California AHMCT Program
University of California Davis
California Department of Transportation

BRIDGE INSPECTOR IMAGE ENHANCEMENT GROUND-BASED STATION (IGBS)

Odetics
Advanced Systems Development
Communications Division
1585 S. Manchester Avenue
Anaheim, California 92802

AHMCT Research Report

UCD-ARR-95-05-26-01

Final Report of Contract
54Q168MOU#92-11

October 10, 1995

ABSTRACT

The Image Enhancement Ground-Based System (IEGBS) is a system of special hardware and software developed especially to view, capture, and enhance images from a color video camera performing a bridge inspection. The lightweight camera can be mounted on a robotic arm or, as it is currently configured, on a remote controlled aerial vehicle. This will enable the camera to get under the bridge to view the areas that need inspecting. The digital image processing capabilities of the IEGBS brings a significant, new technology to the bridge inspection process that should make it faster, safer, and more thorough.

EXECUTIVE SUMMARY

Maintenance of the transportation infrastructure is important to the safety of the people using the nation's roads and bridges as well as to the efficient transportation of goods. Proper maintenance begins with proper inspection. The current inspection process for bridges is performed by suspending the inspection crew under a bridge to first perform a visual inspection, and then, if required, perform non-destructive tests on small but critical parts of the bridge. This inspection process is time consuming, moderately dangerous, and performed under difficult conditions. Coupled with poor lighting conditions and inaccessibility of some locations under the bridge, the current bridge inspection process can result in incomplete inspections.

Although the current inspection process relies on an initial visual inspection, photographic or video records of the inspection are not part of the standard procedure. This kind of visual documentation of the inspection serves not only to record that a proper visual inspection was performed but also to preserve a visual documentary of a bridge over time so that trends in the aging and maintenance of the bridge can be established.

The Image Enhancement Ground-Based Station (IEGBS) projects was undertaken to address some of these shortcomings with the current bridge inspection process.

The IEGBS is a system of special hardware and software developed especially to view, capture, and enhance images from a color video camera performing a bridge inspection. The lightweight camera can be mounted on a robotic arm or, as it is currently configured, on a remote controlled aerial vehicle. This will enable the camera to get under the bridge to view the areas that need inspecting. The fundamental image processing capabilities of capture, compression, storage and retrieval are provided along with a powerful hardware-based image processor, the IDS system. IDS has some benefit under backlight conditions where foreground objects are darkened by enhancing edges and contrast.

Table of Contents

List of Figures

DISCLOSURE

The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Mechanical Engineering, Aeronautical and Materials at the University of California, Davis and the Division of New Technology and Materials Research at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, state and federal governments and universities.

No outside contracts or subcontracts were used to prepare this report.

The contents of this report reflect the views of the author(s) who is(are) responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation.

CHAPTER 1

INTRODUCTION

Maintenance of the transportation infrastructure is important to the safety of the people using the Nation's roads and bridges as well as to the efficient transportation of goods. Proper maintenance begins with proper inspection. The current inspection process for bridges is performed by suspending the inspection crew under a bridge to first perform a visual inspection, and then, if required, perform non-destructive tests on small but critical parts of the bridge. This inspection process is time consuming, moderately dangerous, and performed under difficult conditions. Coupled with poor lighting conditions and inaccessibility of some locations under the bridge, the current bridge inspection process can result in incomplete inspections.

Although the current inspection process relies on an initial visual inspection, photographic or video records of the inspection are not part of the standard procedure. This kind of visual documentation of the inspection serves not only to record that a proper visual inspection was performed but also to preserve a visual documentary of a bridge over time so that trends in the aging and maintenance of the bridge can be established.

The Image Enhancement Ground-Based Station (IEGBS) projects was undertaken to address some of these shortcomings with the current bridge inspection process.

The IEGBS is a system of special hardware and software developed especially to view, capture, and enhance images from a color video camera performing a bridge inspection. The lightweight camera can be mounted on a robotic arm or, as it is currently configured, on a remote controlled aerial vehicle. This will enable the camera to get under the bridge to view the areas that need inspecting. As the video from the camera is being fed into the IEGBS, the inspector will be able to view the live image on the IEGBS color monitor. This is intended to give the inspector the same visual inspection capabilities as if he was suspended under the bridge. Even in this mode, the inspector has the ability to see areas of concern with much higher detail because he can elect to use the 10:1 zoom lens on the camera to get a close-up look at the area - something he would have to use binoculars for if he was actually suspended under the bridge. The IEGBS, however, can do much more with the video from the camera than simply displaying it on the monitor. Much like a camera, the IEGBS can take a still image of what the camera is currently viewing. Unlike a photographic camera which stores images on film, the IEGBS stores pictures on the computer disk in a digital image. These images can then be instantly recalled from the computer disk to be viewed on the monitor so you can skip the delays and guess-work associated developing photographic film. Because the images are stored in digital form, they can be easily copied, transmitted to another location using a modem, and/or further processed to enhance some parts of the image that are hard to discern. The image processing can be performed using the general purpose computer or the specialized Intensity Dependent Summation (IDS) computer. The IDS computer is a specially designed computer for enhancing edges and contrast in real time - as you watch it. The IEGBS also has a computer controlled video tape recorder that can be used

to record the entire video inspection on tape along with the inspector's audio annotation. Once it is on tape, the user can play it back as usual as well as grab individual images from the tape for storage on the computer disk.

The IEGBS can provide the inspection crew the benefits of remote video inspection, image storage and review, image enhancement to aid in finding cracks, and safer working conditions.

In summary, by properly operating the IEGBS, the user will be able to:

- Freeze/Unfreeze the video from the camera
- Process the live video using the IDS Image Processor
- Take a digital "snap shot" of the video image
- View/Process digital snap shots
- Record the video using a Hi8 video tape recorder
- View/Process video previously recorded on video tape
- Digitally zoom in on part of the image

CHAPTER 2

TECHNICAL DISCUSSION

Project Objectives

The objective of the Image Enhancement Ground-Based System (IEGBS) is to provide a system which will acquire, display, enhance, store, and retrieve video imagery related to the inspection of bridges and other similar structures. Specifically, the primary IEGBS objective is to acquire full-color images from a video camera and enhance the images so that cracks and other structural degradations are made more apparent to the inspector viewing the images. The IEGBS shall also be capable of storing images for later retrieval and viewing.

Technical Approach

In order to meet the stated objectives, Odetics selected a technical approach that combines its expertise in digital image processing with its experience in user-friendly, graphical user interfaces. The result is a PC-based system that can capture full-color digital images from the video source, enhance them using software on the PC or using special image processing hardware, and then display them to the user. All the while, the user is presented with a Windows[1] graphical user interface that simplifies learning the system operation. Figure 1 is a photograph of the IEGBS with the inspection camera mounted on a tripod. A closer examination of each of the major elements of the system show the power of this approach.

Digital Image Processing

The fundamental feature of the IEGBS is that it is a digital image processing system. This fundamental feature dictates much of the architecture of the IEGBS.

Image processing can be conceptually thought of as a set of operations that use images as the operands or data. While there exist optical image "processors" that use lenses and filters to perform operations and analog video processors, the image processing in the context of this document will be digital image processing performed using a digital computer or other kind of digital hardware. Just like word processors manipulate and format the elements of a text document, i.e. letters, words, paragraphs, etc., a digital image processor manipulates and operates on the elements of a digital image, i.e. pixels, frames, etc. The first step, therefore is to get an image that can be seen with a video camera into the digital elements that can be operated upon. To see how this is accomplished, a short digression is warranted to understand about video cameras and video signals.

---

[1] Windows™ Microsoft Corporation.

Figure 1 Photo of IEGBS

Video Sources and Signals.

The video signal from a video camera or a video tape recorder is a standardized format for encoding the color and intensity of an image. It also includes synchronization signals to break the color and intensity signal into lines, fields, and frames. These synchronization signals are required to display and capture the color and intensity information. A frame is a full resolution image that is normally displayed on a video monitor. A field is a subdivision of a frame - it takes two fields to make a frame. The video camera produces 30 frames of video per second. In order to transform the analog video frames into digital images, the IEGBS uses a special board called a "frame grabber."

Video Frame Grabber / Digitizer.

In essence, the frame grabber takes one frame of the video signal and digitizes it so that it looks like a big array of numbers (640 columns x 480 rows). Each one of these numbers is a picture element ("pixel"). Once the video image is in the form of an array of pixels, then the computer can be used to operate/process these numbers to either save them to the hard disk as a digital snap shot, or process the pixels to make a new image.

The video frame grabber selected for the IEGBS is the DVA-4000[2]. The DVA-4000 is a high-speed board designed to synchronize with the incoming video signal and digitize the color and intensity information at the full 30 frames per second. The DVA-4000 is equipped to handle both NTSC video as well as S-Video signals. Since S-Video has better color quality and spatial frequency resolution, it is used as the video input to the DVA-4000. Besides performing the digitizing function, it also performs the inverse function, namely, taking a digitized image and outputting a video signal for displaying on the Host Computer color monitor. The DVA-4000 performs some image processing independent of the Host Computer CPU. Some of the specific functions used in the IEGBS include:

- Scaling the size and aspect ratio of the image.
- Overlaying video with graphics.
- Overlaying graphics with video.
- Adjusting color, intensity, and contrast of the digitized image.

The DVA-4000 works in conjunction with the Host Computer to provide for storing and retrieving the digitized video frames on the hard disk drive. Direct program control of the DVA-4000 from the Host Computer program is accomplished using the MIC-II control and driver software. By controlling the DVA-4000 from the Host Computer program, the IEGBS becomes an integrated software and hardware system that is much easier to use than if the frame grabber was controlled using a separate program.

---

[2] DVA-4000™ VideoLogic

The image processing capabilities of the DVA-4000 are limited mainly to those functions that capture and display digital images - this is the function of the board. To perform image processing functions that enhance the detection of cracks and other structural flaws, special image processing hardware and/or software must be used. The IEGBS provides two image processing solutions: high-speed, specialized image processing hardware, and general purpose, flexible image processing software.

Intensity Dependent Summation (IDS) Image Processor.

The time it takes to perform some image processing functions is often greatly reduced if specialized image processing hardware is used. The driving reasons for this are that the digital images require quite a lot of memory to hold them, and the image processing operations often must be repeated for every pixel in the image. For example, a full color, full resolution image is an array of 640 x 480 pixels with each pixel being 3 bytes long. This means that a full color digital image is about 1 megabyte in size. Even for relatively simple image processing functions, the number of multiplies and additions can easily exceed 5 million. A complex operation like IDS can exceed 64 million operations per image. While image processing functions performed on a general purpose computer are very flexible since they are performed in software, a typical PC would take several seconds to perform this number of operations even if it had enough memory to hold the input images and result image completely in memory. Specialized image processing hardware can be configured with large amounts of high-speed memory to hold the images, and specialized, high-speed computational hardware to perform a relatively small set of operations very quickly. The result is real-time (30 frames per second) image processing. Another way to judge the speed of the IDS hardware is that the number of operations performed on the IDS hardware is of the order of 2000 million operations per second. The resulting IDS processed images are black & white (gray scale) images that can reveal some image features that may be hard to see under some lighting conditions.

The IDS adaptive image processor is based on the human visual system. It performs a special adaptive filter algorithm with a 32 x 32 pixel convolver on a 512 x 512 pixel image at real-time video rate. The performance of this system, which is automatically adaptive to the individual pixel intensity, is much better than standard, nonadaptive image convolvers. One of the useful effects this process has on the dynamic range of images is that it produces localized automatic gain control so that the gain in bright regions of the image is reduced by increasing spatial resolution relative to dark regions. Range compression preserves contrast optimally in very dark and very bright regions of the image with this IDS adaptive imager. The IDS process provides edge enhancement that is virtually independent of scene illumination and is very robust to noise at low intensity levels. Odetics has rigorously tested the IDS algorithm on laboratory images and outdoor, real-world scenes and conducted intensive research to implement the IDS model in computer hardware for real-time processing. The result of this research is a custom gate array digital processor and eight associated PC boards which can process a 512 x 512 pixel image at video rate.

Previously reported work on the IDS model (Cornsweet 1985) has shown that its application as an image processing procedure produces three classes of results that are of significant interest in machine vision systems. First, for images of low intensity, IDS provides an optimal tradeoff between smoothing photon noise and spatial resolution, maintaining constant signal-to-nose ratio at all mean intensity levels. Second, after IDS processing, the extraction of edges from an image requires only a zero-crossing detector, thus allowing IDS processing to serve as a preprocessor for many feature extraction algorithms. Third, the output corresponding to each edge step in an IDS processed image is uniquely determined by the ratio of the image intensities on the two sides of the edge and therefore preserves the ratio of the reflectance of objects and background, independent of changes in the mean intensity (uniformity of illumination principle), falling on them. The second and third results also indicate that the use of IDS in an early processing stage at or near the image plane can provide very significant bandwidth reduction in the transmission of information from a remote sensor.

When IDS is used to process an image, there is an automatic tradeoff of resolution. Noise averaging occurs locally within a given image so that an object that is in deep shadow will be just as detectable as a similar one in full illumination, given that the objects are large enough. That is to say that IDS performs automatic gain control - reducing the gain when the image gets brighter. However, IDS performs automatic gain control in a way that is fundamentally different from that performed by ordinary gain control mechanisms in two ways. Standard gain control mechanisms reduce the light (with an iris) or reduce the gain of an amplifier. The IDS operation is local so that the gain in a bright region of the image can be reduced at the same time as the gain in a dark region is increased. Further, there is no loss of light or signal in the IDS method. In effect, every light photon is always used; high photon fluxes are being used to improve local resolution, thus preserving an optimal signal-to-noise ratio.

IDS Hardware and Software Architecture.

The IDS adaptive image processing system is implemented as a separate system that is independent of the IEGBS Host Computer. It is based on the DataCube[3] pipeline architecture. The IDS system consists of 8 VME boards each with two Odetics custom gate array digital processors. These boards are supported by an Odetics custom histogram board, a DataCube processor board (MAXSP), two DataCube digital image memory boards (FRAMESTORE), and a DataCube video digitizer (DIGIMAX). It also has its own control processor using a 68EC030-based VME CPU board from Ironics[4] (IV-3234) with an Ethernet interface daughter board (IV-SETI). Since the IDS system processor boards are based on a totally different architecture and form factor (VME) than the ISA-bus boards that plug into a PC, there was no direct way of integrating the IDS system with the Host Computer short of a new design. Instead of a tightly coupled, direct connection, the Host Computer and the IDS system are loosely coupled using the

[3]DataCube, Inc. Peabody, MA
[4]Ironics Inc. Ithaca, NY

Ethernet port connecting the two computers. Using this Ethernet connection, the Host computer can:

- send commands to the IDS control computer,
- receive status messages from the IDS control computer,
- send images to be processed to the IDS control computer, and
- receive processed images from the IDS control computer.

Since the IDS control computer is independent of the Host Computer, custom software was developed for the IDS system to perform the IDS image processing functions as well as handle the Ethernet messages. The IDS control processor has its own real-time operating system called VxWorks[5] which resides along with the control software in an EPROM on the IV-3234 board. VxWorks is a UNIX-like operating system with extensions for real-time operation. Software for the target system (the IV-3234) is developed on a host system (Sun 3). This arrangement makes it possible to download the VxWorks operating system and code via the Ethernet during the development cycle, and then "burn" an EPROM with the final, minimized operating system and code. This arrangement permits the use of the full operating system and debugging features during development and then burn a very compact operating system that can easily fit into a 256Kb EPROM. As configured, the IEGBS "boots" from the EPROM when the power is turned on and starts all the necessary tasks to perform command interpretation and image processing functions. When the Host Computer starts up, it will look to see if the IDS computer has booted up successfully. If it does not detect the IDS computer, then an error message is issued so that the operator can take the action to reset the IDS image processor using the front panel switch.

See Appendix A for the complete source code for the IDS Control Computer Software.

[5]Wind River Systems, Inc. Emeryville, CA

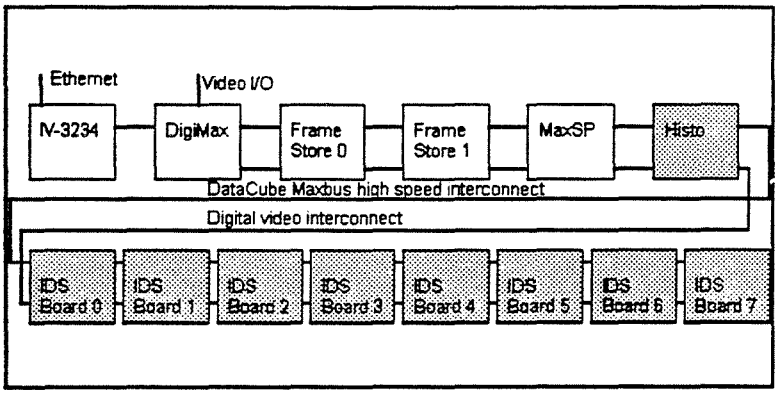


Figure 2. Block Diagram of IDS System

The shaded blocks represent the custom boards to implement the IDS image processing function. The boards communicate using the DataCube Maxbus high speed interconnection bus and the VME bus. All of the boards' control registers are memory mapped into the address space of the IV-3234 control computer. Whenever the IV-3234 control program wants to change the operation of the IDS system, it does so by setting the control registers on the boards. Once the control registers are set, the boards will perform the operation at 30 video frames per second. The input to the DigiMax comes directly from the inspection camera. The output of the DigiMax goes to the 8mm VCR where it can be recorded and/or switched to the monitor for viewing.

Image Processing Software.

Besides the IDS image processor, the IEGBS can use software image processing techniques. While the IDS image processor is very fast, it is also very specialized. Using the Host Computer and image processing software provides flexibility at the price of performance. Because the data rates for digital data are very high, the Host Computer and image processing software cannot be expected to perform in real time. Instead, the image processing software is designed to operate on previously saved digital images.

Digital Zoom.

The IEGBS includes one software image processing function that provides a "digital zoom" on an area of a previously saved image. Digital zoom is accomplished by expanding a pixel into four adjacent pixels, thus making the image four times bigger. However, unlike optical zooming using a lens, digital zoom causes the image quality to be reduced as the image is zoomed. Although the digital zoom can be selected on a previously zoomed image, the practical limit for this technique is a 4x zoom.

Images in Reports.

Digital images are stored in a number of industry-wide formats as well as a number of proprietary formats. Some of these formats go by the names GIF, TIFF, BMP, DIB, Targa, DVA, etc. Because of the proliferation of these formats, image file format conversion software has become a necessary tool for any digital image processing system. This file format conversion becomes necessary when an application cannot read the format that an image is currently stored in. For example, the IEGBS can only store images in two different formats, DIB or DVA. If a report generation software package cannot read either of these formats, then the images of interest must be run through a format converter package like HiJaak Pro[6] which is included with IEGBS.

Image Cataloging.

As the number of digital images on the disk grows, it becomes more difficult to later come back and find the images of interest. The inherent problem is that the digital images are stored by name on the Host Computer disk while users cannot be expected to explicitly describe each image they capture. The problem is exacerbated by the limited file naming capabilities of the Windows 3.1 environment. HiJaak Pro permits the user to set up a "thumbnail" browser of the images that are stored on disk. Essentially, these are reduced size images that permit the user to quickly scan through hundreds of images. However, because the images are reduced in size, it makes it more difficult to identify the subject of the image. This is particularly true for relatively nondescript images of bridge structures.

Video Subsystem

The video subsystem of the IEGBS is of primary importance since the digital image processing subsystem depends on good quality source video. The video subsystem consists of the inspection video camera, and a high quality video recorder that also performs the function of a video switcher.

[6]Inset Systems, Brookfield, CT

Video Camera.

The video camera selected for the IEGBS is the Sony CCD-TR101 handicam. Using the CCD-TR101 handicam instead of an ordinary video camera has several advantages namely:

- motorized zoom lens,
- auto-iris lens,
- auto-focus lens,
- image stabilization circuit, and
- built in remote control capability.

The remote control capability using the built in LANC protocol permits controlling the camera from the IEGBS. The IEGBS issues VISCA commands from one of its serial data ports which get translated to LANC commands by a Sony CI-1000 "VBox." This arrangement is shown schematically below.

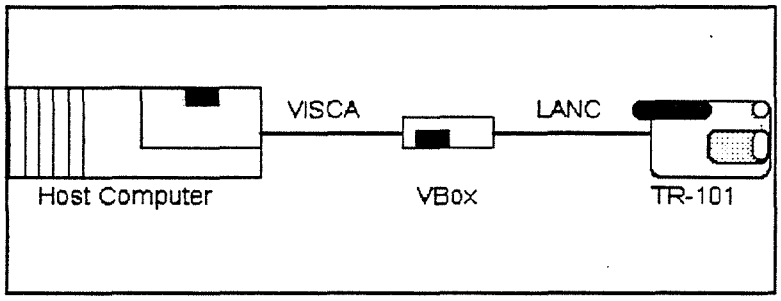


Figure 3. Block Diagram of Video System

In the current implementation of the IEGBS, the handicam is mounted on the Moller aerial vehicle and the LANC command cable is part of the aerial vehicle control and power umbilical cable. While this configuration is workable, the preferred method would be to multiplex the VISCA[7] commands onto the aerial vehicle control fiber optic and have the VBox on the aerial vehicle. This configuration would avoid the problems of transmitting the LANC signal up a noisy, long cable bundle.

Remote Positioning of Camera.

For a remote camera to be useful as an inspection camera, remote positioning of the camera is

[7] VISCA™ is Sony's Video System Control Architecture, a set of platform-independent command codes that can be incorporated into software packages to allow synchronized control of multiple video devices.

essential. As it is currently implemented, the handicam is mounted on a tilt-only (elevation) platform mounted on the side of the aerial vehicle. Side mounting the camera, as opposed to top mounting, provides the ability to look down as well as up so the field of view and the inspection vantage points are greatly increased. There is no immediate need for a pan (azimuth) control of the camera since the vehicle yaw provides the same kind of result albeit with less angular control. Remote control of the tilt is accomplished by controlling a model servo motor attached to the tilt platform mounted on the aerial vehicle. By using a joystick at the IEGBS, the user varies the duty cycle of the control pulses to the servo motor. The motion is not precise, but is simple and adequate for remote inspection. Because the tilt servo motor control signals are sent up an already unwieldy aerial vehicle control and power umbilical cable, a better method for controlling the tilt platform would be to send tilt commands up to the local processor on the aerial vehicle via a fiber optic link and have it in turn command the servo motor.

