressionDeviceAudio
OPICompressionDeviceH261	cOPICompressionDeviceH261
OPICompressionDeviceVideo	cOPICompressionDeviceVideo
OPIDecompressionDeviceAudio	cOPIDecompressionDeviceAudio
OPIDevice	cOPIDevice
OPIDeviceInstance	cOPIDeviceInstance
OPIFormat	cOPIFormat
OPIFormatAudio	cOPIFormatAudio
OPIFormatVideo	cOPIFormatVideo
OPINameList	cOPINameList
OPIOutputDeviceAudio	cOPIOutputDeviceAudio
OPIPipeline	cOPIPipeline
OPIStage	cOPIStage
OPIStream	cOPIStream
OPISystem	cOPISystem
OPIWindow	cOPIWindow

Table B- 2 Mapping of C++ classes to C types

OPI C Functions

The C++ member functions are represented as C calls where the first parameter is the C type that represents the C++ class. Calls to "new C++Class" are replaced with class_name_create () calls. The mapping of these calls is shown in Tables B-3 & B-4. Information on individual parameters is available in the matching C++ reference section in Appendix A.

C Functions	***C++ Member Functions***
opi_ai_getAttributeType	OPIAttributeInfo :: getAttributeType
opi_ai_getGettable	OPIAttributeInfo :: getGettable
opi_ai_getMax	OPIAttributeInfo :: getMax
opi_ai_getMin	OPIAttributeInfo :: getMin
opi_ai_getSettable	OPIAttributeInfo :: getSettable
opi_ai_getType	OPIAttributeInfo :: getType
opi_buffer_bufferSize	OPIBuffer :: bufferSize
opi_buffer_create	OPIBuffer :: OPIBuffer
opi_buffer_delete	OPIBuffer :: ~OPIBuffer
opi_buffer_getData	OPIBuffer :: data
opi_buffer_getFrameCount	OPIBuffer :: frame_count
opi_buffer_getFrameSlices	OPIBuffer :: frame_slices
opi_buffer_getFsize	OPIBuffer :: fsize
opi_buffer_getUserData	OPIBuffer :: userData
opi_buffer_setFrameCount	OPIBuffer :: frame_count
opi_buffer_setFrameSlices	OPIBuffer :: frame_slices
opi_buffer_setFsize	OPIBuffer :: fsize
opi_buffer_setSize	OPIBuffer :: setSize
opi_buffer_setUserData	OPIBuffer :: userData
opi_capAudio_getChannels	OPICaptureDeviceAudio :: getChannels

C Functions	***C++ Member Functions***
opi_capAudio_getSampleRate	OPICaptureDeviceAudio :: getSampleRate
opi_capAudio_setChannels	OPICaptureDeviceAudio :: setChannels
opi_capAudio_setSampleRate	OPICaptureDeviceAudio :: setSampleRate
opi_capVideo_getFrameRate	OPICaptureDeviceVideo :: getFrameRate
opi_capVideo_getOverlayRate	OPICaptureDeviceVideo :: getOverlayRate
opi_capVideo_setFrameRate	OPICaptureDeviceVideo :: setFrameRate
opi_capVideo_setOverlayRate	OPICaptureDeviceVideo :: setOverlayRate
opi_cd_getCompType	OPICompressionDevice :: getCompType
opi_colorcube_create	OPIColorcube::OPIColorcube()
opi_colorcube_delete	OPIColorcube::~OPIColorcube()
opi_colorcube_getColors	OPIColorcube::getColors
opi_colorcube_getColorSpace	OPIColorcube::getColorSpace
opi_colorcube_getDefaultColorcube	OPIColorcube::getDefaultColorcube
opi_colorcube_getLimits	OPIColorcube::getLimits
opi_colorcube_getMult	OPIColorcube::getMult
opi_colorcube_getOffset	OPIColorcube::getOffset
opi_colorcube_getSize	OPIColorcube::getSize
opi_colorcube_setLimits	OPIColorcube::setLimits
opi_compAudio_getSampleBlockSize	OPICompressionDeviceAudio :: getSampleBlockSize
opi_compH261_getBlockRefreshPeriod	OPICompressionDeviceH261 :: getBlockRefreshPeriod
opi_compH261_getPIP	OPICompressionDeviceH261 :: getPIP
opi_compH261_getPIPLocation	OPICompressionDeviceH261 :: getPIPLocation
opi_compH261_setBlockRefreshPeriod	OPICompressionDeviceH261 :: setBlockRefreshPeriod