Video Recorder.

Since remote bridge inspection using a camera mounted on an aerial vehicle or even an arm is a dynamic process, providing a video recorder to record the entire inspection process will permit inspectors to review the inspection at a later date without the time constraints of trying to take manual notes or digital snapshots as the inspection is in progress. The video cassette recorder (VCR) chosen for the IEGBS is the Sony CVD-1000[8]. This VCR uses a Hi-8 video tape which is smaller and can record at resolution higher than the more common VHS tape. High resolution recording was a driving factor in its selection since the video is going to be used in an inspection process where fine spatial details are important. The CVD-1000 has another advantage in that it is specifically designed to be a computer peripheral. In other words, all the functions (and more) that can be performed using the hidden control buttons, can be performed using commands from the Host Computer (See section on User Interface). The commands are in the VISCA command protocol, and are issued by the Host Computer from one of its serial ports. The CVD-1000 also functions as a video switcher. It has one S-Video and two composite video input and output ports that can be selected using the appropriate VISCA commands. Having a video switcher enables the ability to switch between the video coming directly from the camera or the processed video coming from the IDS processor. The CVD-1000 can, of course, act as the video source so that previously recorded inspection videos can be viewed on the IEGBS monitor and processed as if it was coming from the camera. The Hi-8 format also permits two independent audio tracks. In this case, the IEGBS uses the PCM audio track as an audio narration track so that the user can add narration about the video he is watching while the inspection is occurring. Later, when the tape is being reviewed, the audio annotation will provide valuable insight about the inspection process and the images that are recorded. The CVD-1000 will automatically put a time and date stamp on the video so there is no guess-work about when the video was collected.

[8]Sony Computer Peripheral Products Company, San Jose, CA

Host Computer

The Host Computer, as the name implies, plays host to the user interface and control software as well as some special hardware for image capture and storage. The user interface and control software is covered in its own section. This section describes the hardware of the Host Computer.

Ruggedized PC.

The Host Computer is a ruggedized PC[9] that is compatible with desktop PC's, but has been enhanced by the addition of a heavy-gauge steel chassis that is rack mountable and additional cooling capacity. The Host Computer is based on an Intel 486-66 processor with a VL local video bus and 16 megabytes of memory. The speed of the processor and the additional memory improves the processing time of image processing software routines. Since digitized images can use from 300 kilobytes to 1000 kilobytes of disk memory when they are saved, a large, fast disk is essential. The host computer is equipped with a 540 megabyte disk and its associated SCSI-2 disk controller for local storage of digital inspection images.

Special Hardware.

In addition to the above standard hardware and the DVA-4000 frame grabber that is described in the section titled Video Frame Grabber / Digitizer, the Host Computer has some additional hardware that makes it a more versatile host for the IEGBS. The Host Computer is equipped with a CD-ROM drive that provides a very compact and rugged means for long-term archiving of digital inspection images. Each CD is capable of storing over 1500 images, but the images have to be put on the CD by a service bureau, i.e. the IEGBS can only read CD's it cannot write them. In order to provide a writeable and removable media, the IEGBS is also equipped with a Bernoulli[10] removable hard disk. While not quite as fast as the internal hard disk, nor as rugged as a CD-ROM, each removable disk cartridge can hold up to 450 compressed images. These disk cartridges can be used for long-term archiving of images or as a high capacity media for transferring digital images to a CD-ROM mastering service bureau.

User Interface and Control Software

Beyond the hardware, the most important function that the Host Computer performs is to integrate all the different parts of the IEGBS so that they can be controlled from the User Interface. The User Interface presents a simple, easy to understand front end that hides the details of the underlying control software.

---

[9] Chassis - 7408-23V, Motherboard - 4862MB66PIVL. Industrial Computer Source, San Diego CA
[10] Iomega Corp., Roy, Utah

Windows 3.1 Application.

The IEGBS User Interface is designed to comply with the common Windows 3.1 graphical user interface. The User Interface is divided into two main areas: the control button bar and the image display window. The control button bar holds a group of Windows buttons that are used to perform the functions on the IEGBS. The keyboard is only used to enter textual information, not activate commands. Using buttons with icons and names instead of the more cryptic function buttons on the keyboard should make it easier to learn and use. The image display window, as the name implies, shows the live video from the camera as well as digital still images that were saved on disk.

The User Interface and Control software was written using Visual C++[11] and supported by graphic controls from the Visual Control Pack[12]. Visual C++ extends the C++ language with the Microsoft Foundation Classes that makes developing Windows applications conform to Object-Oriented Design and Development (OODD) practices. OODD makes it possible to later reuse and/or modify the User Interface and Control software much easier. Visual C++ also provides an application development environment that helps to keep track of the many elements that go into a Windows Application.

See Appendix B for full User Manual that shows and describes the icons and their functions.

See Appendix C for the complete source code for the User Interface and Control Software.

Enclosure

In order for the IEGBS to be used in the field for bridge inspection, all the components must be housed in a rugged enclosure. A standard 19" rack enclosure was chosen because of its wide-spread availability and inherent ruggedness. The IEGBS components are divided between two enclosures for the following reasons:

- reduces component weight and size,
- isolates the heat generated by the video monitor to its own enclosure,
- modules can be stacked as shown in Figure 1, or placed side-by-side, and
- breaks the IEGBS into smaller field replaceable units for better system up-time and spare parts allocation.

[11] Visual C++ ™ Microsoft Corporation.
[12] Microsoft Corporation.

All the components in the processor enclosure are mounted on rails to facilitate the accessibility of the components once they are mounted in the enclosure. Figure 4 labels the components in each of the enclosures.

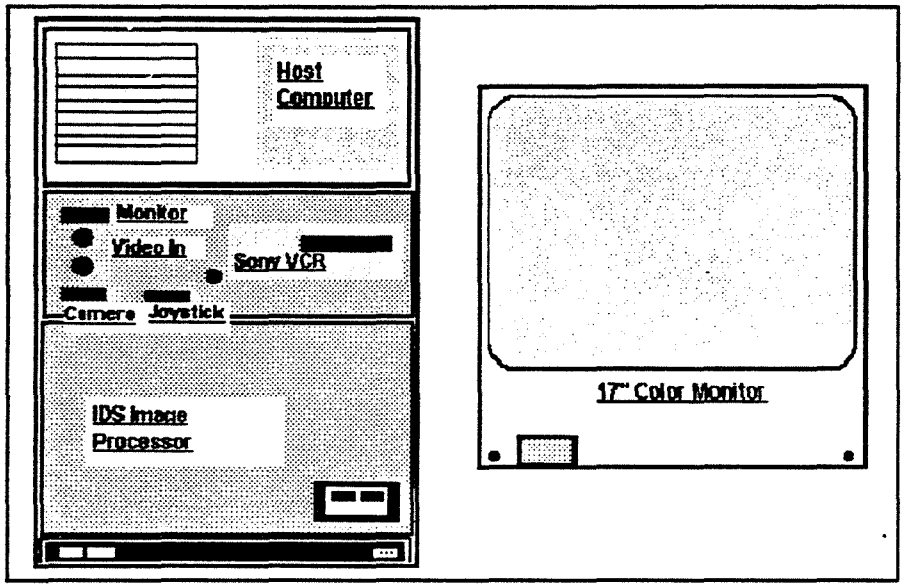


Figure 4. IEGBS Enclosure

Bridge Inspection Images

In order to access the suitability of the IEGBS to the bridge inspection process, video tapes were made using the Sony CCD TR-101 at two bridges during the inspection process. These video tapes were then used as the source video in the IEGBS CVD-1000 video tape recorder. In order to compare the IDS processed images with normal video images, several images were captured from the video tapes and processed using the IDS image processor in the IEGBS. The comparison images are divided into four sets. Note: The normal video images at the top of the comparison image sets are normally viewed as color images, while the IDS processed images are normally black and white (grayscale). For the purposes of this comparison, both images are printed as grayscale images since the comparison is with an IDS image which is normally grayscale.

Set 1 - Vertical column bolted to concrete pad.

Image 1N is a normal image recorded using the Sony CCD TR-101. This image was selected because it illustrates the effect of a bright background on the objects in the foreground - the foreground objects are dark. The problem with the dark foreground objects is not with the sensitivity of the camera, rather it is the effect the bright background has on the camera auto-iris. Without the bright background, the camera iris will open up and the dark foreground objects will appear adequately illuminated. While the inspection is in progress, the operator can manually

open the iris to permit better definition of the foreground objects. However, from a tape, this option is not possible. One way to enhance objects that appear too dark is to perform image processing to enhance hidden edges. In this case, the image is processed using the IDS image processor that is part of the IEGBS. The result of the processing is shown in Image 1P. As with all IDS processed images, the edges of objects that are well illuminated are thin and sharp while the edges of objects that are poorly illuminated are wide and blurry. This is a feature of the IDS adaptive edge enhancing kernel. In this case, Image 1P shows a better definition of the disk on the column as well as some of the bolt heads on and near the disk. The added noise due to enhancement, however, detracts from the appearance of the image. IDS processed images have the background intensity removed which causes the image to be a medium gray with saturated black and white edges.

Set 2 - Horizontal beam with bright background.

Image 2N has similar qualities as Image 1N - dark foreground object with a bright background. In this case, the inspection camera is at the same elevation as the structural members under the bridge deck and the sunlit background is present in the scene. As in Image 1N, the bright background closes the camera auto-iris which makes the structural beams too dark. Image 2P shows the result after processing with the IDS image processor. Some of the rivet heads along the bottom of the beam web are more apparent than in the original image. The details in the bright background get "washed out" because IDS eliminates the background intensity and replaces it with a medium gray.

Set 3 - Crack in structural member.

Unlike the previous two Image Sets, Set 3 has a plainly visible crack in the structural member under observation with no backlighting problems. Under these conditions, IDS image processing does not add any benefit since the crack is clearly visible in the normal video image. Image 3N shows the normal video image, image 3P shows the IDS processed image.

Image Comparison Set 1.

Vertical column bolted to concrete pad.

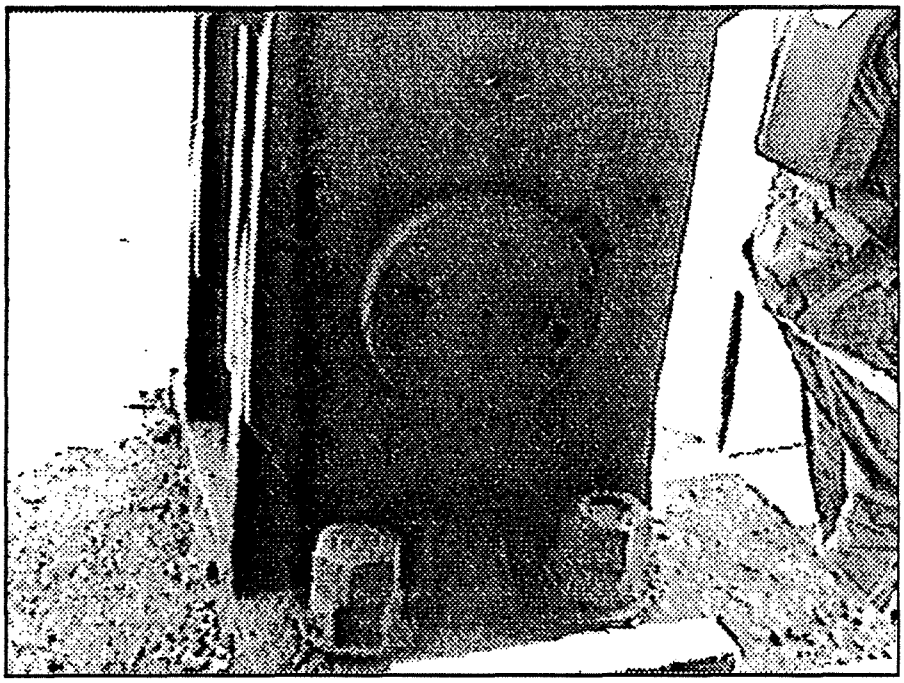


Image 1N. Normal Video Image

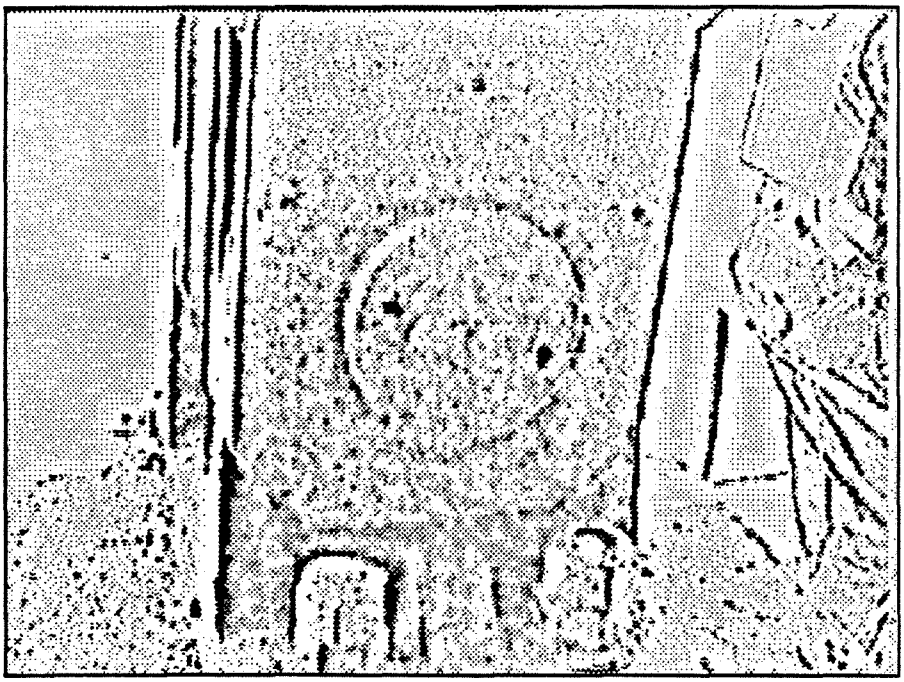


Image 1P. IDS Processed Image

Image Comparison Set 2.

Horizontal beam with bright background.

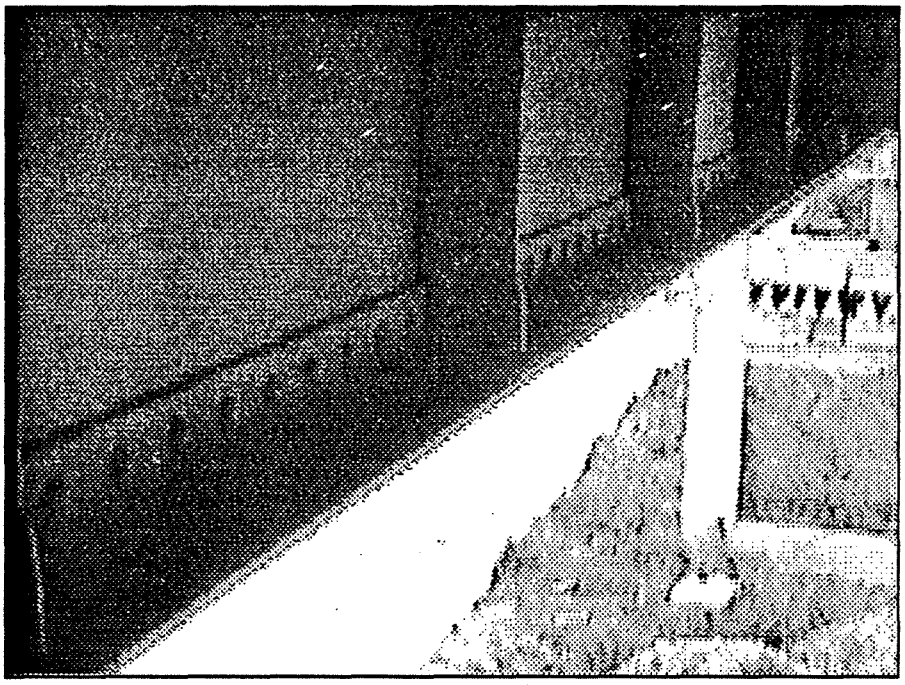


Image 2N. Normal Video Image

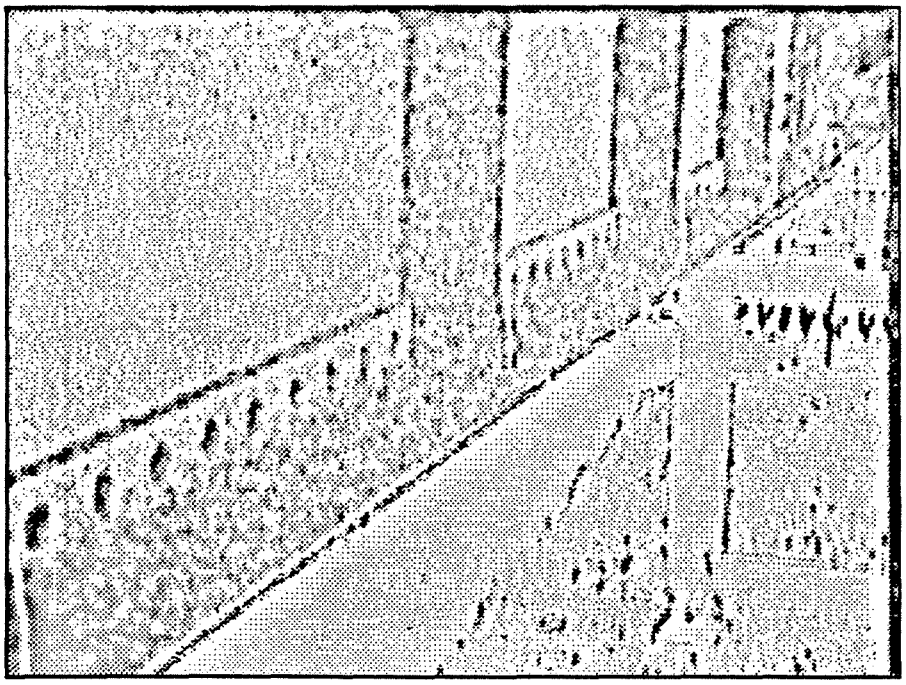


Image 2P. IDS Processed Image

Image Comparison Set 3.

Crack in structural member.

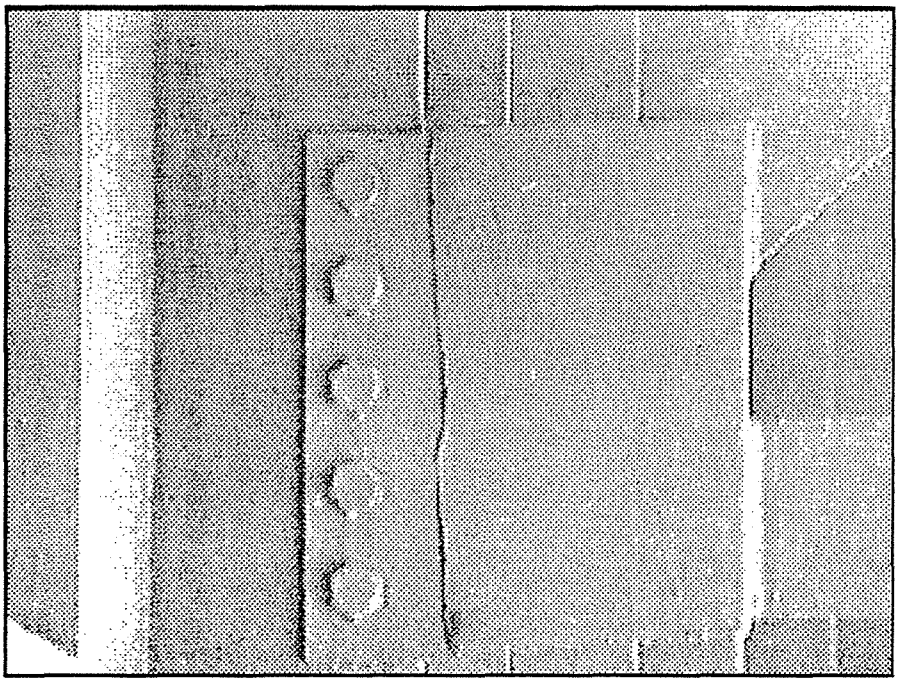


Image 3N. Normal Video Image

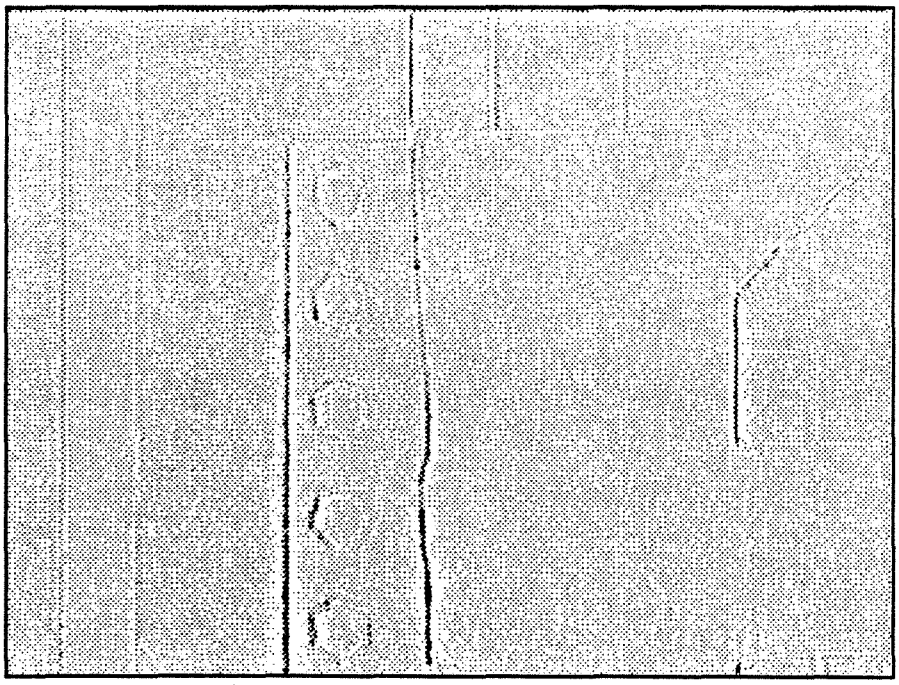


Image 3P. IDS Processed Image

Set 4 - Closeup of crack in structural member.

This set of images is a closeup of the same crack presented in Image Set 3. Again, because there is no backlight, the normal image has a well balanced illumination, the crack is plainly visible in the normal video image. While this set does not show any benefit to the IDS processing, it does illustrate how even a 10:1 remote controlled zoom lens on the camera can give the operator an extreme closeup of structural members under inspection.

Set 5 - Missing nut from pier bolt.

While the emphasis of the IEGBS has been on its IDS image processing capability to enhance cracks, its digital image documentation capability also warrants some merit. In this image set, the inspection video recorded a missing nut from a bolt that holds the pier cap in place. The Image 5N can be used as part of an inspection report that clearly illustrates the missing part. At the next inspection of the bridge, this image can be quickly recalled on the IEGBS monitor so that the inspector can quickly determine that corrective action has been performed. Image 5P shows how a 4x digital zoom can magnify an image. While optical zoom using the camera lens provides a higher quality image, digital zoom can be performed after the fact on an image that has already been saved to disk or a video tape.

Image Comparison Set 4.

Closeup of crack in structural member.

Image 4N. Normal Video Image

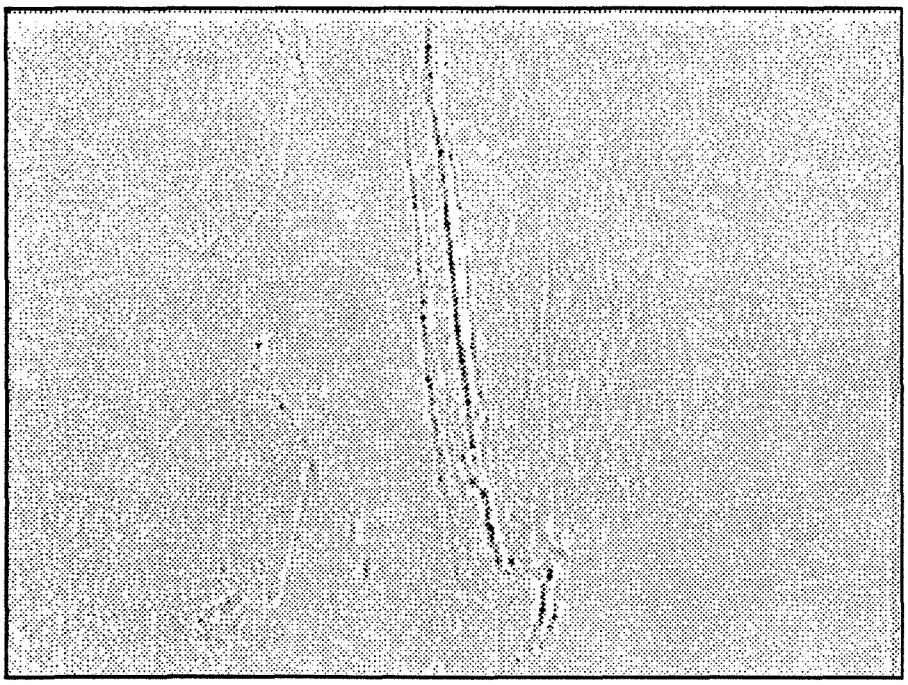


Image 4P. Processed Video Image

Image Set 5.

Missing nut from pier bolt.

Image 5N. Normal Video Image

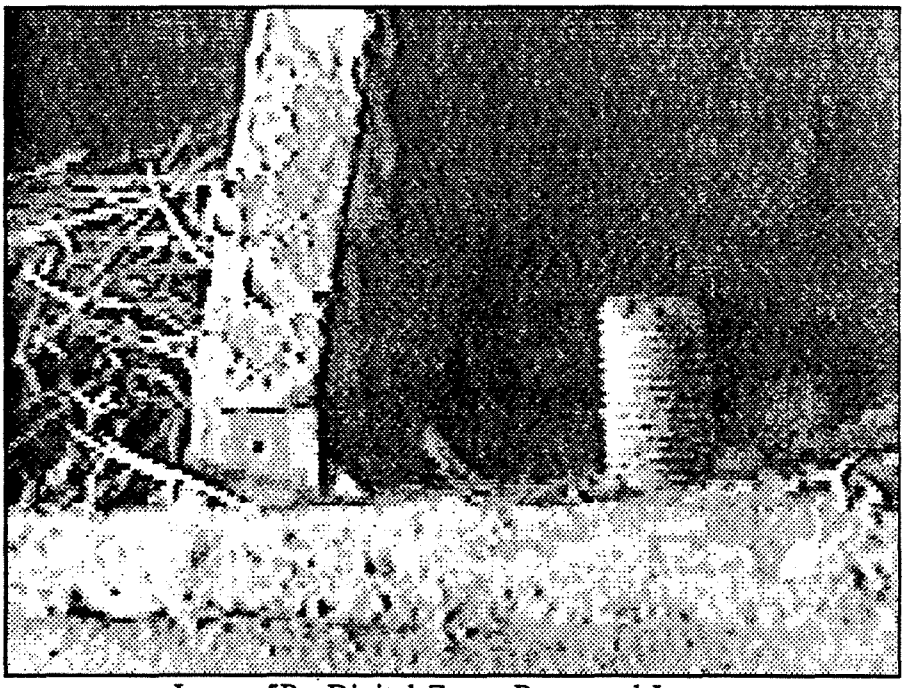


Image 5P. Digital Zoom Processed Image

CHAPTER 3

CONCLUSIONS AND RECOMMENDATIONS

Methods for making bridge inspection faster, safer, and more thorough are needed for the aging transportation infrastructure. The IEGBS provides the image processing and storage capabilities needed for a video-based inspection system. As structural inspection makes the transition from visual and photographic inspection to a fully remote controlled video inspection system, a system is needed that can capture, store, and display digital images from the inspection process. These digital images can be archived for reports or viewing at the next inspection of the same structure to see how the structure has changed over time.

As with any new technology, it must not only provide significant benefit, but must be usable by the inspection teams with minimal training. The IEGBS accomplishes this by wrapping a user-friendly graphical interface around the control software so that the inspector is presented with easy to understand "control buttons" rather than memorizing and typing commands. An integrated display that shows the live video, still images, and the control buttons also facilitates learning and using the system so that the inspector can concentrate on the job of inspection rather than coping with a difficult video system.

The IEGBS also brings the capabilities of digital image processing to the inspection site. The fundamental image processing capabilities of capture, compression, storage, and retrieval are provided along with a powerful hardware-based image processor, the IDS system. The real-time image processing capability of the IDS has some benefit under backlight conditions where foreground objects are darkened. However, these conditions are not that common and can be handled in other ways besides using IDS image processing for a nominal improvement. Based on test image results, it is not recommended that a real-time IDS processor be incorporated in future implementations of the IEGBS. A hardware assisted IDS processor that can perform the processing on single digital images may still find some benefit over a purely software edge and contrast enhancement approach.

Near term applications for the IEGBS include using it in combination with a video camera mounted on an arm. Mounting a video camera on a pan-and-tilt platform that is, in turn, mounted on a man-capable inspection arm is probably the fastest and lowest cost approach to begin using video bridge inspection. Using a lower weight, longer reach arm that can be remotely controlled will provide the next level of capability for a video-based bridge inspection system. Such an arm has the potential for facilitating the quick inspection of most of the under-the-deck structures as well as some of the above-deck hardware. However, a development effort will be required to simplify the remote control this type of arm using only video from cameras mounted on the arm.

An aerial vehicle camera platform will provide the most freedom for the inspection camera at the expense of less stability and more difficult control. However, given the right combination of payload and high-level automatic control that will simplify flying, the aerial vehicle platform will be able to get the inspection camera into areas difficult to access using an arm.

While the IEGBS has many of the necessary features needed for a video-based inspection system, it does not have a complete set of features that the users may desire. Future enhancements could include: (1) smaller, lighter weight system, (2) digital video and sound capture, (3) integration with a "compound document" database, and (4) high-speed radio or cellular telephone modem link.

A smaller, lighter weight system will make it easier to deploy in the field. The goal here should be to make the system easily handled by one person - probably 10 kg or less. A self-contained, suite-case sized unit can be designed using current technology by replacing the current CRT display with a flat-screen, active-matrix LCD display, and eliminating the IDS processor. Making the IEGBS man-portable will greatly facilitate its use in the field.

The current IEGBS can only capture video and sound to the video tape recorder. The technology for direct video to disk is getting less expensive and more available. This will enable a future enhancement to the IEGBS to include direct digital capture of video segments and sound. This ability will enable the operator in the field to create "compound documents" detailing the inspection.

Unlike documents comprised of text and graphics, compound documents can include other elements such as digitized images, video segments, and sound. Using a compound document format, the inspector will be able to create an inspection report that includes: (1) text that describes the inspection procedure, bridge name, and location, (2) graphics that show CAD drawings of bridge sections, (3) still images of the current as well as past inspections, (4) digital video that shows some dynamic process like a crack opening and closing under traffic loads, and (5) digital sound that could be voice annotation of the inspection. An integrated compound document database will be needed to organized these elements into a single document.

An integrated radio or cellular telephone modem link will make the IEGBS part of a much larger network of computers dedicated to the collection, processing, and storage of inspection data. This will not only enable real-time data links to a central computer, but also links to other IEGBS units. This will enable a coordinated inspection of a single bridge from multiple IEGBS units that can share images and sound in real time.

As the technology and infrastructure for the "Information Superhighway" comes into place over the next 5 to 10 years, rapid transmission of digital information and images between remote inspection sites and a central data server will be a real possibility. For a video based inspection system that incorporates a local computer, this will mean almost

immediate access to huge amounts of information about the structure under inspection that are stored in the central server, and at the same time provide inspectors and supervisors that are distant to the structure under inspection almost instant access to images from the on going inspection. The IEGBS is the first step toward this future inspection information network.

REFERENCES

Cornsweet, T.N., and J.I. Yellott, Jr., "Intensity-dependent Spatial Summation," J. Opt. Soc. Am. Vol. 2, No. 10, pp 1769-1786, October 1985.