C Functions	***C++ Member Functions***
opi_compH261_setPIP	OPICompressionDeviceH261 ::
opi_compH261_setPIPLocation	OPICompressionDeviceH261 :: setPIPLocation
opi_compVideo_getBitRate	OPICompressionDeviceVideo :: getBitRate
opi_compVideo_getQuality	OPICompressionDeviceVideo :: getQuality
opi_compVideo_setBitRate	OPICompressionDeviceVideo :: setBitRate
opi_compVideo_setQuality	OPICompressionDeviceVideo :: setQuality
opi_decompAudio_getSampleBlockSize	OPICompressionDeviceAudio :: getSampleBlockSize
opi_device_delete	OPIDevice :: ~OPIDevice
opi_device_getAudioCapabilities	OPIDevice :: getAudioCapabilities
opi_device_getInstanceByIndex	OPIDevice :: getInstanceByIndex
opi_device_getInstanceByName	OPIDevice :: getInstanceByName
opi_device_getInstances	OPIDevice :: instances
opi_device_getName	OPIDevice :: getName
opi_device_getVideoCapabilities	OPIDevice :: getVideoCapabilities
opi_di_getAttributeInfo	OPIDeviceInstance :: getAttributeInfo
opi_di_getAttribute	OPIDeviceInstance :: getAttribute
opi_di_getAvailableAttributes	OPIDeviceInstance :: availableAttributes
opi_di_getDevice	OPIDeviceInstance :: getDevice
opi_di_getInFmt	OPIDeviceInstance :: inFmt
opi_di_getName	OPIDeviceInstance :: getName
opi_di_getOutFmt	OPIDeviceInstance :: outFmt
opi_di_setAttribute	OPIDeviceInstance :: setAttribute
opi_fmt_aud_create	OPIFormatAudio :: OPIFormatAudio
opi_fmt_aud_setCommon	OPIFormatAudio :: OPIFormat

C Functions	***C++ Member Functions***
opi_fmt_vid_create	OPIFormatVideo :: OPIFormatVideo
opi_fmt_vid_setCommon	OPIFormatVideo :: OPIFormat
opi_win_getColorcube	OPIWindow :: getColorcube
opi_nl_getItemName	OPINameList :: getItemName
opi_nl_getNumItems	OPINameList :: getNumItems
opi_nl_nameNdx	OPINameList :: nameNdx
opi_outAudio_getChannels	OPIOutputDeviceAudio :: getChannels
opi_outAudio_getSampleRate	OPIOutputDeviceAudio :: getSampleRate
opi_outAudio_setChannels	OPIOutputDeviceAudio :: setChannels
opi_outAudio_setSampleRate	OPIOutputDeviceAudio :: setSampleRate
opi_pipe_addStage	OPIPipeline :: addStage
opi_pipe_create	OPIPipeline :: OPIPipeline
opi_pipe_delete	OPIPipeline :: ~OPIPipeline
opi_pipe_finishConstruction	OPIPipeline :: finishConstruction
opi_pipe_firstStage	OPIPipeline :: firstStage
opi_pipe_getDeviceFromHandle	OPIPipeline :: getDeviceFromHandle
opi_pipe_getInStream	OPIPipeline :: getInStream
opi_pipe_getOutStream	OPIPipeline :: getOutStream
opi_pipe_mapToDevices	OPIPipeline :: mapToDevices
opi_pipe_pause	OPIPipeline :: pause
opi_pipe_resume	OPIPipeline :: resume
opi_pipe_setInStream	OPIPipeline :: setInStream
opi_pipe_setOutStream	OPIPipeline :: setOutStream
opi_pipe_start	OPIPipeline :: start
opi_pipe_stop	OPIPipeline :: stop
opi_win_setColorcube	OPIWindow :: setColorcube