APPENDIX A

SOURCE CODE FOR IDS CONTROL COMPUTER

```
/*
$File, date, etc.$
$ModHeadBegin$

$Revision$
$LogFile$ (c) 1994, ODETICS, INC.

SUMMARY:

Bridge Inspector Project
IDS Image Processor

DESCRIPTION:

This file contains the IDS Image Processor control commands.

SEE ALSO:

idsControl.h

HISTORY:

$Log$


$ModHeadEnd$
*/

#include "idsControl.h"



/* ************************* Local Prototypes ************************* */
/*
        Commands:
*/


/* ********************** Main Processing Functions ********************** */
/*
        o Configure registers on DigiMax
                Initialize LUT's

        o Configure registers on FrameStores
*/
void initDigiMax(void)
{
/* Configure the DigiMax as follows:
        Disable Interrupts, Interlace, Ext. Phase lock
        0.0 volt offset
        No input filter, 1 gain, P12 pin 1 Analog input
        Enable Primary on P6, P7, P8, Select ILUT BankA
        Select Overlay OLUT BankA, Select Non-overlay OLUT BankB, Display MAXbus P6
        Disable Overlay
        Create uniform (don't alter values) input and output LUT's
*/
```

```
BYTE  i;

setByte(bDgSTiming,  (BYTE) 0x01);

setByte(bDgDCOffset, (BYTE) 0x0f);

setByte(bDgAnalog, (BYTE) 0xd0);

setByte(bDgC0, (BYTE) 0x00);

setByte(bDgC1, (BYTE) 0x04);

setByte(bDgOvlyCM, (BYTE) 0x00);

for(i=0; i<255;i++) {
        setByte(bDgLutA,  i);
        setByte(bDgGLutD,  i);
        setByte(bDgBLutD,  i);
        setByte(bDgRLutD,  i);
        setByte(bDgADCLutD,  i);
}
} /* End of initDigimax() */



void initFS0(void)
{
/* Configure FS0 as follows:
        60Hz, Interlace, Mode 4
        Constant 0
        End Acq B[0,1], interlace, even field, Start Acq FS_MEM,interlace, evenfield
        FS_P7 = 00, FS_RCV = DQ_P4
        All Pipelines Normal
        Select Primary output, Write Enable on all
        Horizontal and vertical position at 0

*/
setByte(bFs0C0,  (BYTE) 0x03);

setByte(bFs0K0,  (BYTE) 0x00);

setByte(bFs0Gate,  (BYTE) 0x08);

setByte(bFs0C1,  (BYTE) 0x00);

setByte(bFs0XmtGate,  (BYTE) 0x00);

setByte(bFs0C2,  (BYTE) 0x70);

setByte(bFs0MemWEna,  (BYTE) 0x00);

setShort(wFs0WordGHP,  (SHORT) 0x0000);

setShort(wFs0WordGVP,  (SHORT) 0x0000);

setShort(wFs0ByteGHP,  (SHORT) 0x0000);
```

```
setShort(wFs0ByteGVP, (SHORT) 0x0000);

} /* End of initFS0() */


void initFS1(void)
{
/* Configure FS1 as follows:
        60Hz, Interlace, Mode 4
        Constant 0
        End Acq B[0,1], interlace, even field, Start Acq FS_MEM,interlace, evenfield
        FS_P7 = 00, FS_RCV = DQ_P4
        All Pipelines Normal
        Select Primary output, Write Enable on all
        Horizontal and vertical position at 0

*/
setByte(bFs1C0, (BYTE) 0x03);

setByte(bFs1K0, (BYTE) 0x00);

setByte(bFs1Gate, (BYTE) 0x08);

setByte(bFs1C1, (BYTE) 0x00);

setByte(bFs1XmtGate, (BYTE) 0x00);

setByte(bFs1C2, (BYTE) 0x70);

setByte(bFs1MemWEna, (BYTE) 0x00);

setShort(wFs1WordGHP, (SHORT) 0x0000);

setShort(wFs1WordGVP, (SHORT) 0x0000);

setShort(wFs1ByteGHP, (SHORT) 0x0000);

setShort(wFs1ByteGVP, (SHORT) 0x0000);

} /* End of initFS1() */


void freezeFS0(void)
{
/* Stop the acquisition on FS0_Byte_Mem to freeze the video */

BYTE   gate, freezeMask = 0xf7;

gate = getByte(bFs0Gate);

setByte(bFs0Gate, gate & freezeMask);


} /* End of freezeFS0() */
```

```
void liveFS0(void)
{
/* Start the acquisition on FS0_Byte_Mem  */

BYTE   gate, liveMask = 0x08;

gate = getByte(bFs0Gate);

setByte(bFs0Gate, gate | liveMask);


} /* End of liveFS0() */

/*
        o Configure registers on DigiMax
                Initialize LUT's

        o Configure registers on FrameStores
*/
void con(int val)
{
/* Configure the DigiMax as follows:
        Disable Interrupts, Interlace, Ext. Phase lock
        0.0 volt offset
        No input filter, 1 gain, P12 pin 1 Analog input
        Enable Primary on P6, P7, P8, Select ILUT BankA
        Select Overlay OLUT BankA, Select Non-overlay OLUT BankB, Display MAXbus P6
        Disable Overlay
        Create uniform (don't alter values) output LUT's
        Set Input LUT to val
*/
BYTE   i;

setByte(bDgSTiming,  (BYTE) 0x01);

setByte(bDgDCOffset, (BYTE) 0x0f);

setByte(bDgAnalog, (BYTE) 0xd0);

setByte(bDgC0, (BYTE) 0x00);

setByte(bDgC1, (BYTE) 0x04);

setByte(bDgOvlyCM, (BYTE) 0x00);

for(i=0; i<255;i++) {
        setByte(bDgLutA,  i);
        setByte(bDgGLutD,  i);
        setByte(bDgBLutD,  i);
        setByte(bDgRLutD,  i);
        /**** setByte(bDgADCLutD,  i);  ***/
}

for(i=0; i<255;i++) {
```

```
        setByte(bDgLutA, i);
        setByte(bDgADCLutD, (BYTE) val);
}


} /* End of initDigimax() */

void rampFS0(void)
{
/* Write a ramp in FS0
*/
int     x, y;

for(y=0;y<512;y++)
 for(x = 0;x<512;x++){
    setByte(bFs0ByteMem + (x + y*512), (BYTE) (x & 0x0ff));
 }
}
```

```
#include        "dgHead.h"
#include        "digi.h"
sa_dgP8InpMd(base,value)
unsigned char  *base;
int value;
{
  if(value==DQ_ALTERN)
    *(base+CR0) |= 0x80;
  else
    *(base+CR0) &= ~0x80;
}

sa_dgP6InpMd(base,value)
unsigned char *base;
int value;
{
  if(value==DQ_ALTERN)
    *(base+CR0) |= 0x20;
  else
    *(base+CR0) &= ~0x20;
}

int *sa_dgOpen(dgbase)
int dgbase;
{
  return(int *)dgbase;
}
```

```
/*
 * etherDispatch.c - A task on the ids control computer that receives
 * messages from the
 */
/* IE-GBS host computer and calls the appropriate function.                    */
/* Sets up ids side of the socket. */
/*
 * This task will automatically block on the recv command waiting for a
 * message packet. This means tasks of a lower priority will run, yet
 * status commands from the host will not be blocked by a lower priority
 * task. Conversely, if message packets are constantly being sent over,
 * the lower priority task will be blocked.
 */

/*
 * modification history ---------------- 7/6/94      Uses MESSAGE_SIZE
 * length messages. Messages must be NULL-terminated
 */


#include <vxWorks.h>
#include <socket.h>
#include <in.h>
#include "etherDispatch.h"
#include <taskLib.h>

#include "idsControl.h"
#include "frame.h"
#include "digi.h"
#include "idsParse.h"


unsigned short *s_cone, *s_del2g_09, *s_exp_10, *s_exp_05;

/*
 * Define the external function declarations so that the spawn function
 * will link to the function.
 */
extern int     idsRTLoop ();
extern int     idsLoadram ();
extern int     idsDecomp ();
extern int     idsSendFrm();
extern int     idsGetFrm();


/* IDS operation status */
int          status;                /* This will be extern to all other
                                     * functions that use it. */

/*****************************************************************************
*
* dispatch
* This function receives character packets from the  inspector host computer
* and translates them into actions.
* The message packets reflect the codes that are used from the ether port to control the ids
* processor.
```

```
* The valid messages are:
*       >acquire        // Start acquiring images in board 0 FS 0
*       >freeze // Freeze image in board 0 FS 0
*       >init    // Initialize the IDS boards
*       >decomp         // Decompress the IDS spread functions
*       >load   // Load the IDS spread functions
*       >histo(low, high)    .      // Modify the histogrammer
*       >spread(0or1)   // Select spread function
*       >realtime       // Begin real-time IDS processing (spawns a task)
*       >endreal        // End real-time IDS processing (deletes the spawned task)
*       >oneshot        // Process one frame of data store it in board 0 FS 1
*       >sendfrm        // Send the processed image
*       >getfrm// Accept a frame of image data
*       >status // Return status code
* RETURNS: OK or ERROR
*/

dispatch ()
{
   int          sock, snew, imExSock;     /* socket fd's */
   struct sockaddr_in serverAddr;         /* server's address */
   struct sockaddr_in clientAddr;/* client's address */
   int          client_len; /* length of clientAddr */
   char          select[MESSAGE_SIZE], statusMsg[MESSAGE_SIZE];
   extern int     errno;   /* for error referencing */

   LONG          lineCount, i;

   pBYTE         lineBuf, B1, B2, B3, B4, sBuf, imageBuf, imagePtr, pPix;
   pSHORT         sPix;

   /*
    * Up to 10 parameters can be passed to the token-identified
    * functions.
    */
   int          token, param[10];

   FUNCPTR        pRTLoop, pDecomp, pLoad, pSendFrm, pGetFrm;
   int          loopTask = 0, decompTask = 0, loadTask = 0, sendTask=0, getTask = 0;

   /* Define the function pointers */
   pRTLoop = (FUNCPTR) idsRTLoop;
   pDecomp = (FUNCPTR) idsDecomp;
   pLoad = (FUNCPTR) idsLoadram;
   pSendFrm = (FUNCPTR) idsSendFrm;
   pGetFrm = (FUNCPTR) idsGetFrm;


   /* Allocate memory for the buffers */
   lineBuf = (pBYTE) malloc (512);
   B1 = (pBYTE) malloc (512);
   B2 = (pBYTE) malloc (512);
   B3 = (pBYTE) malloc (512);
   B4 = (pBYTE) malloc (512);
   sBuf = (pBYTE) malloc (512);
   imageBuf = (pBYTE) malloc (512 * 512);
```

```
    if (!lineBuf || !B1 || !B2 || !B3 || !B4 || !imageBuf)
    {
        printf (" OUT OF IMAGE MEMORY\n");
        return (NULL);
    }



    /* Allocate enough memory for the spread functions. */
    s_cone = (unsigned short *) malloc (128 * 1024);
    /* Not used ----->> s_del2g_09=(unsigned short *)malloc(128*1024); */

    s_exp_10 = (unsigned short *) malloc (128 * 1024);
    /* Not used ------->>> s_exp_05=(unsigned short *)malloc(128*1024); */
    if (!s_cone || !s_exp_10)
    {
        printf (" OUT OF MEMORY\n");
        return (NULL);
    }

    /* Before opening the communication socket, create tasks to decomp and load spread
functions */

    status = RAW;                   /* IDS is operating but uninitialized */

    /*
     * Spawn a VxWorks task to perform the decompressing of the spread
     * functions
     */
    decompTask = taskSpawn ("idsDecompTask", DECOMP_PRIORITY,
                            VX_DEALLOC_STACK | VX_STDIO,
                            IDS_STACK_SIZE, pDecomp);
    if (decompTask != ERROR)
        taskActivate (decompTask);

    /* Spawn a VxWorks task to load the spread functions into memory */
    loadTask = taskSpawn ("idsLoadTask", LOAD_PRIORITY,
                          VX_DEALLOC_STACK | VX_STDIO,
                          IDS_STACK_SIZE, pLoad);
    if (loadTask != ERROR)
        taskActivate (loadTask);


    /*
     * Zero out the sock_addr structures. This MUST be done before the
     * socket calls.
     */

    bzero (&serverAddr, sizeof (serverAddr));
    bzero (&clientAddr, sizeof (clientAddr));

    /*
     * Open the socket. Use ARPA Internet address format and stream
     * sockets. Format described in "socket.h".
     */
```

```
sock = socket (AF_INET, SOCK_STREAM, 0);

if (sock == ERROR)
      exit (1);

/* Set up our internet address, and bind it so the client can connect. */

serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons (SERVER_NUM);

printf ("\nBinding SERVER\n");

if (bind (sock, (SOCKADDR *) & serverAddr, sizeof (serverAddr)) == ERROR)
{
      printf ("bind failed, errno = %d\n", errno);
      close (sock);
      exit (1);
}

/* Listen, for the client to connect to us. */

printf ("Listening to client\n");

if (listen (sock, 2) == ERROR)
{
      printf ("listen failed\n");
      close (sock);
      exit (1);
}

/* The client has connected.  Accept, and receive chars */

printf ("Accepting CLIENT\n");

client_len = sizeof (clientAddr);

snew = accept (sock, (SOCKADDR *) & clientAddr, &client_len);

if (snew == ERROR)
{
      printf ("accept failed\n");
      close (sock);
      exit (1);
}

printf ("CLIENT: port = %d: family = %d: addr = %lx:\n",
         clientAddr.sin_port, clientAddr.sin_family,
         clientAddr.sin_addr.s_addr);



/* Now open the image exchange socket */
if ((imExSock = openImExSocket ()) == ERROR)
      printf ("ERROR opening imExSock\n");

 /* Receive mesages and determine which action to perform. */
```

```
do
{
        if (recv (snew, select, MESSAGE_SIZE, 0) == 0)
        {
          /* client has disappeared */
          break;
        }

        token = idsParse (select, param);


        switch (token)
        {
         case ACQUIRE:
          ids_fs0AcquireMd (0, FS_ACQUIRE);
          ids_fsSelOutSrc (0, FS_0OUT);
          ids_dgP8InpMd (DQ_PRIMARY);
          break;

         case FREEZE:
          ids_fs0AcquireMd (0, FS_FREEZE);
          ids_fsSelOutSrc (0, FS_0OUT);
          ids_dgP8InpMd (DQ_PRIMARY);
          break;

         case INIT:
          printf (" Initialize       \n ");
          idsInitdq ();
          status = INITIAL;
          break;

         case HISTO:
          printf (" Modify histogrammer \n");
          /* idsHisto2(histoLevel, aluConst, shift) */
          idsHisto2 ((BYTE) param[0], (BYTE) param[1], (BYTE) param[2]);
          break;

         case LOAD:
          printf (" Loadram          \n");
          idsLoadram ();
          break;

         case DECOMP:
          printf (" Decompress Spreads  \n");
          idsDecomp ();
          break;

         case SPREAD:
          printf ("Select spread  %d\n", param[0]);
          idsFrspsel (param[0]);
          break;

         case REALTIME:
          /* Spawn a VxWorks Task */
          /* Make sure only one idsLoopTask is running. */
          if ((loopTask = taskNameToId("idsLoopTask")) != ERROR){
```

```
        taskDelete (loopTask); /* Delete old task if it exists */
    }
    loopTask = taskSpawn ("idsLoopTask", IDS_PRIORITY,
                          VX_DEALLOC_STACK | VX_STDIO,
                          IDS_STACK_SIZE, pRTLoop, HSHIFT);
    if (loopTask != ERROR)
        taskActivate (loopTask);
    printf (" Begin Processing   \n");
    break;

case ENDREAL:
    printf (" End Processing     \n");
    if ((loopTask = taskNameToId("idsLoopTask")) != ERROR){
        taskDelete (loopTask);
        status = READY;
    }
    break;

case ONESHOT:
    printf (" One-shot           \n");
    idsOne ();
    break;

case GETFRM:                    /* Receive an image over the net and load
                                * it into Board 0 FS 0 */
    if (imExSock != ERROR)
    {
        getTask = taskSpawn ("idsGetTask", DECOMP_PRIORITY,
                  VX_DEALLOC_STACK | VX_STDIO,
                  IDS_STACK_SIZE, pGetFrm,
                  imExSock, lineBuf, imageBuf);
        if (getTask != ERROR)
            taskActivate (getTask);

        /**idsGetFrm (imExSock, lineBuf, imageBuf);  **/

    }
    break;

case SENDFRM:       /* Send an image over the net  from Board
                     * FS 1 */
    if (imExSock != ERROR)
    {
        sendTask = taskSpawn ("idsSendTask", DECOMP_PRIORITY,
                  VX_DEALLOC_STACK | VX_STDIO,
                  IDS_STACK_SIZE, pSendFrm,
                  imExSock, lineBuf, B1, B2, B3, B4, sBuf, imageBuf);
        if (sendTask != ERROR)
            taskActivate (sendTask);

        /** idsSendFrm (imExSock, lineBuf, B1, B2, B3, B4,  sBuf, imageBuf);  **/


    }
    break;
```

```
            case STATUS:
              /* Send a status message to the host */
              sprintf(statusMsg, "%d", status);
              send (snew, statusMsg,MESSAGE_SIZE, 0);
              printf ("Status = %d \n", status);
              break;

            case INVALID:
              printf ("Invalid cmd\n");
              break;

            default:
              printf ("Unknown cmd \n");
              break;
          }
    } while (TRUE);

    printf ("      QUIT \n");

    free (s_cone);
    /* Not used --->>> free(s_del2g_09);  */
    free (s_exp_10);
    /* Not used ---->>> free(s_exp_05);  */

    free (lineBuf);
    free (B1);
    free (B2);
    free (B3);
    free (B4);
    free (sBuf);
    free (imageBuf);

    /* close the socket on the dispatch side */
    close (snew);
    close (sock);

    if (imExSock != ERROR)
          close (imExSock);

    printf ("\n...goodbye\n");
}

/*******************************************************************************/
openImExSocket ()
{
    int          sock, snew;          /* socket fd's */
    struct sockaddr_in serverAddr;          /* server's address */
    struct sockaddr_in clientAddr;/* client's address */
    int          client_len; /* length of clientAddr */

    /*
     * Zero out the sock_addr structures. This MUST be done before the
     * socket calls.
     */

    bzero (&serverAddr, sizeof (serverAddr));
```

```
bzero (&clientAddr, sizeof (clientAddr));

/*
 * Open the socket. Use ARPA Internet address format and stream
 * sockets. Format described in "socket.h".
 */

sock = socket (AF_INET, SOCK_STREAM, 0);

if (sock == ERROR)
      return (ERROR);

/* Set up our internet address, and bind it so the client can connect. */

serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons (IMAGE_X_PORT);

printf ("\nBinding SERVER\n");

if (bind (sock, (SOCKADDR *) & serverAddr, sizeof (serverAddr)) == ERROR)
{
      printf ("bind failed, errno = %d\n", errno);
      close (sock);
      return (ERROR);
}

/* Listen, for the client to connect to us. */

printf ("Listening to client\n");

if (listen (sock, 2) == ERROR)
{
      printf ("listen failed\n");
      close (sock);
      return (ERROR);
}

/* The client has connected.  Accept, and receive chars */

printf ("Accepting Image Ex\n");

client_len = sizeof (clientAddr);

snew = accept (sock, (SOCKADDR *) & clientAddr, &client_len);

if (snew == ERROR)
{
      printf ("accept failed\n");
      close (sock);
      return (ERROR);
}

close (sock);          /* Since the accept worked, close the old
                        * sock. */
```

```
    printf ("Image Ex: port = %d: family = %d: addr = %lx:\n",
            clientAddr.sin_port, clientAddr.sin_family,
            clientAddr.sin_addr.s_addr);
    return (snew);

}
```

```
/*
Summary:
Bridge Inspector Project
IDS Image Processor
Decompress the spread functions.

Description:
This decompresses all the spread functions into the processor RAM.

See Also:
S. Strang's sa_decomp.c (the basis for this program)

*/

#include "idsControl.h"
#include "idsParse.h"

static int f_cone,f_del2g_09,f_exp_10,f_exp_05;

idsDecomp ()
{
  extern int    status;


  status = DECOMPRESSING;

  printf ("Decompress cone \n");
  scone ();
  f_cone = 1;

  printf ("Decompress exp10 \n");
  sexp10 ();
  f_exp_10 = 1;

  /********
  No Vx Works Floating point support
  sexp05();
   f_exp_05=1;
  ***************/
  /**************
  No VxWorks Floating point support
  sdel2g09();
  f_del2g_09=1;
  ******************/

  status = DONE_DECOMP;
}

/*************************************************************************/
extern unsigned short *s_cone;
scone()
{
 extern unsigned char spreadc[];

 register unsigned char *cptr;
 register int i=0;
```

```
    register j;
    register shift=0;
    unsigned char zeros, ones;

    cptr=spreadc;
    /* Decode spread from run-length coded into "normal" sr spread format */
    while(!( *cptr==0 && *(cptr+1)==0 ) ){
      zeros=*cptr++;
      for(j=0;j<zeros;j++){
            s_cone[i]=s_cone[i] & 0xFFFE ;
        shift++;
        if( (shift%16) == 0)
          i++;
        else
          s_cone[i]<<=1;
      }
      ones=*cptr++;
      for(j=0;j<ones;j++){
            s_cone[i]=s_cone[i] | 0x0001 ;
        shift++;
        if( (shift%16) == 0)
          i++;
           else
          s_cone[i]<<=1;
      }
    }
}

/*****************************************************************************/

extern unsigned short *s_exp_10;
sexp10()
{
  extern unsigned char spreade[];

  register unsigned char *cptr;
  register int i=0;
  register j;
  register shift=0;
  unsigned char zeros, ones;

  cptr=spreade;

/* Decode spread from run-length coded into "normal" sr spread format */
  while(!( *cptr==0 && *(cptr+1)==0 ) ){
    zeros=*cptr++;
    for(j=0;j<zeros;j++){
          s_exp_10[i]=s_exp_10[i] & 0xFFFE ;
      shift++;
      if( (shift%16) == 0)
        i++;
      else
        s_exp_10[i]<<=1;
    }
    ones=*cptr++;
    for(j=0;j<ones;j++){
```

```
            s_exp_10[i]=s_exp_10[i] | 0x0001 ;
        shift++;
        if( (shift%16) == 0)
          i++;
           else
          s_exp_10[i]<<=1;
      }
    }

}

/*****************************************************************************/
```

```
#include "idsControl.h"

#include "digi.h"


/*      These are trimmed down versions of the Datacube routines.
        All hardware accesses are done immediately
        No structure or shadow registers are used, unlesss specifically needed.

        These are intended to be used for the standalone system.

        Only those functions neede are implemented.
*/

/* all base addresses are for register base, not card base */



ids_dgSelDtoASrc(value)
BYTE   value;
{
  *bDgC1 &= 0xfc;       /* Clear bits */
  *bDgC1 |= value;
}


ids_waitNextEven()
{
while(*bDgSTiming & 0x40) { }  /* Loop until the next even field */

}


ids_waitNextOdd()
{
while((*bDgSTiming & 0x40)  == 0x00) { }  /* Loop until the next odd field */

}


ids_dgWaitOdd()
{

   ids_waitNextEven();
   ids_waitNextOdd();
}

ids_dgWaitEven()
{
    ids_waitNextOdd();
    ids_waitNextEven();
}

ids_waitNextLowV()
{
while(*bDgSTiming & 0x80) { }  /* Loop until the next low flag */
```

```
}

ids_waitNextHighV()
{
while((*bDgSTiming & 0x80)  == 0x00) { }  /* Loop until the next high flag */

}


ids_dgWaitFld(value)
int       value;
/* Loop for value number of high->low transitions of VA flag */
{
   int    i;

   for(i=0;i<value;i++) {
     ids_waitNextHighV();  /* Loop until the next high flag */
     ids_waitNextLowV();  /* Loop until the next low flag */
   }

}


ids_dgP8InpMd(value)
BYTE   value;
{
  *bDgC0 &= 0x7f;        /* Clear bits */
  *bDgC0 |= (value << 7);
}




ids_fs12AcqStrt(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0Gate &= 0xcf; /* clear bits */
         *bFs0Gate |= (value << 4);
   } else {
         *bFs1Gate &= 0xcf; /* clear bits */
         *bFs1Gate |= (value << 4);
   }
}

ids_fs0AcqStrt(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0Gate &= 0xfc; /* clear bits */
         *bFs0Gate |= (value);
```

```
    } else {
        *bFs1Gate &= 0xfc; /* clear bits */
        *bFs1Gate |= (value);
    }
}

ids_fsSelOutSrc(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
    if(boardNum == 0){
        *bFs0C1 &= 0x3f; /* clear bits */
        *bFs0C1 |= (value << 6);
    } else {
        *bFs1C1 &= 0x3f; /* clear bits */
        *bFs1C1 |= (value << 6);
    }
}

ids_fs1Input(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
    if(boardNum == 0){
        *bFs0C1 &= 0xf3; /* clear bits */
        *bFs0C1 |= (value << 2);
    } else {
        *bFs1C1 &= 0xf3; /* clear bits */
        *bFs1C1 |= (value << 2);
    }
}

ids_fs2Input(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
    if(boardNum == 0){
        *bFs0C1 &= 0xcf; /* clear bits */
        *bFs0C1 |= (value << 4);
    } else {
        *bFs1C1 &= 0xcf; /* clear bits */
        *bFs1C1 |= (value << 4);
    }
}

ids_fs0Input(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
    if(boardNum == 0){
        *bFs0C1 &= 0xfc; /* clear bits */
        *bFs0C1 |= (value );
    } else {
        *bFs1C1 &= 0xfc; /* clear bits */
        *bFs1C1 |= (value );
    }
```

```
}

ids_fsVIntlMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0C0 &= 0xbf; /* clear bits */
         *bFs0C0 |= (value << 6);
   } else {
         *bFs1C0 &= 0xbf; /* clear bits */
         *bFs1C0 |= (value << 6);
   }
}

ids_fs12VResMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0C0 &= 0xfd; /* clear bits */
         *bFs0C0 |= (value << 1);
   } else {
         *bFs1C0 &= 0xfd; /* clear bits */
         *bFs1C0 |= (value << 1);
   }
}

ids_fs12TimeMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0Gate &= 0xbf; /* clear bits */
         *bFs0Gate |= (value << 6);
   } else {
         *bFs1Gate &= 0xbf; /* clear bits */
         *bFs1Gate |= (value << 6);
   }
}

ids_fs12IncScroll(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
int    value;
{
   if(boardNum == 0){
         *wFs0WordGVP += value;
   } else {
         *wFs1WordGVP += value;
   }
}

#ifdef __CPLUSPLUS
void ids_fs12IncPan(BYTE boardNum, short value)
#else
void ids_fs12IncPan( boardNum, value)
```

```
BYTE boardNum;
short value;
#endif
/* boardNum  0 for Fs0 and 1 for Fs1 */
{
   if(boardNum == 0){
         *wFs0WordGHP += value;
   } else {
         *wFs1WordGHP += value;
   }
}

ids_fs0IncScroll(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
int        value;
{
   if(boardNum == 0){
         *wFs0ByteGVP += value;
   } else {
         *wFs1ByteGVP += value;
   }
}

#ifdef __CPLUSPLUS
void ids_fs0IncPan(BYTE boardNum, short value)
#else
void ids_fs0IncPan(boardNum, value)
BYTE boardNum;
short value;
#endif

/* boardNum  0 for Fs0 and 1 for Fs1 */
{
   if(boardNum == 0){
         *wFs0ByteGHP += value;
   } else {
         *wFs1ByteGHP += value;
   }
}


ids_fs0VResMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0C0 &= 0xfe; /* clear bits */
         *bFs0C0 |= (value );
   } else {
         *bFs1C0 &= 0xfe; /* clear bits */
         *bFs1C0 |= (value );
   }
}

ids_fs0TimeMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
```

```
BYTE   value;
{
   if(boardNum == 0){
         *bFs0Gate &= 0xfb; /* clear bits */
         *bFs0Gate |= (value << 2);
   } else {
         *bFs1Gate &= 0xfb; /* clear bits */
         *bFs1Gate |= (value << 2);
   }
}

ids_fs0HPipMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0XmtGate &= 0xfe; /* clear bits */
         *bFs0XmtGate |= (value );
   } else {
         *bFs1XmtGate &= 0xfe; /* clear bits */
         *bFs1XmtGate |= (value );
   }
}

 ids_fs0VPipMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0XmtGate &= 0xfb; /* clear bits */
         *bFs0XmtGate |= (value << 2 );
   } else {
         *bFs1XmtGate &= 0xfb; /* clear bits */
         *bFs1XmtGate |= (value << 2);
   }
}

ids_fs0AcquireMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
         *bFs0Gate &= 0xf7; /* clear bits */
         *bFs0Gate |= (value << 3 );
   } else {
         *bFs1Gate &= 0xf7; /* clear bits */
         *bFs1Gate |= (value << 3);
   }
}


ids_fs12AcquireMd(boardNum,value)
BYTE   boardNum; /* 0 for Fs0 and 1 for Fs1 */
BYTE   value;
{
   if(boardNum == 0){
```

```
            *bFs0Gate &= 0x7f; /* clear bits */
            *bFs0Gate |= (value << 7 );
        } else {
            *bFs1Gate &= 0x7f; /* clear bits */
            *bFs1Gate |= (value << 7);
        }
    }


    /*****************************************************/
    ids_hjP5InpMd(value)
    int value;
    {
      if(value==DQ_ALTERN)
        *(pSHORT)HISTO_base = 0x1;
      else
        *(pSHORT)HISTO_base = 0x0;
    }


    /*****************************************************************/
```

```
#include "idsControl.h"
#include "frArg.h"


/** #include <sa_datacube.h>  */
/** #include <frHead.h> ***/.
/*** #include <fpp.h>   ***/

#define PORT 1
#define DPORT 0
#ifdef __CPLUSPLUS
void idsFrspsel(int sel)
#else
void idsFrspsel(sel)
int sel;
#endif
{
int  base,addr;
char spsel;
pBYTE cptr;
int lowbase,highbase;
SHORT *ctrl,ctrl_shad;

lowbase=(int)FPPSR_0;
highbase=(int)FPPSR_7;


for(base=lowbase;base<=highbase;base+=0x100){
   addr=base;
  ctrl=(pSHORT)addr;
  ctrl_shad=0;
  ctrl_shad=ctrl_shad &~ NORMAL;
  *ctrl=ctrl_shad;
  ctrl_shad=ctrl_shad | AB_DC;
  ctrl_shad=ctrl_shad &~ VIDEO_REG;
  if(sel == 0)
    ctrl_shad=ctrl_shad &~ SPSEL_1;
  else
    ctrl_shad=ctrl_shad | SPSEL_1;
  ctrl_shad=ctrl_shad | NORMAL;
  *ctrl=ctrl_shad;
  }
 }
```

```
/* idsGetFrm.c
 * Receive an image over the net and load it into Board 0 FS 0
 */

#include <vxWorks.h>
#include <socket.h>
#include <in.h>
#include "etherDispatch.h"
#include <taskLib.h>

#include "idsControl.h"
#include "frame.h"
#include "digi.h"
#include "idsParse.h"

int         idsGetFrm (imExSock, lineBuf, imageBuf)
int         imExSock;
pBYTE       lineBuf,  imageBuf;

{

  LONG              lineCount, i;

  pBYTE             imagePtr;


  printf ("Loading image... ");
  imagePtr = &imageBuf[0];
  for (lineCount = 0; lineCount < 512; lineCount++)
  {                                 /* Get a 512 x 512 image */
       if (recv (imExSock, lineBuf, 512, 0) == 0)
       {
          /* client has disappeared */
          break;
       }
       /* Copy lineBuf to imageBuf */
       for (i = 0; i < 512; i++)
       {
          *imagePtr = lineBuf[i];
          imagePtr++;
       }

  }
  if (lineCount == 512)
  {                                 /* We have a full 512 x 512 image */
       /* copy to board 0 frame store 0 */
       for (i = 0; i < (512 * 512); i++)
          setByte (bFs0Mem0 + i, imageBuf[i]);
  }

  printf ("Done loading image\n");


}
```

```
#include "maxsp.h"
#include "idsControl.h"

/* Modification history
7/6/94  Rather than a fixed set of options, you pass in a set of parameters
        that determines the operation. See idsHisto.h
*/

idsHisto2(histoLevel, aluConst, shift)
BYTE   histoLevel,aluConst, shift;
{
   pBYTE        cptr;

  cptr = (pBYTE) HISTO_base;

  msSelMin();  /* select min,clipp upper bounds */
  msSelPixMd(MS_PA);     /* Set PCRA always  */
  msSetClip(MS_BYPASS);     /* set to 0 for unsigned under flow */
  msOperA(MS_PCRA,MS_MNMXOUT);  /* A=Output of Min-Max */
  msOperB(MS_PCRA,MS_ALUCOUT); /* B=Constant */
  msSelOpMode(MS_PCRA,MS_AMINUSB); /* A - B */
  msCarryMd(MS_PCRA,DQ_DISABLE);
  msWtAluLo(0);
  msPrDataMd(1); /* select logical operations */
  msClipMd(MS_00UNSGD);
  msMinMaxMd(MS_PCRA,MS_NORMAL);


  *(cptr+0x21)=histoLevel;
  msWtAluHi(aluConst);
  msShift(shift);

}



/*****************************************
Minimal MaxSp support functions to perform the histogram function.

*****************************************/

static   SHORTMoscAShadow= 0, MoscBShadow = 0;
/* These registers need to be shadowed because they are write only. */

msSelMin()      /* select min */
{
  *wMAXSPC0 &= 0xff3f;          /* Clear bits */
  *wMAXSPC0 |= 0x0040;
}

msSelPixMd(value)
SHORTvalue;
{
  *wMAXSPC0 &= 0xf0ff;          /* Clear bits */
  *wMAXSPC0 |= (value << 8);
```

```
}

msSetClip(value)
SHORT value;
{
  *wMAXSPC1 &= 0xe7ff;        /* Clear bits */
  *wMAXSPC1 |= (value << 11);
}

msOperA(PCRSel,OpA)
SHORT PCRSel, OpA;
{
  if (PCRSel == MS_PCRA) {
    MoscAShadow &= 0xfffc;  /* clear shadow bits */
    MoscAShadow |= OpA;     /* Set shadow bits */
    *wMAXSPMoscA = MoscAShadow;  /* Write shadow bits */
  } else {
    MoscBShadow &= 0xfffc;  /* clear shadow bits */
    MoscBShadow |= OpA;     /* Set shadow bits */
    *wMAXSPMoscB = MoscBShadow;  /* Write shadow bits */
  }
}

msOperB(PCRSel,OpB)
SHORT PCRSel, OpB;
{
  if (PCRSel == MS_PCRA) {
    MoscAShadow &= 0xfff3;  /* clear shadow bits */
    MoscAShadow |= (OpB << 2);        /* Set shadow bits */
    *wMAXSPMoscA = MoscAShadow;  /* Write shadow bits */
  } else {
    MoscBShadow &= 0xfff3;  /* clear shadow bits */
    MoscBShadow |= (OpB << 2);        /* Set shadow bits */
    *wMAXSPMoscB = MoscBShadow;  /* Write shadow bits */
  }
}

msSelOpMode(PCRSel,Op)
SHORT PCRSel, Op;
{
  if (PCRSel == MS_PCRA) {
    MoscAShadow &= 0xff8f;  /* clear shadow bits */
    MoscAShadow |= (Op << 4);         /* Set shadow bits */
    *wMAXSPMoscA = MoscAShadow;  /* Write shadow bits */
  } else {
    MoscBShadow &= 0xff8f;  /* clear shadow bits */
    MoscBShadow |= (Op << 4);         /* Set shadow bits */
    *wMAXSPMoscB = MoscBShadow;  /* Write shadow bits */
  }
}

msCarryMd(PCRSel,Op)
SHORT PCRSel, Op;
{
  if (PCRSel == MS_PCRA) {
    MoscAShadow &= 0xff7f;  /* clear shadow bits */
```

```
    MoscAShadow |= (Op << 7);          /* Set shadow bits */
    *wMAXSPMoscA = MoscAShadow; /* Write shadow bits */
  } else {
    MoscBShadow &= 0xff7f; /* clear shadow bits */
    MoscBShadow |= (Op << 7);          /* Set shadow bits */
    *wMAXSPMoscB = MoscBShadow; /* Write shadow bits */
  }
}

msWtAluLo(value)
BYTE   value;
/* Set lower byte */
{
   *wMAXSPK2 = (SHORT) value;
}

msPrDataMd(DataMode)
SHORT DataMode;
{
  *wMAXSPC0 &= 0xffdf;          /* Clear bits */
  *wMAXSPC0 |= (DataMode << 5);
}

msClipMd(clip)
SHORT clip;
{
  *wMAXSPC1 &= 0xe7ff;          /* Clear bits */
  *wMAXSPC1 |= (clip << 11);
}

msMinMaxMd(PCRSel,Op)
SHORT PCRSel, Op;
{
  if (PCRSel == MS_PCRA) {
    MoscAShadow &= 0xfeff; /* clear shadow bits */
    MoscAShadow |= (Op << 8);          /* Set shadow bits */
    *wMAXSPMoscA = MoscAShadow; /* Write shadow bits */
  } else {
    MoscBShadow &= 0xfeff; /* clear shadow bits */
    MoscBShadow |= (Op << 8);          /* Set shadow bits */
    *wMAXSPMoscB = MoscBShadow; /* Write shadow bits */
  }
}

msWtAluHi(value)
SHORT value;
/* Set upper byte */
{
   *wMAXSPK2 = (SHORT) (value << 8 );
}

msShift(value)
SHORT value;
{
  *wMAXSPC1 &= 0x1fff;          /* Clear bits */
  *wMAXSPC1 |= (value << 13);
```