C Functions	***C++ Member Functions***
opi_stream_addData	OPIStream :: addData
opi_stream_create	OPIStream :: OPIStream
opi_stream_delete	OPIStream :: ~OPIStream
opi_stream_getBuffer	OPIStream :: getBuffer
opi_stream_getData	OPIStream :: getData
opi_stream_isInternal	OPIStream :: isInternal
opi_stream_returnBuffer	OPIStream :: returnBuffer
opi_stream_setBuffer	OPIStream :: setBuffer
opi_stream_setDataDoneCallback	OPIStream :: setDataDoneCallback
opi_stream_setDataReadyCallback	OPIStream :: setDataReadyCallback
opi_sys_create	OPISystem :: OPISystem
opi_sys_delete	OPISystem :: ~OPISystem
opi_sys_getControlInstance	OPISystem :: getControlInstance
opi_sys_getDeviceByName	OPISystem :: getDeviceByName
opi_sys_getDevices	OPISystem :: devices
opi_sys_setDebugLevel	OPISystem :: setDebugLevel
opi_win_create	OPIWindow :: OPIWindow
opi_win_create1	OPIWindow :: OPIWindow
opi_win_create2	OPIWindow :: OPIWindow
opi_win_delete	OPIWindow :: ~OPIWindow
opi_win_getHandle	OPIWindow1 :: getHandle
opi_win_isInternal	OPIWindow :: isInternal
opi_win_resize	OPIWindow :: resize

Table B- 3 Mapping of C functions to C++ member functions

C++ Member Functions	***C Functions***
OPIAttributeInfo :: getAttributeType	opi_ai_getAttributeType
OPIAttributeInfo :: getGettable	opi_ai_getGettable
OPIAttributeInfo :: getMin	opi_ai_getMin
OPIAttributeInfo	opi_ai_getMin
OPIAttributeInfo :: getSettable	opi_ai_getSettable
OPIAttributeInfo :: getType	opi_ai_getType
OPIBuffer :: ~OPIBuffer	opi_buffer_delete
OPIBuffer :: bufferSize	opi_buffer_bufferSize
OPIBuffer :: data	opi_buffer_getData
OPIBuffer :: frame_count	opi_buffer_getFrameCount
OPIBuffer :: frame_count	opi_buffer_setFrameCount
OPIBuffer :: frame_slices	opi_buffer_getFrameSlices
OPIBuffer :: frame_slices	opi_buffer_setFrameSlices
OPIBuffer :: fsize	opi_buffer_getFsize
OPIBuffer :: fsize	opi_buffer_setFsize
OPIBuffer :: OPIBuffer	opi_buffer_create
OPIBuffer :: setSize	opi_buffer_setSize
OPIBuffer :: userData	opi_buffer_getUserData
OPIBuffer :: userData	opi_buffer_setUserData
OPICaptureDeviceAudio :: getChannels	opi_capAudio_getChannels
OPICaptureDeviceAudio :: getSampleRate	opi_capAudio_getSampleRate
OPICaptureDeviceAudio :: setChannels	opi_capAudio_setChannels
OPICaptureDeviceAudio :: setSampleRate	opi_capAudio_setSampleRate
OPICaptureDeviceVideo :: getFrameRate	opi_capVideo_getFrameRate
OPICaptureDeviceVideo :: getOverlayRate	opi_capVideo_getOverlayRate

C++ Member Functions	***C Functions***
OPICaptureDeviceVideo :: setFrameRate	opi_capVideo_setFrameRate
OPICaptureDeviceVideo :: setOverlayRate	opi_capVideo_setOverlayRate
OPIColorcube::OPIColorcube()	opi_colorcube_create
OPIColorcube::~OPIColorcube()	opi_colorcube_delete
OPIColorcube::getColors	opi_colorcube_getColors
OPIColorcube::getColorSpace	opi_colorcube_getColorSpace
OPIColorcube::getDefaultColorcube	opi_colorcube_getDefaultColorcube
OPIColorcube::getLimits	opi_colorcube_getLimits
OPIColorcube::getMult	opi_colorcube_getMult
OPIColorcube::getOffset	opi_colorcube_getOffset
OPIColorcube::getSize	opi_colorcube_getSize
OPIColorcube::setLimits	opi_colorcube_setLimits
OPICompressionDevice :: getCompType	opi_cd_getCompType
OPICompressionDeviceAudio :: getSampleBlockSize	opi_compAudio_getSampleBlockSize
OPICompressionDeviceAudio :: getSampleBlockSize	opi_decompAudio_getSampleBlockSize
OPICompressionDeviceH261 :: getBlockRefreshPeriod	opi_compH261_getBlockRefreshPeriod
OPICompressionDeviceH261 :: getPIP	opi_compH261_getPIP
OPICompressionDeviceH261 :: getPIPLocation	opi_compH261_getPIPLocation
OPICompressionDeviceH261 :: setBlockRefreshPeriod	opi_compH261_setBlockRefreshPeriod
OPICompressionDeviceH261 :: setPIP	opi_compH261_setPIP
OPICompressionDeviceH261 :: setPIPLocation	opi_compH261_setPIPLocation
OPICompressionDeviceVideo :: getBitRate	opi_compVideo_getBitRate