}

```
/*
$File, date, etc.$
$ModHeadBegin$

$Revision$
$LogFile$ (c) 1994, ODETICS, INC.

SUMMARY:

Bridge Inspector Project
IDS Image Processor

DESCRIPTION:

This file contains the DataCube initialization functions.

SEE ALSO:

idsControl.h

HISTORY:

$Log$


$ModHeadEnd$
*/

#include "idsControl.h"



/* ************************* Local Prototypes ************************** */
/*
        Commands:
*/


/* ********************** Main Processing Functions ********************** */
/*
        o Configure registers on DigiMax
                Initialize LUT's

        o Configure registers on FrameStores
*/


idsInitdq()
{
  initDigiMax();
  initFS0();
  initFS1();
  initMaxSP();
}
```

```
/*** From S. Strang sa_initdq()
  dgInit(dgfd0,DG_UNSGD);
  dgSyncMd(dgfd0,DG_EXTERN);
  dgSelDtoASrc(dgfd0,DG_P8DISP);
  dgAnFilter(dgfd0,DG_0MHZ);
  dgAnOffs(dgfd0,22);
  dgAnGain(dgfd0,1);
  fsInit(fsfd0,FS_T60);
  fsSel0Input(fsfd0,FS_0P4);
  fsSel1Input(fsfd0,FS_0P9);
  fsSel2Input(fsfd0,FS_0P10);
  fsInit(fsfd1,FS_T60);
  fsSel0Input(fsfd1,FS_0P4);
  fsSel1Input(fsfd1,FS_0P9);
  fsSel2Input(fsfd1,FS_0P10);
  msInit(msfd0,DQ_M512);
*****/
#ifdef __CPLUSPLUS
void initDigiMax(void)
#else
void initDigiMax()
#endif
{
/* Configure the DigiMax as follows:
        Disable Interrupts, Interlace, Ext. Phase lock
        22 offset
        No input filter, 1 gain, P12 pin 1 Analog input
        Enable Primary on P6, P7, P8, Select ILUT BankA
        Select Overlay OLUT BankA, Select Non-overlay OLUT BankB, Display MAXbus P8
        Disable Overlay
        Create uniform (don't alter values) input and output LUT's
*/
BYTE   i;

setByte(bDgSTiming,  (BYTE) 0x01);

setByte(bDgDCOffset, (BYTE) 0x16);

setByte(bDgAnalog, (BYTE) 0xd0);

setByte(bDgC0, (BYTE) 0x00);

setByte(bDgC1, (BYTE) 0x06);

setByte(bDgOvlyCM, (BYTE) 0x00);

for(i=0; i<255;i++) {
        setByte(bDgLutA,  i);
        setByte(bDgGLutD,  i);
        setByte(bDgBLutD,  i);
        setByte(bDgRLutD,  i);
        setByte(bDgADCLutD,  i);
}
} /* End of initDigimax() */
```

```
#ifdef __CPLUSPLUS
void initFS0(void)
#else
void initFS0()
#endif
{
/* Configure FS0 as follows:
        60Hz, Interlace, Mode 4
        Constant 0
        End Acq B[0,1], interlace, even field, Start Acq FS_MEM,interlace, evenfield
        FS_P7 = FS0, FS0 = P4, FS1 = P9, FS2 = P10
        All Pipelines Normal
        Select Primary output, Write Enable on all
        Horizontal and vertical position at 0
*/
setByte(bFs0C0, (BYTE) 0x03);

setByte(bFs0K0, (BYTE) 0x00);

setByte(bFs0Gate, (BYTE) 0x08);

setByte(bFs0C1, (BYTE) 0x24);

setByte(bFs0XmtGate, (BYTE) 0x00);

setByte(bFs0C2, (BYTE) 0x70);

setByte(bFs0MemWEna, (BYTE) 0x00);

setShort(wFs0WordGHP, (SHORT) 0x0000);

setShort(wFs0WordGVP, (SHORT) 0x0000);

setShort(wFs0ByteGHP, (SHORT) 0x0000);

setShort(wFs0ByteGVP, (SHORT) 0x0000);

} /* End of initFS0() */

#ifdef __CPLUSPLUS
void initFS1(void)
#else
void initFS1()
#endif
{
/* Configure FS1 as follows:
        60Hz, Interlace, Mode 4
        Constant 0
        End Acq B[0,1], interlace, even field, Start Acq FS_MEM,interlace, evenfield
        FS_P7 = FS0, FS0 = P4, FS1 = P9, FS2 = P10
        All Pipelines Normal
        Select Primary output, Write Enable on all
        Horizontal and vertical position at 0

*/
setByte(bFs1C0, (BYTE) 0x03);
```

```
setByte(bFs1K0, (BYTE) 0x00);

setByte(bFs1Gate, (BYTE) 0x08);

setByte(bFs1C1, (BYTE) 0x24);

setByte(bFs1XmtGate, (BYTE) 0x00);

setByte(bFs1C2, (BYTE) 0x70);

setByte(bFs1MemWEna, (BYTE) 0x00);

setShort(wFs1WordGHP, (SHORT) 0x0000);

setShort(wFs1WordGVP, (SHORT) 0x0000);

setShort(wFs1ByteGHP, (SHORT) 0x0000);

setShort(wFs1ByteGVP, (SHORT) 0x0000);

} /* End of initFS1() */

#ifdef __CPLUSPLUS
void initMaxSP(void)
#else
void initMaxSP()
#endif
{
/* Configure the MaxSP as follows:
        Mult unsigned, signed/unsigned round disable,PCRA always, select max (MM),
                select unsigned Min/Max (MMLA), select primary P4,P5,P6,P7,P8
        0 Y Mult Const Register
        0 ALU const Register
        Disable shifter, disable clipper, Delay 1 line = 512 pixels=0x200
        Select Mult as ALU A input, Select P6/P7 as ALU B input, ALU op all zeros,
                disable carry, noninverted Min/Max, P4 as X, P6 as Y,
                Transform address = P4 + upper nibble P5

        Select Mult as ALU A input , Select P6/P7 as ALU B input, ALU op all zeros,
                disable carry, noninverted Min/Max, P4 as X, P6 as Y,
                Transform address = P4 + upper nibble P5
*/
setShort(wMAXSPC0, (SHORT) 0x0020);
setShort(wMAXSPK0K1, (SHORT) 0x0000);
setShort(wMAXSPK2, (SHORT) 0x0000);
setShort(wMAXSPC1, (SHORT) 0x0200);
setShort(wMAXSPMoscA, (SHORT) 0x0000);
setShort(wMAXSPMoscB, (SHORT) 0x0000);

} /* End of initMaxSP() */

#ifdef __CPLUSPLUS
void freezeFS0(void)
#else
void freezeFS0()
```

```
#endif
{
/* Stop the acquisition on FS0_Byte_Mem to freeze the video */

BYTE   gate, freezeMask = 0xf7;

gate = getByte(bFs0Gate);

setByte(bFs0Gate, gate & freezeMask);


} /* End of freezeFS0() */


#ifdef __CPLUSPLUS
void  liveFS0(void)
#else
void liveFS0()
#endif
{
/* Start the acquisition on FS0_Byte_Mem  */

BYTE   gate, liveMask = 0x08;

gate = getByte(bFs0Gate);

setByte(bFs0Gate, gate | liveMask);


} /* End of liveFS0() */
```

```
/*
Summary:
Bridge Inspector Project
IDS Image Processor
Loads the spread functions into memory.

Description:
Since we can load two spread functions, and we currently have only two, we
will load the cone function into SP0 and the exp10 into SP1

See Also:
S. Strang's sa_loadram_sr.c (the basis for this program)

*/

#include "idsControl.h"
#include "idsParse.h"

#include "frArg.h"
int          error = 0;
int          spsel;
unsigned short *spread;

idsLoadram ()
{
   extern int     status;

   extern unsigned short *s_cone, *s_del2g_09, *s_exp_10, *s_exp_05;

   register int   i = 0;
   register       j;
   register       shift = 0;
   unsigned char  zeros, ones;

   status = LOADING;

   spread = s_cone;
   spsel = 0;
   for (i = 0; i < 8; i++)
   {
         printf ("Loading cone to Board %d spread 0.\n", i);
         loadboard (i);
   }

   if (error != 0)
   {
         printf ("Loading errors = %d\n", error);
   }


   /***** Load the exp10 spread to spsel 1 *********/
   error = 0;
   spread = s_exp_10;
   spsel = 1;

   for (i = 0; i < 8; i++)
```

```
    {
        printf ("Loading exp 10 to Board %d spread 1.\n", i);
        loadboard (i);
    }

    if (error != 0)
    {
        printf ("Loading errors = %d\n", error);
    }


    status = READY;

}

/*****************************************************************************/

int         loadboard (board)
int         board;
{
    int         data0, data1;
    int         mem, x, y;
    unsigned char  *memptr, *video, memdata;
    int         pixel;
    unsigned char  *base;
    unsigned short *ctrl, ctrl_shad;

    switch (board)
    {
     case 0:
         base = (unsigned char *) FPPSR_0;
         break;
     case 1:
         base = (unsigned char *) FPPSR_1;
         break;
     case 2:
         base = (unsigned char *) FPPSR_2;
         break;
     case 3:
         base = (unsigned char *) FPPSR_3;
         break;
     case 4:
         base = (unsigned char *) FPPSR_4;
         break;
     case 5:
         base = (unsigned char *) FPPSR_5;
         break;
     case 6:
         base = (unsigned char *) FPPSR_6;
         break;
     case 7:
         base = (unsigned char *) FPPSR_7;
         break;
    }

    memptr = (unsigned char *) base;
```

```
memptr += 0x41;                 /* Make pointer point to first memory
                                 * location */
ctrl = (unsigned short *) base;
video = (unsigned char *) base;
video += 0x21;
ctrl_shad = 0;
if (spsel == 0)
      ctrl_shad = ctrl_shad & ~SPSEL_1;
else
      ctrl_shad = ctrl_shad | SPSEL_1;
ctrl_shad = ctrl_shad | NORMAL;
ctrl_shad = ctrl_shad | VIDEO_REG;
ctrl_shad = ctrl_shad | AB_AIN;
*ctrl = ctrl_shad;
for (pixel = 0x00; pixel <= 0xff; pixel++)
{
      *video = pixel;
      ctrl_shad = ctrl_shad | ANINE_1;
      ctrl_shad = ctrl_shad & ~ATEN_1;
      *ctrl = ctrl_shad;
      for (mem = 0; mem < 16; mem++)
      {
         x = 14 - 2 * mem + 16 * (mem / 8);
         y = 15 - board * 2 - (mem / 8);
         data0 = ((int) spread[pixel * 256 + y * 16 + x]) & 0x000f;
         data1 = ((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x000f;
         *(memptr + mem * 2) = (unsigned char) (data1 + (data0 << 4));          /* Low */
      }

      ctrl_shad = ctrl_shad & ~ANINE_1;
      ctrl_shad = ctrl_shad & ~ATEN_1;
      *ctrl = ctrl_shad;
      for (mem = 0; mem < 16; mem++)
      {
         x = 14 - 2 * mem + 16 * (mem / 8);
         y = 15 - board * 2 - (mem / 8);
         data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0x00f0) / 16;
         data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x00f0) / 16;
         *(memptr + mem * 2) = (unsigned char) (data1 + (data0 << 4));          /* Mid Low */
      }
      ctrl_shad = ctrl_shad | ATEN_1;
      ctrl_shad = ctrl_shad | ANINE_1;
      *ctrl = ctrl_shad;
      for (mem = 0; mem < 16; mem++)
      {
         x = 14 - 2 * mem + 16 * (mem / 8);
         y = 15 - board * 2 - (mem / 8);
         data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0x0f00) / 256;
         data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x0f00) / 256;
         *(memptr + mem * 2) = (unsigned char) (data1 + (data0 << 4));          /* Mid High */
      }
      ctrl_shad = ctrl_shad | ATEN_1;
      ctrl_shad = ctrl_shad & ~ANINE_1;
      *ctrl = ctrl_shad;
      for (mem = 0; mem < 16; mem++)
      {
```

```
            x = 14 - 2 * mem + 16 * (mem / 8);
            y = 15 - board * 2 - (mem / 8);
            data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0xf000) / 4096;
            data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0xf000) / 4096;
            *(memptr + mem * 2) = (unsigned char) (data1 + (data0 << 4));          /* High */
        }
    }
    for (pixel = 0x00; pixel <= 0xff; pixel++)
    {
        *video = pixel;
        ctrl_shad = ctrl_shad | ANINE_1;
        ctrl_shad = ctrl_shad & ~ATEN_1;
        *ctrl = ctrl_shad;
        for (mem = 0; mem < 16; mem++)
        {
            x = 14 - 2 * mem + 16 * (mem / 8);
            y = 15 - board * 2 - (mem / 8);
            data0 = ((int) spread[pixel * 256 + y * 16 + x]) & 0x000f;
            data1 = ((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x000f;
            memdata = *(memptr + mem * 2);
            if (memdata != (unsigned char) (data1 + (data0 << 4)))          /* Low */
                error++;
        }

        ctrl_shad = ctrl_shad & ~ANINE_1;
        ctrl_shad = ctrl_shad & ~ATEN_1;
        *ctrl = ctrl_shad;
        for (mem = 0; mem < 16; mem++)
        {
            x = 14 - 2 * mem + 16 * (mem / 8);
            y = 15 - board * 2 - (mem / 8);
            data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0x00f0) / 16;
            data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x00f0) / 16;
            memdata = *(memptr + mem * 2);
            if (memdata != (unsigned char) (data1 + (data0 << 4)))          /* Mid Low */
                error++;
        }
        ctrl_shad = ctrl_shad | ATEN_1;
        ctrl_shad = ctrl_shad | ANINE_1;
        *ctrl = ctrl_shad;
        for (mem = 0; mem < 16; mem++)
        {
            x = 14 - 2 * mem + 16 * (mem / 8);
            y = 15 - board * 2 - (mem / 8);
            data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0x0f00) / 256;
            data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0x0f00) / 256;
            memdata = *(memptr + mem * 2);
            if (memdata != (unsigned char) (data1 + (data0 << 4)))          /* Mid High */
                error++;
        }
        ctrl_shad = ctrl_shad | ATEN_1;
        ctrl_shad = ctrl_shad & ~ANINE_1;
        *ctrl = ctrl_shad;
        for (mem = 0; mem < 16; mem++)
        {
            x = 14 - 2 * mem + 16 * (mem / 8);
```

```
                y = 15 - board * 2 - (mem / 8);
                data0 = (((int) spread[pixel * 256 + y * 16 + x]) & 0xf000) / 4096;
                data1 = (((int) spread[pixel * 256 + y * 16 + x + 1]) & 0xf000) / 4096;
                memdata = *(memptr + mem * 2);
                if (memdata != (unsigned char) (data1 + (data0 << 4)))          /* High */
                    error++;
            }
        }
    }
```

```
#include "idsControl.h"
#include "frame.h"
#include "digi.h"
#include "oneShot.h"

idsOne()
{
  ids_dgSelDtoASrc(DG_P8DISP);

  ids_fs12AcqStrt(0,FS_12EVEN);
  ids_fs0AcqStrt(0,FS_0EVEN);
  ids_fsSelOutSrc(0,FS_1OUT);
  ids_fs2Input(0,FS_1P10);
  ids_fs1Input(0,FS_1P9);
  ids_fs0Input(0,FS_0P4);
  ids_fsVIntlMd(0,FS_INTL);



  ids_dgWaitOdd();

  ids_dgWaitEven();

/* Set board 0, fs0, Mode 3 */
  ids_fs0VResMd(0,FS_LORES);
  ids_fs0TimeMd(0,FS_PROCESS);

  ids_fs0HPipMd(0,FS_RELEASE);
  ids_fs0VPipMd(0,FS_RELEASE);

  ids_hjP5InpMd(DQ_PRIMARY);

/* Set board 0, fs1, Mode 3 */
  ids_fs12VResMd(0,FS_LORES);
  ids_fs12TimeMd(0,FS_PROCESS);


/* Get a frame the ids result into, board 0, framestore 1 */
  ids_fs12AcquireMd(0,FS_ACQUIRE);
  ids_fs12AcquireMd(0,FS_FREEZE);

  ids_dgWaitFld(2);  /* Must wait 2 frames to align the output frame with input frame */


/* Set board 0, fs1, Mode 4 */
  ids_fs12VResMd(0,FS_HIRES);
  ids_fs12TimeMd(0,FS_NORMAL);

/* Select board 0 frame 1 output  */
  ids_fs12IncScroll(0,SCROLL_INC);  /* The pan and scroll values needed to align input and
output */
  ids_fs12IncPan(0,PAN_INC);  /* These values are derived using trial images */
  ids_dgP8InpMd(DQ_PRIMARY);

/* Set board 0, fs0, Mode 4 */
  ids_fs0VResMd(0,FS_HIRES);
```

```
    ids_fs0TimeMd(0,FS_NORMAL);

    ids_fs0HPipMd(0,FS_NORM);
    ids_fs0VPipMd(0,FS_NORM);

}
```

```
/* idsParse.c - */
/* This function parses the messages sent from the host computer. */
/* RETURNS:  int action code and pointers to parameters. */
/*
 * modification history --------------------
 *
 */



#include <vxWorks.h>
#include <strLib.h>

/*******
#include <socket.h>
#include <in.h>
#include "etherDispatch.h"
#include <taskLib.h>
***********/

#include "idsControl.h"
#include "idsParse.h"

/********
#include "frame.h"
#include "digi.h"
********/


int         idsParse (str, param)
char        *str;         /* NULL-terminated string to parse */
int         param[];      /* Parameters to the action */
{
   int          i;
   char          tokens[10][20];
   char         *pChar;
   int          tokenNum = 0, tokenIndex = 0;
   int          scanning = FALSE;

   /*
    * Break the string into a array of sub-strings. Each sub-string will
    * be a token.
    */
   pChar = str;             /* Init pChar to point to beginning of
                                    * string to parse */

   while (*pChar != NULL)
   {
         if (separator (*pChar))
         {                          /* If *pChar is a separator, then we are
                                     * at end of token. */
            if (scanning)
            {
                  tokens[tokenNum][tokenIndex] = NULL; /* Terminate token with
                                                        * NULL */
                  tokenNum++;
```

```
                tokenIndex = 0;
                scanning = FALSE;
            }
            pChar++;

        }
        else
        {
            scanning = TRUE;
            tokens[tokenNum][tokenIndex] = *pChar;
            tokenIndex++;
            pChar++;            /* Go to next char */
        }

}
/*
 * If we were scanning when the end of the input string was reached,
 * terminate the last token with a NULL.
 */
if (scanning)
{
        tokens[tokenNum][tokenIndex] = NULL; /* Terminate token with
                                              * NULL */
        tokenNum++;
        scanning = FALSE;
}


for (i = 1; i < tokenNum; i++)
{
        /* Set the parameters */
        /*
         * The parameters will all be integers. Look at the first two char
         * to determine if it is hex
         */
        if (strncmp (&tokens[i][0], "0x", 2) == 0)
            sscanf (&tokens[i][0], "%x", &param[i - 1]);
        else
            sscanf (&tokens[i][0], "%d", &param[i - 1]);
        /** printf("\ntoken %d = %s      param = %d \n", i, &tokens[i][0], &param[i-1]);  ***/
}

/* Compare the first token string with the strings of the valid tokens */
if (strcmp (&tokens[0][0], "acquire") == 0)
        return (ACQUIRE);
if (strcmp (&tokens[0][0], "freeze") == 0)
        return (FREEZE);
if (strcmp (&tokens[0][0], "init") == 0)
        return (INIT);
if (strcmp (&tokens[0][0], "decomp") == 0)
        return (DECOMP);
if (strcmp (&tokens[0][0], "load") == 0)
        return (LOAD);
if (strcmp (&tokens[0][0], "histo") == 0)
        return (HISTO);
if (strcmp (&tokens[0][0], "spread") == 0)
```

```
        return (SPREAD);
    if (strcmp (&tokens[0][0], "realtime") == 0)
        return (REALTIME);
    if (strcmp (&tokens[0][0], "endreal") == 0)
        return (ENDREAL);
    if (strcmp (&tokens[0][0], "oneshot") == 0)
        return (ONESHOT);
    if (strcmp (&tokens[0][0], "sendfrm") == 0)
        return (SENDFRM);
    if (strcmp (&tokens[0][0], "getfrm") == 0)
        return (GETFRM);
    if (strcmp (&tokens[0][0], "status") == 0)
        return (STATUS);
    /* If no matches occur, return invalid token. */
    return (INVALID);

}



/* Compare the character to a set of separator characters */
int         separator (aChar)
char        aChar;
{
    switch (aChar)
    {
     case ' ':
        return (TRUE);
        break;
     case ',':
        return (TRUE);
        break;
     case '(':
        return (TRUE);
        break;
     case ')':
        return (TRUE);
        break;

     default:
        return (FALSE);
    }
}



int         test1 ()
{
    int         *pa;
    int         result, i;

    pa = (int *) malloc (10 * sizeof (int));

    result = idsParse ("acquire(1,2,3,4,5,123,456)", pa);
    for (i = 0; i < 7; i++)
```

```
    {
        printf ("\ntoken  = %d      pa = %d \n", result, pa[i]);
    }

    free (pa);
}


int         test2 ()
{
    int         pa[10];
    int         result, i;


    result = idsParse ("init(1,2,3,4,5,123,456)", pa);
    for (i = 0; i < 7; i++)
    {
        printf ("\ntoken  = %d      pa = %d \n", result, pa[i]);
    }

}
```

```
/* idsRTLoop.c
* This function is intended to run as a task so that it can be terminated by a
* higher priority task. Otherwise, it will run forever.
*/



#include "idsControl.h"

/*** #include "ids_bases.h"  ***/
#include "frame.h"
#include "digi.h"

#include "idsParse.h"

/* See also: idsOne.c */

idsRTLoop(p1)
int p1;  /* The amount of horizontal shift, derived using test images */
{
   extern int status;

  ids_dgSelDtoASrc(DG_P8DISP);

  ids_fs12AcqStrt(0,FS_12EVEN);
  ids_fs0AcqStrt(0,FS_0EVEN);
  ids_fsSelOutSrc(0,FS_1OUT);
  ids_fs2Input(0,FS_2P10);
  ids_fs1Input(0,FS_1P9);
  ids_fs0Input(0,FS_0P4);
  ids_fsVIntlMd(0,FS_INTL);

  ids_fs12AcqStrt(1,FS_12EVEN);
  ids_fs0AcqStrt(1,FS_0EVEN);
  ids_fsSelOutSrc(1,FS_1OUT);
  ids_fs2Input(1,FS_1P10);
  ids_fs1Input(1,FS_1P9);
  ids_fs0Input(1,FS_0P4);
  ids_fsVIntlMd(1,FS_INTL);

  ids_dgWaitOdd();
  ids_dgWaitEven();

  status = RUNNING;

  do {
    ids_dgWaitFld(2);
/* Set board 1, fs1, Mode 4 */
    ids_fs12VResMd(1,FS_HIRES);
    ids_fs12TimeMd(1,FS_NORMAL);

/* Select board 1 frame 1 output  */
    ids_fs12IncScroll(1,5);
    ids_fs12IncPan(1,p1);
    ids_dgP8InpMd(DQ_ALTERN);
```

```
/* Set board 1, fs0, Mode 4 */
    ids_fs0VResMd(1,FS_HIRES);
    ids_fs0TimeMd(1,FS_NORMAL);

    ids_fs0HPipMd(1,FS_NORM);
    ids_fs0VPipMd(1,FS_NORM);

/* Set board 0, fs0, Mode 3 */
    ids_fs0VResMd(0,FS_LORES);
    ids_fs0TimeMd(0,FS_PROCESS);

    ids_fs0HPipMd(0,FS_RELEASE);
    ids_fs0VPipMd(0,FS_RELEASE);
    ids_hjP5InpMd(DQ_PRIMARY);

/* Set board 0, fs1, Mode 3 */
    ids_fs12VResMd(0,FS_LORES);
    ids_fs12TimeMd(0,FS_PROCESS);

/* Get a frame, board 1, framestore 0 */
    ids_fs0AcquireMd(1,FS_ACQUIRE);
    ids_fs0AcquireMd(1,FS_FREEZE);

/* Get a frame, board 0, framestore 1 */
    ids_fs12AcquireMd(0,FS_ACQUIRE);
    ids_fs12AcquireMd(0,FS_FREEZE);

    ids_dgWaitFld(2);

/* Set board 0, fs1, Mode 4 */
    ids_fs12VResMd(0,FS_HIRES);
    ids_fs12TimeMd(0,FS_NORMAL);

/* Select board 0 frame 1 output */
    ids_fs12IncScroll(0,5);
    ids_fs12IncPan(0,p1);
    ids_dgP8InpMd(DQ_PRIMARY);

/* Set board 0, fs0, Mode 4 */
    ids_fs0VResMd(0,FS_HIRES);
    ids_fs0TimeMd(0,FS_NORMAL);

    ids_fs0HPipMd(0,FS_NORM);
    ids_fs0VPipMd(0,FS_NORM);

/* Set board 1, fs0, Mode 3 */
    ids_fs0VResMd(1,FS_LORES);
    ids_fs0TimeMd(1,FS_PROCESS);
    ids_fs0HPipMd(1,FS_RELEASE);
    ids_fs0VPipMd(1,FS_RELEASE);

    ids_hjP5InpMd(DQ_ALTERN);

/* Set board 1, fs1, Mode 3 */
    ids_fs12VResMd(1,FS_LORES);
    ids_fs12TimeMd(1,FS_PROCESS);
```

```
/* Get a frame, board 0, framestore 0 */
    ids_fs0AcquireMd(0,FS_ACQUIRE);
    ids_fs0AcquireMd(0,FS_FREEZE);

/* Get a frame, board 1, framestore 1 */
    ids_fs12AcquireMd(1,FS_ACQUIRE);
    ids_fs12AcquireMd(1,FS_FREEZE);


  } while(1);
}
```

```
/* idsSendFrm.c
* Sends a frame over the net from  Board 0 FS1
*/

#include <vxWorks.h>
#include <socket.h>
#include <in.h>
#include "etherDispatch.h"
#include <taskLib.h>

#include "idsControl.h"
#include "frame.h"
#include "digi.h"
#include "idsParse.h"



int         idsSendFrm (imExSock, lineBuf, B1, B2, B3, B4, sBuf, imageBuf)
int             imExSock;
pBYTE lineBuf, B1, B2, B3, B4, sBuf, imageBuf;

{

   LONG         lineCount, i;
   pSHORT     sPix;



   printf ("Sending image... ");
   for (lineCount = 0; lineCount < 512; lineCount += 4)
   {
         for (i = 0; i < 512; i++)
         {
            sPix = (pSHORT) (sFs0Mem1 + ((lineCount * 512) + i));
            B1[i] = (BYTE) (getShort (sPix) >> 8);
         }
         for (i = 0; i < 512; i++)
         {
            sPix = (pSHORT) (sFs0Mem1 + (((lineCount + 1) * 512) + i));
            B2[i] = (BYTE) (getShort (sPix) >> 8);
         }
         for (i = 0; i < 512; i++)
         {
            sPix = (pSHORT) (sFs0Mem1 + (((lineCount + 2) * 512) + i));
            B3[i] = (BYTE) (getShort (sPix) >> 8);
         }
         for (i = 0; i < 512; i++)
         {
            sPix = (pSHORT) (sFs0Mem1 + (((lineCount + 3) * 512) + i));
            B4[i] = (BYTE) (getShort (sPix) >> 8);
         }

         for (i = 0; i < 256; i++)
            lineBuf[i] = B1[i];
         for (i = 0; i < 256; i++)
```

```
            lineBuf[i + 256] = B3[i];
        hShift (lineBuf, sBuf, HSHIFT);
        send (imExSock, sBuf, 512, 0);
        for (i = 0; i < 256; i++)
            lineBuf[i] = B2[i];
        for (i = 0; i < 256; i++)
            lineBuf[i + 256] = B4[i];
        hShift (lineBuf, sBuf, HSHIFT);
        send (imExSock, sBuf, 512, 0);
        for (i = 0; i < 256; i++)
            lineBuf[i] = B1[i + 256];
        for (i = 0; i < 256; i++)
            lineBuf[i + 256] = B3[i + 256];
        hShift (lineBuf, sBuf, HSHIFT);
        send (imExSock, sBuf, 512, 0);
        for (i = 0; i < 256; i++)
            lineBuf[i] = B2[i + 256];
        for (i = 0; i < 256; i++)
            lineBuf[i + 256] = B4[i + 256];
        hShift (lineBuf, sBuf, HSHIFT);
        send (imExSock, sBuf, 512, 0);
    }

    printf ("Done sending image\n");

}


/*****************************************************************************/
hShift (lineBuf, sBuf, shift)
pBYTE       lineBuf, sBuf;
int         shift;
{
    int         i;

    if (shift < 0)
    {
        for (i = 0; i < 512 + shift; i++)
            sBuf[i - shift] = lineBuf[i];
        for (i = 512 + shift; i < 512; i++)
            sBuf[i - (512 + shift)] = lineBuf[i];
    }

    if (shift > 0)
    {
        for (i = 0; i < 512 - shift; i++)
            sBuf[i] = lineBuf[i + shift];
        for (i = 512 - shift; i < 512; i++)
            sBuf[i] = lineBuf[i - (512 - shift)];

    }

    if (shift == 0)
    {
        for (i = 0; i < 512; i++)
            sBuf[i] = lineBuf[i];
```

```
    }
}
```

```
/*
Summary:
Bridge Inspector Project
IDS Image Processor
Command processing shell.

Description:
This shell processes commands that come over the ether net as well as those that
come from the control console.

See Also:
S. Strang's shell.c (the basis for this program)

*/
#include <vxWorks.h>
#include <taskLib.h>

#include "idsControl.h"
#include "frame.h"
#include "digi.h"

unsigned short *s_cone,*s_del2g_09,*s_exp_10,*s_exp_05;
extern int idsRTLoop();

/* IDS operation status */
int       status; /* This will be extern to all other functions that use it. */


void idsShell( )
{
  static int mode=0;
  char select, c;
  int     exit = FALSE;
  FUNCPTR     pfunc;
   int   loopTask=0;

 pfunc = (FUNCPTR) idsRTLoop;


  /* Allocate enough memory for the spread functions. */
  s_cone=(unsigned short *)malloc(128*1024);
  s_del2g_09=(unsigned short *)malloc(128*1024);
  s_exp_10=(unsigned short *)malloc(128*1024);
  s_exp_05=(unsigned short *)malloc(128*1024);
  if(!s_cone || !s_del2g_09 || !s_exp_10 || !s_exp_05){
    printf(" OUT OF MEMORY\n");
    exit = TRUE;
  }

  do {
    printf(" SELECT (i,t,a,z,m,l,d,0,1,b,e,o,q): ");
   select = (char) getchar();
    /* Filter out \n and \r */
   if(((c = (char) getchar()) != '\n') & (c != '\r'))  select = c;

   printf("\n");
```

```
 /* Convert Upper case to lower*/
switch(select) {
    case 'a':
     case 'A':
        ids_fs0AcquireMd(0,FS_ACQUIRE);
        ids_fsSelOutSrc(0,FS_0OUT);
        ids_dgP8InpMd(DQ_PRIMARY);

        break;
    case 'z':
     case 'Z':
        ids_fs0AcquireMd(0,FS_FREEZE);
        ids_fsSelOutSrc(0,FS_0OUT);
        ids_dgP8InpMd(DQ_PRIMARY);

        break;

   case 'q':
   case 'Q': break;

     case 'i' :
     case 'I':
        printf(" Initialize       \n ");
        idsInitdq();
              break;
     case 't' :
     case 'T':
        printf(" Test ram         \n");
        idsTestram();
              break;
     case 'm' :
     case 'M':
        printf(" Modify histogrammer \n");
        /*** idsHisto2(histoLevel, aluConst, shift)  ***/
        /** OLD CASE case '7' :            *(cptr+0x21)=0xbf;
                 msWtAluHi(0x40);
                 msShift(7);
        **************/

        idsHisto2(0xbf, 0x40, 7);
              break;
     case 'l' :
     case 'L':
        printf(" Loadram          \n");
                idsLoadram();
              break;
     case 'd' :
     case 'D':
        printf(" Decompress Spreads  \n");
                idsDecomp();
              break;
      case '0':
        printf("Select spread 0\n");
        idsFrspsel(0);
        break;
```

```
        case '1':
          printf("Select spread 1\n");
          idsFrspsel(1);
          break;
        case 'b' :
        case 'B':
          /* Spawn a VxWorks Task */
          /* Make sure the priority is low so we can kill it using another higher priority task */
#define IDS_PRIORITY 10
#define IDS_STACK_SIZE      1024
 loopTask = taskSpawn("idsLoopTask", IDS_PRIORITY,  VX_DEALLOC_STACK | VX_STDIO,
          IDS_STACK_SIZE, pfunc , -24);
if(loopTask != ERROR) taskActivate(loopTask);
          printf(" Begin Processing   \n");
          /*** idsRTLoop();  ****/
                break;
        case 'e' :
        case 'E':
          printf(" End Processing     \n");
          if(loopTask != ERROR) taskDelete(loopTask);
          break;
        case 'o' :
        case 'O':
          printf(" One-shot          \n");
          idsOne();
                  break;


        default:
          printf("Unknown command \n");
          break;
        }
      } while((select   != 'q') & (select != 'Q'));

    printf("      QUIT \n");

    free(s_cone);
    free(s_del2g_09);
    free(s_exp_10);
    free(s_exp_05);


  }
```

```
/*
Summary:
Bridge Inspector Project
IDS Image Processor
Test the spread function memory.

Description:
This loads a pattern into the RAM and reads it back.

See Also:
S. Strang's sa_testram_sr.c (the basis for this program)

*/

#include "idsControl.h"

#include "frArg.h"

#define MEM_OFFSET 0x41
#define VIDEO_OFFSET        0x21

void idsTestram()
{
  BYTE *base;
  BYTE mem;
  BYTE *memptr,*video,memdata;
  int pixel,data,i;
  SHORT *ctrl,ctrl_shad;
  int errors[4] , totalErr = 0;
  char ch, ans;

  for(i=0;i<4;i++) errors[i] = 0;

 for(base=(BYTE*)FPPSR_0;base<=(BYTE*)FPPSR_7;base+=0x100){

    memptr=(BYTE *)base;
    memptr+=MEM_OFFSET;  /* Make pointer point to first memory location */
    ctrl=(SHORT *)base;
    video=(BYTE *)base;
    video+=VIDEO_OFFSET;
    *ctrl=0;
    /* Can't read the control register so create a shadow of it */
    ctrl_shad=0;
    ctrl_shad= ctrl_shad | (SHORT)NORMAL;
    ctrl_shad= ctrl_shad | (SHORT)VIDEO_REG ;
    ctrl_shad= ctrl_shad | (SHORT)AB_AIN;
    *ctrl=ctrl_shad;
    for(data=0x00;data<=255;data+=0x55) { /* data = 00, 0x55, 0xAA, 0xFF */
      for(pixel=0x00;pixel<=0xff;pixel++) {
        *video=pixel;
        ctrl_shad=ctrl_shad &(SHORT)~ANINE_1;
        ctrl_shad=ctrl_shad &(SHORT)~ATEN_1;
        ctrl_shad=ctrl_shad &(SHORT)~SPSEL_1;
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
```

```
                memdata= *(memptr+mem*2) ;
                if(memdata!=(BYTE)data){
                  errors[data/0x55]++;
                   printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x,  Expected val = %x,
Actual val = %x\n",
          base, pixel, ctrl_shad, mem, data, memdata);
          totalErr++;
          if (totalErr > 100) {
                    ans = 'n';
                    printf("Continue ? ");
                    ans = (char) getchar();
                    printf("%c", (char)getchar());
                    if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
          }
              }
          }

          ctrl_shad=ctrl_shad | ANINE_1;
          *ctrl=ctrl_shad;
          for(mem=0;mem<=15;mem++){
              *(memptr+mem*2) = (BYTE)data;
              memdata= *(memptr+mem*2) ;
              if(memdata!=(BYTE)data){
                errors[data/0x55]++;
                 printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x,  Expected val = %x,
Actual val = %x\n",
          base, pixel, ctrl_shad, mem, data, memdata);
          totalErr++;
          if (totalErr > 100) {
                    ans = 'n';
                    printf("Continue ? ");
                    ans = (char) getchar();
                    printf("%c", (char)getchar());
                    if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
          }


              }
          }

          ctrl_shad=ctrl_shad | ATEN_1;
          *ctrl=ctrl_shad;
          for(mem=0;mem<=15;mem++){
              *(memptr+mem*2) = (BYTE)data;
              memdata= *(memptr+mem*2) ;
              if(memdata!=(BYTE)data){
                errors[data/0x55]++;
                 printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
          base, pixel, ctrl_shad, mem, data, memdata);
          totalErr++;
          if (totalErr > 100) {
                    ans = 'n';
                    printf("Continue ? ");
                    ans = (char) getchar();
                    printf("%c",(char)getchar());
```

```
                if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
        }


            }
        }


        ctrl_shad=ctrl_shad &~ANINE_1;
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
            memdata= *(memptr+mem*2) ;
            if(memdata!=(BYTE)data){
              errors[data/0x55]++;
               printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
        base, pixel, ctrl_shad, mem, data, memdata);
        totalErr++;
        if (totalErr > 100) {
                ans = 'n';
                printf("Continue ? ");
                ans = (char) getchar();
                printf("%c",(char)getchar());
                if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
        }


            }
        }


        ctrl_shad=ctrl_shad | SPSEL_1;
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
            memdata= *(memptr+mem*2) ;
            if(memdata!=(BYTE)data){
              errors[data/0x55]++;
               printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
        base, pixel, ctrl_shad, mem, data, memdata);
        totalErr++;
        if (totalErr > 100) {
                ans = 'n';
                printf("Continue ? ");
                ans = (char) getchar();
                printf("%c",(char)getchar());
                if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
        }


            }
        }

        ctrl_shad=ctrl_shad | ANINE_1;
```

```
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
            memdata= *(memptr+mem*2) ;
            if(memdata!=(BYTE)data){
              errors[data/0x55]++;
               printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
        base, pixel, ctrl_shad, mem, data, memdata);
        totalErr++;
        if (totalErr > 100) {
                ans = 'n';
                printf("Continue ? ");
                ans = (char) getchar();
                printf("%c",(char)getchar());
                if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
        }


            }
        }


        ctrl_shad=ctrl_shad &~ATEN_1;
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
            memdata= *(memptr+mem*2) ;
            if(memdata!=(BYTE)data){
              errors[data/0x55]++;
               printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
        base, pixel, ctrl_shad, mem, data, memdata);
        totalErr++;
        if (totalErr > 100) {
                ans = 'n';
                printf("Continue ? ");
                ans = (char) getchar();
                printf("%c",(char)getchar());
                if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
        }


            }
        }

        ctrl_shad=ctrl_shad &~ANINE_1;
        *ctrl=ctrl_shad;
        for(mem=0;mem<=15;mem++){
            *(memptr+mem*2) = (BYTE)data;
            memdata= *(memptr+mem*2) ;
            if(memdata!=(BYTE)data){
              errors[data/0x55]++;
               printf("Error: Base = %x, Pixel = %x,  Ctrl = %x, mem = %x, Expected val = %x,
Actual val = %x\n",
        base, pixel, ctrl_shad, mem, data, memdata);
```

```
            totalErr++;
            if (totalErr > 100) {
                    ans = 'n';
                    printf("Continue ? ");
                    ans = (char) getchar();
                    printf("%c",(char)getchar());
                    if(ans != 'y' && ans != 'Y') return; else totalErr = 0;
            }


               }
            }

         } /* End pixel for-loop */
       } /* End data for-loop */
     } /* End base for-loop */


/*** Just use the stdio printf function *****/
     printf("\nError Report \n");
     if(errors[0]||errors[1]||errors[2]||errors[3]) {
       printf(" 0x00 : %d \n  ",errors[0]);
       printf(" 0x55 : %d \n  ",errors[1]);
       printf(" 0xAA : %d \n  ",errors[2]);
       printf(" 0xFF : %d \n  ",errors[3]);
     } else printf("No errors detected. \n");

}
```

```
/*
$File, date, etc.$
$ModHeadBegin$

$Revision$
$LogFile$ (c) 1994, ODETICS, INC.

SUMMARY:

Bridge Inspector Project
IDS Image Processor

DESCRIPTION:

This file contains the IDS Image Processor interrupt handlers.

SEE ALSO:

idsControl.h

HISTORY:

$Log$


$ModHeadEnd$
*/

#include "idsControl.h"

/*************************** Interrupt Support Functions ***************************/

void enableDigiMaxInterrupt(void)
{
BYTE   timing, enableMask = 0x08;

timing = getByte(bDgSTiming);
setByte(bDgSTiming, timing | enableMask);

} /* End enableDigiMaxInterrupt() */


void disableDigiMaxInterrupt(void)
{
BYTE   timing, disableMask = 0xf7;

timing = getByte(bDgSTiming);
setByte(bDgSTiming, timing & disableMask);

} /* End disableDigiMaxInterrupt() */



/******************************* Interrupt Handler Functions ***********************/
LOCAL int interruptHandlerDigiMax(int parameter)
{
```

```
/* In this example, count 10 fields then fields for 10 fields then back to live for 10
fields and repeat.
  The call is invoked with a parameter but is not used.
*/

static int          fieldCount = 0, live = 1;
int       level;

disableDigiMaxInterrupt();

if (fieldCount < 10)
   fieldCount++;

else if (live == 1) {
   live = 0;
   fieldCount = 0;
   freezeFS0();
  } else {
     live = 1;
     fieldCount = 0;
     liveFS0();
  }

enableDigiMaxInterrupt();

} /* End of interruptHandlerDigiMax() */

int nothing(int p) {
}


void installHandlerDigiMax(void)
{
/* Install the DigiMax interrupt handler. */

if (OK == intConnect(INUM_TO_IVEC (vectorDigiMax), (FUNCPTR) interruptHandlerDigiMax,
0)) {
        printf("Handler installed \n");
        enableIRQ5();
        enableDigiMaxInterrupt();
}else
        printf("Handler could not be installed");

} /* End of installHandlerDigiMax() */


void enableIRQ5(void)
{
/* The DigiMax is configured to interrupt at level 5 */
/* The control register on the iv3234 must be changed to enable the processor to respond */
setByte((pBYTE) 0xfffc0017, (BYTE) 0x7d);

} /* End of enableIRQ5() */
```

```
/*
$File, date, etc.$
$ModHeadBegin$

$Revision$
$LogFile$ (c) 1994, ODETICS, INC.

SUMMARY:

Bridge Inspector Project
IDS Image Processor Support Functions

DESCRIPTION:

This file contains the IDS Image Processor support functions.

SEE ALSO:

idsControl.h

HISTORY:

$Log$


$ModHeadEnd$
*/

#include "idsControl.h"

/* define ANSI extension flag for vxWorks linker */
int __ANSILIB;



/* ************************* Functions ********************************* */
#ifdef __CPLUSPLUS
BYTE getByte(const BYTE *addr)
#else
BYTE getByte(addr)
pBYTE addr;
#endif
{
return(*addr);
}

#ifdef __CPLUSPLUS
WORD getWord(const WORD *addr)
#else
WORD getWord(addr)
WORD *addr;
#endif
{
return(*addr);
}
```

```
#ifdef __CPLUSPLUS
SHORT getShort(const SHORT *addr)
#else
SHORT getShort(addr)
SHORT *addr;
#endif
{
return(*addr);
}

#ifdef __CPLUSPLUS
LONG getLong(const LONG *addr)
#else
LONG getLong(addr)
LONG *addr;
#endif
{
return(*addr);
}


#ifdef __CPLUSPLUS
void setByte(pBYTE addr,  BYTE val)
#else
void setByte(addr, val)
pBYTE addr;
BYTE val;
#endif
{
*addr = val;
}

#ifdef __CPLUSPLUS
void setWord(pWORD addr, WORD val)
#else
void setWord(addr, val)
pWORD addr;
WORD val;
#endif
{
*addr = val;
}

#ifdef __CPLUSPLUS
void setShort(pSHORT addr, SHORT val)
#else
void setShort(addr, val)
pSHORT addr;
SHORT val;
#endif
{
*addr = val;
}
```