C++ Member Functions	***C Functions***
OPICompressionDeviceVideo :: getQuality	opi_compVideo_getQuality
OPICompressionDeviceVideo :: setBitRate	opi_compVideo_setBitRate
OPICompressionDeviceVideo :: setQuality	opi_compVideo_setQuality
OPIDevice :: ~OPIDevice	opi_device_delete
OPIDevice :: getAudioCapabilities	opi_device_getAudioCapabilities
OPIDevice :: getInstanceByIndex	opi_device_getInstanceByIndex
OPIDevice :: getInstanceByName	opi_device_getInstanceByName
OPIDevice :: getName	opi_device_getName
OPIDevice :: getVideoCapabilities	opi_device_getVideoCapabilities
OPIDevice :: instances	opi_device_getInstances
OPIDeviceInstance :: availableAttributes	opi_di_getAvailableAttributes
OPIDeviceInstance :: getAttribute	opi_di_getAttribute
OPIDeviceInstance :: getAttributeInfo	opi_di_getAttributeInfo
OPIDeviceInstance :: getName	opi_di_getName
OPIDeviceInstance :: setAttribute	opi_di_setAttribute
OPIFormatAudio :: OPIFormat	opi_fmt_aud_setCommon
OPIFormatAudio :: OPIFormatAudio	opi_fmt_aud_create
OPIFormatVideo :: OPIFormat	opi_fmt_vid_setCommon
OPIFormatVideo :: OPIFormatVideo	opi_fmt_vid_create
OPINameList :: getItemName	opi_nl_getItemName
OPINameList :: getNumItems	opi_nl_getNumItems
OPINameList :: nameNdx	opi_nl_nameNdx
OPIOutputDeviceAudio :: getChannels	opi_outAudio_getChannels
OPIOutputDeviceAudio :: getSampleRate	opi_outAudio_getSampleRate
OPIOutputDeviceAudio :: setChannels	opi_outAudio_setChannels
OPIOutputDeviceAudio :: setSampleRate	opi_outAudio_setSampleRate

C++ Member Functions	***C Functions***
OPIPipeline :: ~OPIPipeline	opi_pipe_delete
OPIPipeline :: addStage	opi_pipe_addStage
OPIPipeline :: finishConstruction	opi_pipe_finishConstruction
OPIPipeline :: firstStage	opi_pipe_firstStage
OPIPipeline :: getDeviceFromHandle	opi_pipe_getDeviceFromHandle
OPIPipeline :: getInStream	opi_pipe_getInStream
OPIPipeline :: getOutStream	opi_pipe_getOutStream
OPIPipeline :: inFmt	opi_di_getInFmt
OPIPipeline :: mapToDevices	opi_pipe_mapToDevices
OPIPipeline :: OPIPipeline	opi_pipe_create
OPIPipeline :: outFmt	opi_di_getOutFmt
OPIPipeline :: pause	opi_pipe_pause
OPIPipeline :: resume	opi_pipe_resume
OPIPipeline :: setInStream	opi_pipe_setInStream
OPIPipeline :: setOutStream	opi_pipe_setOutStream
OPIPipeline :: start	opi_pipe_start
OPIPipeline :: stop	opi_pipe_stop
OPIStream :: ~OPIStream	opi_stream_delete
OPIStream :: addData	opi_stream_addData
OPIStream :: getBuffer	opi_stream_getBuffer
OPIStream :: getData	opi_stream_getData
OPIStream :: isInternal	opi_stream_isInternal
OPIStream :: OPIStream	opi_stream_create
OPIStream :: returnBuffer	opi_stream_returnBuffer
OPIStream :: setBuffer	opi_stream_setBuffer
OPIStream :: setDataDoneCallback	opi_stream_setDataDoneCallback