APPENDIX B

USERS MANUAL AND ON-LINE HELP TEXT

Users Manual
Bridge Inspector
Image Enhancement Ground-Based Station
(IEGBS)

Odetics
Advanced Systems Development
Communications Division
1585 S. Manchester Avenue
Anaheim, California 92802

Prepared for UC Davis AHMCT
Contract# 65Q168MOU#92-11

Lloyd Tripp

Contents

Chapter One Introduction

The Bridge Inspector Image Enhancement Ground-Based Station (IEGBS) is a system of special hardware and software developed especially to view, capture, and enhance images from a video camera performing a bridge inspection. By properly operating the IEGBS, the user will be able to:

- View live video from the camera on the monitor
- Freeze/Unfreeze the video
- Process the live video using the IDS Image Processor
- Take a digital "snap shot" of the video image
- View/Process digital snap shots
- Record the video using a Hi8 video tape recorder
- View/Process video previously recorded on video tape
- Digitally zoom in on part of the image
- Remotely control camera functions

Familiarity with PC's and Windows 3.1

The IEGBS is based on a 486/66 computer running the Microsoft Windows 3.1 user interface. As such, some of the terminology used in this guide assumes some familiarity personal computers (PC's) and the Windows 3.1 user interface. If you are not familiar with Windows 3.1, an on-line Help function is available on the IEGBS (or any PC that has Windows 3.1 installed on it) that can direct you to the Windows on-line Help. From there, you can learn about the basics of the Windows 3.1 user interface and the terms used in this document. There are also a large number of resource books introducing the PC and using Windows 3.1.

This Users Manual will explain how to set up the IEGBS and how to operate it while performing a bridge inspection.

Once you become familiar with using the IEGBS, there is an on-line Help function that you can review on the computer monitor if a printed copy of this Users Manual is not available.

Chapter Two Getting Started

In order to use the IEGBS effectively, the user must become familiar with both the hardware and the software that comprise the total system. If you are already familiar with PC's and Windows 3.1, then many of the concepts introduced in this chapter will be familiar to you. However, because of the specialized nature of the hardware and software, even experienced PC users should look through this chapter.

Image Processing

The IEGBS is, for the most part, a digital image processing system. However, no familiarity with image processing concepts is required. Any image processing concepts will be explained so you will have a better understanding of the operations being performed by the IEGBS and therefore a better understanding of the processing results you are seeing.

Image processing is a set of operations that use images as the operands or data. While there are optical image processors that use lenses and filters to perform operations, image processing in the context of this document will be digital image processing performed using a computer or other kind of digital hardware. Just like word processors manipulate and format the elements of a text document, i.e. letters, words, paragraphs, etc., an image processor manipulates and operates on the elements of a digital image. The first step, therefore, is to get an image that can be seen with a video camera into the digital elements that can be operated upon. Before we can understand how this is performed, you need to know a little about the video signal from a video camera.

The video signal from a video camera or a video tape recorder is grouped into "fields" and "frames." A frame is a full resolution image that you normally see on a video monitor or TV. A field is a subdivision of a frame - it takes two fields to make a frame. Since we want to use the best possible video image, we want to use a full frame image from the video camera. The video camera produces 30 frames of video per second. The IEGBS uses a special board called a "frame grabber" to transform an image that can be seen with a video camera into a digital image that can be processed by the computer. In essence, the frame grabber takes one frame of the video signal and digitizes it so that it looks like a big table (640 columns x 480 rows) of numbers. Each one of these numbers is a picture element or "pixel." Once the video image is in the form of a table of pixels, then the full power of the computer can be used to operate/process these numbers to either

save them to the hard disk as a digital snap shot, or process the numbers to make a new image. The ability to make a new image that enhances those parts of the image of interest - such as cracks - is the purpose of the IEGBS.

In order to make the new, enhanced image visible to the user, we must transform the big table of pixels back into a video image that the user can see. The frame grabber in the IEGBS also performs this reverse function so that a digital image can be shown to the user. The frame grabber can also display live video from the camera or VCR in a window on the monitor. This ability to display live video on the same computer monitor that we display processed images means that a separate video monitor just for live video is not needed.

The time it takes to perform some image processing operations is often greatly reduced if specialized image processing hardware is used. The reason for this is that the digital images require quite a lot memory to hold them, and the image processing operations often have to be repeated for every pixel in the image. For example, in a full color 640 x 480 image, there are 307,200 pixels and each of the pixel values can range from 0 to 16,777,215. This means a full color digital image is about 1 megabyte in size. Even for relatively simple image processing functions, the number of multiplications and additions can easily exceed 5 million. While the image processing operations performed on a general purpose computer are very flexible since they are performed in software, a typical PC would take several seconds to perform this number of operations. Specialized image processing hardware can be configured with large amounts of high speed memory to hold the digital images and specialized, high-speed computational hardware to perform a small set of operations very quickly. The result is real-time (30 frames per second) image processing. The IEGBS is configured with specialized IDS image processing hardware to perform contrast and edge enhancement in real-time. The resulting image is a black & white (gray scale) image that can reveal some image features that may be hard to see under some conditions of camera lighting.

Host Computer

While the IDS Image Processor is very fast, it is also very specialized in that it only performs one type of operation. A general purpose Host Computer is needed to perform all the other functions of the IEGBS.

The Host Computer is comprised of a standard PC motherboard plus the frame grabber board as well as three different disk memories for storing the captured images. The primary disk is a large fixed disk memory that holds the IEGBS software as well as digital images the user has collected. This is the fastest and largest disk memory, but since this disk is fixed, users have to copy images that are stored on this disk to one of the removable disks. While not quite as fast as the fixed disk, each removable disk cartridge can hold up to 450 compressed digital images (more about compression later) making it ideal for archiving images from a bridge inspection. The smallest, albeit most ubiquitous, removable disk memory is the 3.5" floppy disk. Each floppy disk can only hold about four compressed digital images or just one uncompressed image, but because 3.5" floppy disks can be found on most PC's, the floppy is the safest way to transport images from one computer to another.

The Host Computer also has a read-only CD-ROM drive. CD-ROM's are capable of storing large numbers of digital images (approximately 1500 compressed images), but the images have to be put on the disk using a service bureau, i.e. they can not be put on by the host computer. For large numbers of archival images that must be stored for long periods of time, CD-ROM's are probably the best choice.

Frame grabber board used in the Host Computer is normally configured to perform image compression. Image compression is an operation that is used to reduce the amount of memory required to store a digital snap shot from about 1 megabytes to usually less than 300 kilobytes. While image compression will reduce the memory required to hold a digital image, it does take more time to store the digital snapshots and the compressed images are not portable to other types of applications such as a report writer. One helpful feature is that compressed images can always be de-compressed any time the user wants to move the images to another computer or application.

User Interface

Beyond the hardware, the most important function that the Host Computer performs is to integrate (bring together) all the different parts of the IEGBS so that they can be controlled from the User Interface.

The User Interface is divided into two main areas: the control button bar and the image display window. The control button bar holds a

group of Windows buttons that are used to perform the functions of IEGBS. The keyboard is only used to enter textual information, not activate functions. Using buttons with icons and names instead of the more cryptic function buttons on the keyboard should make it easier to learn and use. The image display window, as the name implies, shows the live video from the camera as well as still digital images that were saves on disk.

Detail about the user interface is presented in Chapter 4.

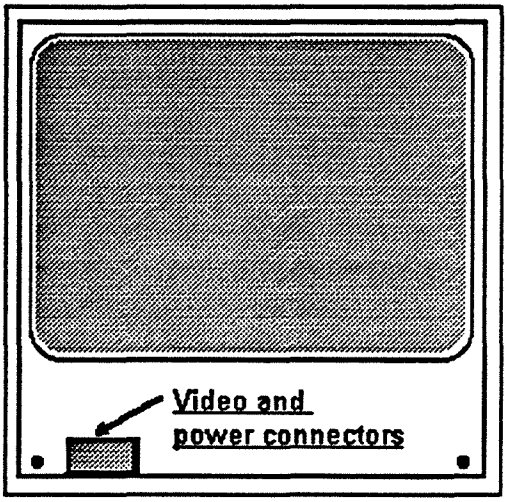
Chapter Three Starting up the IEGBS

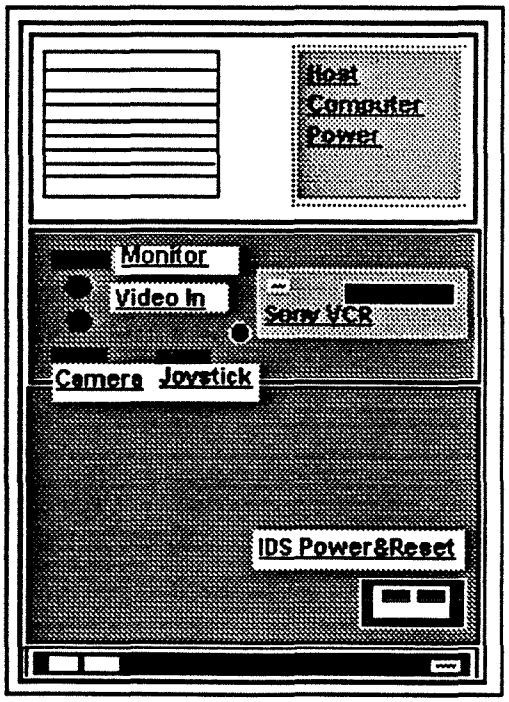
The IEGBS should be started up using the following sequence:

1.Make sure the power strip at the bottom of the large enclosure is in the OFF position.

2.Make sure the Host Computer power switch is off. The power switch is behind the front door on the Host Computer.

3.Make sure the IDS Image Processor power is off - the DC toggle switch is down.





4.Plug in the main power cord coming out of the back of the large enclosure into a 120 VAC outlet.

5.Plug the monitor power cord into one of the sockets on the power strip.

6.Connect the monitor video to the monitor connector on the front panel.

7.Connect the video from the camera to the Video In connector.

8.Connect the Camera control to the Camera connector.

9.Connect the Joystick to the Joystick connector.

10.Turn on the main power on the power strip.

11. Turn on the IDS processor.

12. Turn on the Sony VCR. Power light should be green.

13.Turn on the Host Computer power.

The diagrams to the left indicates where these connections are located.

The IEGBS software will automatically start after you turn on the host computer power. You can then enter commands to the User Interface.

Chapter Four User Interface

All interaction between the user and the IEGBS takes place using the User Interface. The primary way the user commands the IEGBS is by using the mouse to click on the buttons in the button bar at the top of the User Interface screen. This chapter will briefly explain the function of each button. The best way for the user to appreciate the function of each button is to try them out.

By clicking on the video select button, the user will be presented with a control panel from which the user will be able to select the source of the video being shown in the Image Display Window. The user can select either the camera video or the enhanced video from the IDS Image Processor by clicking on the circle next to the name. The user also has the option of "freezing" the video by clicking on the box next to the word Freeze. This action will stop the image on the screen. Clicking on the box again (unchecking the box) will restore the live video.

You can also display a saved image from the disk by clicking on the Display Saved Image button. This will bring up a file selection window with the names of the images that are available for viewing. The file selection window will permit you to switch to another disk if the image you want to view is on a disk other than the default disk. (You set the default disk using the User Options command.) When the saved image is displayed, the video selection will be in the freeze mode. After you have viewed the saved image, you can go back to viewing live video by clicking on the Video Select button again and unchecking the Freeze box.

When you have finished, click on the Done button at the bottom of the Video Selection window.

Note: If you are playing a tape, the video select will not switch as expected since the video is coming from the tape and not from the camera or the IDS Image Processor. Also, while playing a tape the Freeze will only stop the video from showing on the screen, it will not put the VCR in pause.

When the user clicks on the Snap Shot button, a digital picture will immediately be taken. This same function is performed using the top button on the joy stick. After the image is captured, the user will be shown a Save File window that will permit the user to name the digital image. From the Save File window, the user can also select another disk, such as the floppy, instead of the default disk.
If the user has selected the auto-naming feature in the User Options window, then snap shot image will automatically be named and saved.

The IDS Processor control button will bring up a window that will permit you to perform the IDS image enhancement on a single video frame instead of the live video coming from the camera. When the user selects this button, a window with three control buttons appears.

The top button labeled "Process Live Image..." directs the IEGBS to capture a single frame from the current video source and process it through the IDS Image Processor. Since the current video source can be either the live video from the camera, or recorded video from the VCR, the user can process previously recorded images. While the image is being processed, a completion indicator bar appears beneath the button. All other IEGBS operations are stopped while the image is being processed. The resulting image is displayed in the Image Display Window with the Freeze control box activated. After you have reviewed and/or saved the processed image, you have to go to the Video Select control panel, by clicking on the Video Select button, and unchecking the Freeze selection to go back to displaying the current video selection.

By clicking on the middle button labeled "Process Saved Image..." the user will be presented with a file selection window just like the one used in the Video Select control panel when you want to display a saved image. After you select an image file, it will be processed by the IDS Image Processor and the result displayed in the Image Display Window. Just like when we processed a single frame from the current video source, there will be a completion indicator bar below the button. Wait until processing is complete before starting a new operation. After reviewing the processed image, unfreeze the image as described above.

The bottom button is labeled "Advanced Features..." By clicking on this button the user will be presented a control window that will permit finer control of the IDS processing function and also reset the IDS processor in case it loses synchronization with the video signal. At the top of the IDS Advanced Features control window is a "slider" that selects the degree of edge and contrast enhancement. By first selecting the IDS Image Processor as the display source using the Video Selection operations described above, and then clicking on the small arrows at each end of the slider, you will be able to see the result of different degrees of edge and contrast enhancement.

The Reset button on this control panel will get the IDS Image Processor back in synchronization with the video signal. When the IDS Image Processor loses synchronization, the video from the IDS Image Processor looks like it is "tearing" across the middle. Clicking on the Reset button will restore the proper IDS result image.

By clicking on the button labeled "8mm VCR", the user will bring up a control panel for controlling the video cassette recorder (VCR). The Sony CVD-1000 that is used in the IEGBS is a computer controllable Hi8 VCR. This means that all the functions that you are familiar with on your home VCR can be controlled using computer generated messages. Another thing that is different about the CVD-1000 compared to most home VCR's is that it uses a Hi8 tape cassette instead of the more common VHS. Hi8 is more commonly found in Handicams where size is a premium. However, despite its smaller size, it has the best recorded video quality of consumer video tape formats. What this means for the IEGBS is that the inspection videos recorded using the CVD-1000 will be high quality videos that will not lose potentially important details.

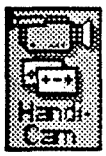
The VCR control panel on the screen functions much like you would expect. Clicking on the buttons in the control panel sends a message to the CVD-1000 to activate the desired function. Some combinations of buttons are also permitted for special functions. They are:

- Play & Fast Fwd: forward visual search.
- Play & Rew: Reverse visual search.
- Play & Pause: Pauses the video.

You can use the frame forward and frame backward buttons below the Pause button to search one frame at a time. The frame forward and frame backward buttons only work when the Pause button is activated (it is highlighted yellow). To resume play on any of the above button combinations, just click on the Play button.

The button next to the time display can change the time display from hours:minutes:seconds:frame to month/day/year. This information is automatically recorded on the tape when the user records an inspection video, and provides an important time stamp needed for archival purposes.

By using the time and frame number stamp on the video, the user will quickly be able to find parts of the video of interest. While the user is in Play or Pause, the top button of the joystick can be pressed to capture the current image being displayed.



By clicking on the button labeled "HandiCam", the user will bring up a panel for controlling the HandiCam that is serving as the video camera for the IEGBS. The panel is primarily used to adjust the Zoom of the lens. The buttons act like on/off switches in that once you click to start the zoom function, you must click again to stop it. When the camera is set in the manual focus mode by setting a switch on the camera, the user can also control the focus from the HandiCam control panel. A data display check box on the control panel will instruct the camera to display graphic symbols on the video that shows, among other things, the position of the zoom lens. Checking and unchecking this box toggles the data display.

By clicking on the button labeled "Image Proc", the user will bring up a panel for performing some image processing functions on images that have been previously saved to disk. Currently, the image processing function is limited to "digitally zooming" in on a part of the image. A 4x zoom from an original image, i.e. one zooming step, is about all that is useful. Unlike an optical zoom using a lens, digital zoom causes the image quality to be reduced as the image is zoomed.

The user can select default values and actions by clicking on the "User Options" button. The control panel lets the user choose between automatic naming of captured images or manual naming. If the user selects automatic naming, the user also specifies the root name of the image. The IEGBS will append numbers to the root name to form the full image name. By using automatic naming, the user can keep taking images with out stopping to name each image.