C++ Member Functions	***C Functions***
OPIStream :: setDataReadyCallback	opi_stream_setDataReadyCallback
OPISystem :: ~OPISystem	opi_sys_delete
OPISystem :: devices	opi_sys_getDevices
OPISystem :: getControlInstance	opi_sys_getControlInstance
OPISystem :: getDevice	opi_di_getDevice
OPISystem :: getDeviceByName	opi_sys_getDeviceByName
OPISystem :: OPISystem	opi_sys_create
OPISystem :: setDebugLevel	opi_sys_setDebugLevel
OPIWindow :: ~OPIWindow	opi_win_delete
OPIWindow :: getColorcube	opi_win_getColorcube
OPIWindow :: getHandle	opi_win_getHandle
OPIWindow :: isInternal	opi_win_isInternal
OPIWindow :: OPIWindow	opi_win_create
OPIWindow :: OPIWindow	opi_win_create1
OPIWindow :: OPIWindow	opi_win_create2
OPIWindow :: resize	opi_win_resize
OPIWindow :: setColorcube	opi_win_setColorcube

Table B-4 Mapping of C++ member functions to C functions

Appendix C - OS Specific Details

Introduction

OPI is currently supported under Windows (NT, 95, 98), Solaris, and Compaq's Digital UNIX. The following OS specific information includes how to link with the library, where files are installed, and how to determine which devices are installed. Normally, everything that is needed to run OPI programs is included in the hardware installation software.

Windows

Installation

Installation of the ODK uses normal InstallShield technology. Just run the executable, and follow the instructions of the InstallShield Wizard.

Include Files

opi.h and opi_c.h are installed in the path supplied when installing the developer's kit.

Link Directives

To build, you must link with otiopi.lib. It is installed in the path supplied when installing the developer's kit.

OPI Dynamically Loadable Libraries

All DLLs are installed to WINNT\SYSTEM32 for NT or WINDOWS\SYSTEM for 95/98.

Installed Devices

A device list is in HKEY_LOCAL_MACHINE\SOFTWARE\Osprey\OPI\Devices. This list shows the mapping of device name to library name.

Solaris

Installation

Packages are provided for OPI runtime support, OPI Developer's support, OPI 150 runtime support, and OPI-1x00 support. Install the general packages and device specific packages for any of the devices you have in your system. Set OPIHOME to the installation directory.

Include Files

opi.h and opi_c.h are installed in $(OPIHOME)/inc.

Link Directives

To build, you must link with libopi.so *(-lopi)*.

It is installed in $(OPIHOME)/lib *(-L$(OPIHOME)/lib)*.

OPI Dynamically Loadable Libraries

All .so's are installed in $(OPIHOME)/lib.

Installed Devices

$(OPIHOME)/config/opi.devices contains a list of installed devices. This list shows the mapping of device name to library name.

Compaq's Digital UNIX

Installation

OPI is distributed in a single tar file containing two or more directories and a README. The README will give detailed installation instructions, but the basic process is to *untar* the file and use the '*sysld*' command to install the DEC product kits. (For more information on 'sysld', run *man sysld*.)

Include Files

opi.h and opi_c.h are installed by the SDK development kit. They are installed to the default kit installation location and symlinked into the */usr/include* directories.

Link Directives

To build, you must link with libopi.so *(-lopi).*

It is installed in the standard kit installation directory and symlinked into the */usr/shlib* directory.

OPI Dynamically Loadable Libraries

All .so's are installed in the standard kit installation directory and symlinked into the */usr/shlib* directory.

Installed Devices

The *opi.devices* file contains a list of installed devices. This list shows the mapping of device name to library name. It is installed in the standard kit installation directory and symlinked into the */var/OSPOPI* directory.

Index

A

availableAttributes, 45, 46, 82, 83, 116, 121

B

bufferSize, 29, 62, 63, 114, 119

C

cOPIAttribute, 112, 113
cOPISystem, 112, 113
cOPIWindow, 112, 113

F

frame_count, 114, 119
frame_slices, 114, 119
fsize, 29, 62, 63, 114, 119

G

getAttribute, 82, 83, 116, 121
getAttributeInfo, 45, 82, 83, 116, 121
getAttributeType, 59, 60, 61, 114, 119
getAudioCapabilities, 79, 80, 116, 121
getBitRate, 76, 116, 120
getBlockRefreshPeriod, 74, 115, 120
getChannels, 64, 92, 114, 117, 119, 121
getCompType, 115, 120
getFrameRate, 66, 115, 119
getGettable, 59, 60, 114, 119
getInstanceByIndex, 53, 116, 121
getInstanceByName, 44, 79, 80, 116, 121
getMin, 45, 59, 60, 114, 119
getName, 53, 79, 81, 82, 84, 116, 121
getOverlayRate, 66, 115, 119
getPIP, 74, 115, 120
getPIPLocation, 74, 115, 120
getQuality, 76, 116, 120
getSampleBlockSize, 73, 78, 115, 116, 120
getSampleRate, 64, 92, 115, 117, 119, 121
getSettable, 59, 60, 114, 119
getType, 59, 61, 114, 119
getVideoCapabilities, 79, 81, 116, 121

I

instances, 42, 43, 44, 45, 46, 53, 79, 80, 99, 116, 121

O

opi_ai_getAttributeType, 114, 119
opi_ai_getGettable, 114, 119
opi_ai_getMin, 114, 119
opi_ai_getSettable, 114, 119
opi_ai_getType, 114, 119
opi_buffer_bufferSize, 114, 119
opi_buffer_create, 114, 119
opi_buffer_delete, 114, 119
opi_buffer_getData, 114, 119
opi_buffer_getFrameCount, 114, 119
opi_buffer_getFrameSlices, 114, 119
opi_buffer_getFsize, 114, 119
opi_buffer_getUserData, 114, 119
opi_buffer_setFrameCount, 114, 119
opi_buffer_setFrameSlices, 114, 119
opi_buffer_setFsize, 114, 119
opi_buffer_setSize, 114, 119
opi_buffer_setUserData, 114, 119
opi_capAudio_getChannels, 114, 119
opi_capAudio_getSampleRate, 115, 119
opi_capAudio_setChannels, 115, 119
opi_capAudio_setSampleRate, 115, 119
opi_capVideo_getFrameRate, 115, 119
opi_capVideo_getOverlayRate, 115, 119
opi_capVideo_setFrameRate, 115, 119
opi_capVideo_setOverlayRate, 115, 120
opi_cd_getCompType, 115, 120
opi_compAudio_getSampleBlockSize, 115, 120
opi_compH261_getBlockRefreshPeriod, 115, 120
opi_compH261_getPIP, 115, 120
opi_compH261_getPIPLocation, 115, 120
opi_compH261_setBlockRefreshPeriod, 115, 120
opi_compH261_setPIP, 115, 120
opi_compH261_setPIPLocation, 116, 120
opi_compVideo_getBitRate, 116, 120
opi_compVideo_getQuality, 116, 120
opi_compVideo_setBitRate, 116, 121
opi_compVideo_setQuality, 116, 121

opi_decompAudio_getSampleBlockSize, 116, 120
opi_device_delete, 116, 121
opi_device_getAudioCapabilities, 116, 121
opi_device_getInstanceByIndex, 116, 121
opi_device_getInstanceByName, 116, 121
opi_device_getInstances, 116, 121
opi_device_getName, 116, 121
opi_device_getVideoCapabilities, 116, 121
opi_di_getAttribute, 116, 121
opi_di_getAttributeInfo, 116, 121
opi_di_getAvailableAttributes, 116, 121
opi_di_getDevice, 116, 123
opi_di_getInFmt, 116, 122
opi_di_getName, 116, 121
opi_di_getOutFmt, 116, 122
opi_di_setAttribute, 116, 121
opi_fmt_aud_create, 116, 121
opi_fmt_aud_setCommon, 116, 121
opi_fmt_vid_create, 116, 121
opi_fmt_vid_setCommon, 116, 121
opi_nl_getItemName, 117, 121
opi_nl_getNumItems, 117, 121
opi_nl_nameNdx, 117, 121
opi_outAudio_getChannels, 117, 121
opi_outAudio_getSampleRate, 117, 121
opi_outAudio_setChannels, 117, 121
opi_outAudio_setSampleRate, 117, 121
opi_pipe_addStage, 117, 122
opi_pipe_create, 117, 122
opi_pipe_delete, 117, 121
opi_pipe_finishConstruction, 117, 122
opi_pipe_firstStage, 117, 122
opi_pipe_getDeviceFromHandle, 117, 122
opi_pipe_getInStream, 117, 122
opi_pipe_getOutStream, 117, 122
opi_pipe_mapToDevices, 117, 122
opi_pipe_pause, 117, 122
opi_pipe_resume, 117, 122
opi_pipe_setInStream, 117, 122
opi_pipe_setOutStream, 117, 122
opi_pipe_start, 117, 122
opi_pipe_stop, 117, 122
opi_stream_addData, 117, 122
opi_stream_create, 117, 122
opi_stream_delete, 118, 122
opi_stream_getBuffer, 118, 122
opi_stream_getData, 118, 122
opi_stream_isInternal, 118, 122
opi_stream_returnBuffer, 118, 122
opi_stream_setBuffer, 118, 122
opi_stream_setDataDoneCallback, 118, 122
opi_stream_setDataReadyCallback, 118, 122
opi_sys_create, 118, 123
opi_sys_delete, 118, 123
opi_sys_getControlInstance, 118, 123
opi_sys_getDeviceByName, 118, 123
opi_sys_getDevices, 118, 123
opi_sys_setDebugLevel, 118, 123
opi_win_create, 118, 123
opi_win_create1, 118, 123
opi_win_create2, 118, 123
opi_win_delete, 118, 123
opi_win_getHandle, 118, 123
opi_win_isInternal, 118, 123
opi_win_resize, 118, 123
OPIAttribute, 40, 45, 58, 59, 60, 83, 84, 112, 113
OPIAttributeInfo, 59, 112, 113, 114, 119
OPIBuffer, 29, 62, 100, 101, 112, 113, 114, 119
OPICaptureDevice, 64, 66, 112, 113
OPICaptureDeviceAudio, 64, 112, 113, 114, 115, 119
OPICaptureDeviceVideo, 51, 54, 66, 112, 113, 115, 119, 120
OPIColorcube, 68, 69, 72, 86, 108, 112, 113
OPICompressionDevice, 73, 76, 112, 113, 115, 120
OPICompressionDeviceAudio, 73, 112, 113, 115, 116, 120
OPICompressionDeviceH261, 51, 54, 74, 112, 113, 115, 116, 120
OPICompressionDeviceVideo, 74, 76, 112, 113, 116, 120, 121
OPIDecompressionDeviceAudio, 78, 112, 113
OPIDevice, 9, 10, 44, 46, 53, 79, 82, 84, 90, 104, 112, 113, 116, 121
OPIDeviceInstance, 9, 10, 11, 17, 40, 43, 44, 45, 46, 53, 80, 82, 96, 104, 112, 113, 116, 121
OPIFormat, 15, 83, 85, 86, 87, 88, 89, 95, 96, 112, 113, 116, 121
OPIFormatAudio, 15, 34, 87, 112, 113, 116, 121
OPIFormatVideo, 15, 16, 20, 28, 48, 88, 89, 112, 113, 116, 121
OPINameList, 46, 80, 83, 90, 104, 112, 113, 117, 121
OPIOutputDeviceAudio, 92, 112, 113, 117, 121
OPIPipeline, 9, 10, 11, 13, 15, 20, 24, 25, 40, 94, 95, 112, 113, 117, 121, 122
OPIStage, 9, 17, 21, 50, 54, 95, 96, 112, 113
OPIStream, 24, 25, 28, 62, 96, 97, 99, 100, 112, 113, 117, 118, 122
OPISystem, 9, 13, 14, 15, 20, 40, 43, 82, 95, 103, 112, 113, 118, 123
OPIWindow, 27, 28, 41, 106, 107, 112, 113, 117, 118, 123

S

setBitRate, 51, 54, 76, 77, 116, 121
setBlockRefreshPeriod, 51, 54, 74, 75, 115, 120
setChannels, 64, 92, 93, 115, 117, 119, 121
setFrameRate, 51, 54, 66, 67, 115, 119
setOverlayRate, 51, 54, 66, 67, 115, 120
setPIP, 74, 75, 115, 120
setPIPLocation, 74, 75, 116, 120
setQuality, 51, 54, 76, 77, 116, 121
setSampleRate, 64, 65, 92, 93, 115, 117, 119, 121
setSize, 62, 63, 114, 119

U

userData, 29, 62, 101, 102, 114, 119