The user also has the option about which file type the image should be saved in. The compressed .DVA files are smaller so you can get more of them on a disk, however, the format is proprietary so very few software packages can read it. The .DIB format can be read by many programs that run on Windows, but they are about three times bigger than the .DVA files.

There is online help available to the user when the "HELP" button is clicked. The online help provides a brief summary of the IEGBS and its operation. It is not designed as a training manual, rather it serves as a reminder to someone who already is familiar with the operation of the IEGBS. Since the IEGBS Help is hooked into the Windows Help function, the user can also access online help about Windows by using this button.

The final button is the "Exit" button. When the user clicks on this button, the IEGBS user interface exits and you are returned to the Windows interface.

Welcome to the Bridge Inspection IEGBS

This help file will guide you in the operation of the Bridge Inspection Image Enhancement Ground-Based System (IEGBS).

To get from one topic to another, you just click the left mouse button when the arrow is over any words that are green.

Contents

If you want to know more about Windows 3.1, click on Windows.

Getting Started

The bridge inspection Image Enhancement Ground-Based System (IEGBS) is a computer system that is used to capture and enhance images taken with a video camera during bridge inspections.

In order to use the system effectively, the user must become familiar with both the hardware and the software that comprise the total system.

IEGBS Hardware - A look at the hardware.

IEGBS Software - A look at the programs and user interface.

Starting the IEGBS - The start up sequence.

A quick look at the hardware

Monitor - 17" color monitor in rugged enclosure. Power and video connections are accessible without opening the front panel. To adjust the brightness and contrast, swing up the front panel to get access to the monitor controls.

Host Computer - The host computer has a 486DX2/66 processor. It is configured with the Floppy as drive A:, Fixed 520 megabyte hard disk as drive C:, a removable 150 megabyte disk as drive D:, and a CD-ROM drive as drive F:. Since the drives are mounted sideways, please note that the floppy disks, removable disks, and CD-ROMs should be inserted with their labels facing RIGHT. The host computer performs all interaction with operator through the user interface.

Sony Hi8 VCR - The Hi8 VCR is similar to a home VCR except that it uses smaller, higher quality Hi8 tapes. It has the same Play, Stop, Record buttons that you are familiar with except that this VCR is designed to be operated by a computer. The buttons under the flip-down panel will not work when the computer link is active - when the VISCA light above the MIC jack is on.

IDS Image Processor - The IDS processor at the bottom of the enclosure is a high speed computer that performs a special type of image processing that enhances edges and image contrast.

For more information on the Host Computer and the IDS Image Processor, see Hardware Details.

Hardware Details

The hardware that comprises the IEGBS has been custom configured to perform the functions required for image enhancement. Consequently, there are a number of devices that are not commonly found in general purpose personal computers. This will highlight some of the special hardware and how it is used.

The Host Computer starts with a 486DX2/66 VLB motherboard with 16 MB RAM and adds:

- VLB SVGA graphics processor (STB-24)
- SCSI disk controller
- 520 MByte SCSI disk
- CD-ROM with SCSI interface
- IOMEGA Bernoulli 150 removable hard disk and controller board
- VideoLogic DVA-4000 video capture and display board
- Intel Ether Express ethernet interface board

The SCSI interface performs somewhat faster than the more common IDE disk controller so that large digital images are more quickly saved and displayed.

The Bernoulli 150 is a fast hard disk that holds up to 150MBytes of data - about 450 compressed digital images.

The DVA-4000 displayes the live video on the screen as well as permits the capture of high-resolution, compressed images to the hard disk.

The ethernet interface is used by the Host Computer to communicate with the IDS processor for fast transfer of digital images.

The IDS Image Processor.

The IDS Image Processor is a VME-bus computer that uses an Ironics IV-3234 single-board 68030 computer and dedicated image processing hardware. While the IV-3234 provides overall control and communication interface to the Host Computer, the image processing is performed using a combination of commercially available boards from Data Cube and nine Odetics proprietary IDS processing boards.

A quick look at the software

The software used for the IEGBS is hosted on the two computers used to perform the IEGBS functions: the Host Computer and the IDS Image Processor.

The Host Computer Software.
The Host Computer Software, as the name implies, executes most of the software used to perform the IEGBS operations. The software is a Windows 3.1 application which means it has a look and feel similar to other Windows software the user may be used to using. Through the user interface, the user directs the computer to perform the desired operation. The software can be broken into the following modules:

- Image display and capture
- VCR control
- Camcorder control
- IDS Image Processor interface.

Image display and capture.
The software directs the image display and capture board to perform the desired operation. The user has the option of setting the image capture format using the User Options control panel.

VCR Control.
The software directs the VCR to perform the operations directed by the VCR control panel.

Camcorder Control.
The software sends zoom and focus commands to the camcorder via the Remote jack on the camcorder.

IDS control.
The IDS commands are sent from the Host Computer to the IDS Image Processor via the ethernet link. Digitized images saved on disk are sent to the IDS and the result images sent back to the Host Computer using the ethernet.

Starting the IEGBS

This is the start up sequence that should be followed.

- Make sure the power strip at the bottom of the large enclosure is in the OFF position - bottom half depressed.
- Make sure the host computer power switch is OFF - open the front door and check that the power switch is depressed above the 0.
- Make sure the IDS processor power is off - the DC toggle switch is in the down position.
- Plug in main power cord coming out the back of the large enclosure into a 120 VAC outlet.
- Plug the monitor power cord into the one of the sockets on the power strip at the bottom of the large enclosure.
- Connect the monitor video to the video connector on the connector panel.
- Turn on the main power on the power strip.
- Turn on the IDS processor.
- Turn on the Sony Hi8 VCR. (Power light should be green.)
- Turn on the host computer.

The IEGBS software will automatically start and you will enter commands using the user interface.

User Interface

All interaction between the user and the IEGBS takes place using the User Interface. The user interface is divided into two main areas: the control button bar at the top of the window, and the image display window.

The Image Display Window.
The image display window, as the name implies, shows the live video image from the camera as well as still images that are saved on disk.

The Control Button Bar.
The buttons across the top of the user interface display command the IEGBS to perform the desired operation. Button operations are summarizedas follows:

- Video select - Select between the video camera or the IDS processed image. The user can optionally freeze or unfreeze the live image. If you click on the Display Saved Image button, you are presented with a list of images which can be displayed.
- Snap shot - Immediately takes a digital picture. If the user has selected the auto-naming option, it will name the image, otherwise the user must name the image. The top button on the joy stick is the same as the Snap shot button on the screen.
- IDS Proc - Permits the user to process saved images using the IDS Image Processor. Advanced features permit finer control of the IDS Image Processor as well as a reset button in case the IDS processor loses synch with the video.
- 8mm VCR - Brings up a VCR control panel on the screen.
- HandiCam - Controls zoom and focus of Handicam. Note: Focus can only be controlled when the focus is set to manual on the handicam.
- Image Proc - User can select to zoom on a portion of the image.
- User Options - Set where you want to save the digitized images and the format.
- Help - Brings up this file.
- Exit - Exits the IEGBS software and takes you back to Windows.

Pop-up Messages

Pop-up message boxes are used to inform the user about the condition of the IEGBS hardware. The user must acknowledge these message boxes by using the mouse to click on the OK button.

To find out more information about the messages listed below, just click on the message.

No Netlink. Reset IDS box and try again.

VISCA Timeout.

No Netlink. Reset IDS box and try again.

Indication.
No Communication between the Host Computer and the IDS Image Processor.

Operation Details.
The Host Computer and the IDS Image Processor communicate using an ethernet link. When the Bridge Inspector IEGBS is started, the Host Computer and the control computer in the IDS Image Processor try to establish the ethernet communication link. If the IDS control computer has not finished its start up initialization before the Host Computer tries to contact it, the communication link will fail.

What to do.
This condition is corrected by performing the following steps:
1. Make sure the power switch on the IDS Image Processor is in the On position.

2. Reset the IDS Image Processor by moving the Reset toggle switch on the front of the IDS front panel UP, then release.

3. Exit the Bridge Inspector IEGBS by clicking on the Exit button on the control bar of the user interface.

4. Wait at least 1 minute so that the IDS Image processor is properly initialized.

5. Restart the Bridge Inspector Host Computer by opening the hinged panel and pressing the Reset button.

NOTE:
If you are not planning to use the IDS Image Processor, you can leave its power Off and ignore this message by just clicking on the Ok button in the message box. The rest of the functions of the Bridge Inspector IEGBS will perform as usual.

VISCA Timeout.

Indication.
Bad communication link to Sony VCR and/or Handicam Control CI-1000.

Operation Details.
Command messages and replies are send between the Host Computer, the Sony VCR and the Handicam Control interface called the CI-1000. If a command message is sent and there is no reply within a specified time limit, this message will appear to alert the user.

What to do.
If the message appears occasionally especially while using the VCR control panel, it probably means connamds are being issued too quickly. Just slow down the rate at which you push the control buttons. Close the message box by clicking on the **Ok** button.

If the messages appear rapidly while using the VCR, or you get a message almost every time you push a button while using the HandiCam control, then message path is broken. Check the following:

- The VCR should be On. (Power light **green**).
- The HandiCam controller CI-1000 must be On.
- Make sure the cable connecting the front panel connector labeled **Camera Control** to the CI-1000 controller is good and connections are tight.

APPENDIX C

SOURCE CODE FOR USER INTERFACE AND CONTROL SOFTWARE

```
// about.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "about.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAbout dialog

CAbout::CAbout(CWnd* pParent /*=NULL*/)
        : CDialog(CAbout::IDD, pParent)
{
        //{{AFX_DATA_INIT(CAbout)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}

void CAbout::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAbout)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAbout, CDialog)
        //{{AFX_MSG_MAP(CAbout)
                // NOTE: the ClassWizard will add message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAbout message handlers
```

```
// about.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CAbout dialog

class CAbout : public CDialog
{
// Construction
public:
        CAbout(CWnd* pParent = NULL);        // standard constructor

// Dialog Data
        //{{AFX_DATA(CAbout)
        enum { IDD = IDD_ABOUTBOX };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CAbout)
                // NOTE: the ClassWizard will add member functions here
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// bridgdoc.cpp : implementation of the CBridgeDoc class
//

#include "stdafx.h"
#include "bridge.h"

#include "bridgdoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBridgeDoc

IMPLEMENT_DYNCREATE(CBridgeDoc, CDocument)

BEGIN_MESSAGE_MAP(CBridgeDoc, CDocument)
        //{{AFX_MSG_MAP(CBridgeDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBridgeDoc construction/destruction

CBridgeDoc::CBridgeDoc()
{
        // TODO: add one-time construction code here
}

CBridgeDoc::~CBridgeDoc()
{
}

BOOL CBridgeDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CBridgeDoc serialization

void CBridgeDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
```

```
	{
		// TODO: add loading code here
	}
}

/////////////////////////////////////////////////////////////////////////////
// CBridgeDoc diagnostics

#ifdef _DEBUG
void CBridgeDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CBridgeDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CBridgeDoc commands
```

```
// bridgdoc.h : interface of the CBridgeDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CBridgeDoc : public CDocument
{
protected: // create from serialization only
        CBridgeDoc();
        DECLARE_DYNCREATE(CBridgeDoc)

// Attributes
public:

// Operations
public:

// Implementation
public:
        virtual ~CBridgeDoc();
        virtual void Serialize(CArchive& ar);      // overridden for document i/o
#ifdef _DEBUG
        virtual  void AssertValid() const;
        virtual  void Dump(CDumpContext& dc) const;
#endif
protected:
        virtual  BOOL  OnNewDocument();

// Generated message map functions
protected:
        //{{AFX_MSG(CBridgeDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //   DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// bridge.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "bridge.h"

#include "mainfrm.h"
#include "bridgdoc.h"
#include "bridgvw.h"

// Define KNIFE_DATA only in this module before the include knifevbx.h
#define         KNIFE_DATA
#include "knifevbx.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBridgeApp

BEGIN_MESSAGE_MAP(CBridgeApp, CWinApp)
        //{{AFX_MSG_MAP(CBridgeApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
        // Global help commands
        ON_COMMAND(ID_HELP_INDEX, CWinApp::OnHelpIndex)
        ON_COMMAND(ID_HELP_USING, CWinApp::OnHelpUsing)
        ON_COMMAND(ID_HELP, CWinApp::OnHelp)
        ON_COMMAND(ID_CONTEXT_HELP, CWinApp::OnContextHelp)
        ON_COMMAND(ID_DEFAULT_HELP, CWinApp::OnHelpIndex)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBridgeApp construction

CBridgeApp::CBridgeApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CBridgeApp object

CBridgeApp NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
```

```
// CBridgeApp initialization

BOOL CBridgeApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

	SetDialogBkColor();	// set dialog background color to gray
	LoadStdProfileSettings();  // Load standard INI file options (including MRU)
	EnableVBX();		// Initialize VBX support
		if(!imkBegin())			// Initialize ImageKnife/VBX
			AfxMessageBox((LPCSTR)"Could not init image knife.",
				MB_OK | MB_ICONEXCLAMATION, 0);
	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	AddDocTemplate(new CSingleDocTemplate(IDR_MAINFRAME,
			RUNTIME_CLASS(CBridgeDoc),
			RUNTIME_CLASS(CMainFrame),	// main SDI frame window
			RUNTIME_CLASS(CBridgeView)));


	// create a new (empty) document
	OnFileNew();

	if (m_lpCmdLine[0] != '\0')
	{
		// TODO: add command line processing here
	}

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

// Implementation
protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CBridgeApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// VB-Event registration
// (calls to AfxRegisterVBEvent will be placed here by ClassWizard)

//{{AFX_VBX_REGISTER_MAP()
        UINT NEAR VBN_CLICK = AfxRegisterVBEvent("CLICK");
        UINT NEAR VBN_SPINUP = AfxRegisterVBEvent("SPINUP");
        UINT NEAR VBN_SPINDOWN = AfxRegisterVBEvent("SPINDOWN");
        UINT NEAR VBN_KEYDOWN = AfxRegisterVBEvent("KEYDOWN");
        UINT NEAR VBN_KEYUP = AfxRegisterVBEvent("KEYUP");
        UINT NEAR VBN_KEYPRESS = AfxRegisterVBEvent("KEYPRESS");
//}}AFX_VBX_REGISTER_MAP

/////////////////////////////////////////////////////////////////////////////
// CBridgeApp commands
```

```
// bridge.h : main header file for the BRIDGE application
//

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"               // main symbols

// Define the joystick timer id
#define CHECK_JOY   105

/////////////////////////////////////////////////////////////////////////////
// CBridgeApp:
// See bridge.cpp for the implementation of this class
//

class CBridgeApp : public CWinApp
{
public:
        CBridgeApp();

// Overrides
        virtual BOOL InitInstance();

// Implementation

        //{{AFX_MSG(CBridgeApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
// VB-Event extern declarations

//{{AFX_VBX_REGISTER()
        extern UINT NEAR VBN_CLICK;
        extern UINT NEAR VBN_SPINUP;
        extern UINT NEAR VBN_SPINDOWN;
        extern UINT NEAR VBN_KEYDOWN;
        extern UINT NEAR VBN_KEYUP;
        extern UINT NEAR VBN_KEYPRESS;
//}}AFX_VBX_REGISTER

/////////////////////////////////////////////////////////////////////////////
```

```
// bridgvw.cpp : implementation of the CBridgeView class
//

#include "stdafx.h"
#include "bridge.h"
#include "mainfrm.h"

#include "bridgdoc.h"
#include "viscadev.h"

#include "vscadlg.h"

#include "camdlg.h"

#include "vidsdlg.h"

#include "commdlg.h"

#include "usroptdg.h"

#include "idsdlg.h"

#include "improcdl.h"

#include "mic2a.h"
#include "about.h"

#include "bridgvw.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <errno.h>

// Define the dialogs that we want to post messages to
CCamDlg                 CameraDlg;


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

static UINT NEAR WM_VISCAINQ = RegisterWindowMessage((LPCSTR)"VISCA_INQ");

static UINT NEAR WM_VISCAERR = RegisterWindowMessage((LPCSTR)"VISCA_ERR");

#define SEND_TIMEOUT_ID         101
#define SEND_TIMEOUT    10

#define SERVER_NUM              1100
#define IM_EX_PORT              1101

#define HOSTNAME                "198.58.69.33" /*ids*/    /* 198.58.69.5 robin */
```

```
#define         MAKEWORD(lo, hi) ((WORD) (((BYTE) (lo)) | (((UINT) ((BYTE)(hi))) << 8)))

/////////////////////////////////////////////////////////////////////////////
// CBridgeView

IMPLEMENT_DYNCREATE(CBridgeView, CView)

BEGIN_MESSAGE_MAP(CBridgeView, CView)
        //{{AFX_MSG_MAP(CBridgeView)
        ON_COMMAND(ID_VISCAActive, OnVISCAActive)
        ON_COMMAND(ID_CAMERAActive, OnCAMERAActive)
        ON_COMMAND(ID_VideoSelect, OnVideoSelect)
        ON_WM_TIMER()
        ON_COMMAND(ID_SNAP_SHOT, OnSnapShot)
        ON_COMMAND(ID_IMAGE_PROC, OnImageProc)
        ON_COMMAND(ID_USER_HELP, OnUserHelp)
        ON_COMMAND(ID_USER_LEVEL, OnUserLevel)
        ON_COMMAND(ID_USER_OPTIONS, OnUserOptions)
        ON_MESSAGE(WM_COMMNOTIFY, OnCommNotify)
        ON_COMMAND(ID_IDS, OnIds)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBridgeView construction/destruction

CBridgeView::CBridgeView()
{
        m_goodPal = CreateVideoPalette();
        initMic();

        m_saveFormat = FALSE;  // Set default format to dva.
        m_saveAutoNameButton = FALSE;
        m_saveBaseName = "";
        m_saveSequenceNum = 0;
        m_saveDestination = HardDisk;

        if(clientSock = openPort(SERVER_NUM)){
                pause(700); // Wait for n msec to allow the IDS listen to occur
                imExSock = openPort(IM_EX_PORT);
                startUpIDS();
        }else // Could not open link
                AfxMessageBox("No Netlink. Reset IDS box and try again.", MB_OK |
MB_ICONEXCLAMATION, 0);
}

CBridgeView::~CBridgeView()
{
        CByteArray*     pCBA;

        // If there are any messages in the messageQ, delete them.
        while(messageQ.GetCount() > 0){
```

```
                pCBA = (CByteArray*)messageQ.RemoveHead();  // Gets the head and
removes it
                // Remove all elements from pCBA
                pCBA->RemoveAll();
                delete []pCBA;
        }

        messageQ.RemoveAll(); // Just to make sure;

        if(clientSock) closePort(clientSock);
        if(imExSock) closePort(imExSock);
        cleanupWSA();

        closeMic();
}

/////////////////////////////////////////////////////////////////////////////
// CBridgeView drawing

void CBridgeView::OnDraw(CDC* pDC)
{
        CBridgeDoc* pDoc = GetDocument();

        MoveVideoWindow();
        TransparentPaint(pDC);
}

/////////////////////////////////////////////////////////////////////////////
// CBridgeView printing

BOOL CBridgeView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CBridgeView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void CBridgeView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}



/////////////////////////////////////////////////////////////////////////////
// CBridgeView diagnostics

#ifdef _DEBUG
void CBridgeView::AssertValid() const
{
        CView::AssertValid();
```

```
}

void CBridgeView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}

CBridgeDoc* CBridgeView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CBridgeDoc)));
        return (CBridgeDoc*) m_pDocument;
}

#endif //_DEBUG

////////////////////////////////////////////////////////////////////
// Dump DCB
// Dump the serial port device control block
#ifdef _DEBUG
void CBridgeView::DumpDCB (DCB* pDCB)
{
        char    dumpBuf[500];

        wsprintf((LPSTR) dumpBuf, (LPSTR) "DCB Struct\n \
        BYTE Id %lx \n \
        UINT BaudRate %lx \n \
        UINT ByteSize %lx \n \
        UINT Parity %lx \n \
        BYTE StopBits %lx \n \
        UINT RlsTimeout %lx \n \
        UINT CtsTimeout %lx \n \
        UINT DsrTimeout %lx \n \
        UINT fBinary :1 %lx \n \
        UINT fRtsDisable :1 %lx \n \
        char EvtChar %lx \n",
        (LONG)pDCB->Id,
        (LONG)pDCB->BaudRate,
        (LONG)pDCB->ByteSize,
        (LONG)pDCB->Parity,
        (LONG)pDCB->StopBits,
        (LONG)pDCB->RlsTimeout,
        (LONG)pDCB->CtsTimeout,
        (LONG)pDCB->DsrTimeout,
        (LONG)pDCB->fBinary,
        (LONG)pDCB->fRtsDisable,
        (LONG)pDCB->EvtChar);

        afxDump << dumpBuf;

}
#endif

////////////////////////////////////////////////////////////////////
/// CBridgeView Palette functions
// Must create and realize a logical palette
BOOL CBridgeView::CreateVideoPalette()
```

```
{
        // CPalette* pPal;
        LPLOGPALETTE lpPal;     // pointer to a logical palette
        HANDLE hLogPal;         // handle to a logical palette
        int i;                  // loop index
        WORD wNumColors = 64;   // number of colors in color table
        BOOL bResult = FALSE;


  if (wNumColors != 0)
  {
                /* allocate memory block for logical palette */
                hLogPal = ::GlobalAlloc(GHND, sizeof(LOGPALETTE)
                                                                        +
sizeof(PALETTEENTRY)
                                                                        * wNumColors);

                /* if not enough memory, clean up and return NULL */
                if (hLogPal == 0)
                {

                        return FALSE;
                }

                lpPal = (LPLOGPALETTE) ::GlobalLock((HGLOBAL) hLogPal);

                /* set version and number of palette entries */
                lpPal->palVersion = PALVERSION;
                lpPal->palNumEntries = (WORD)wNumColors;

                for (i = 0; i < (int)wNumColors; i++)
                {
                        lpPal->palPalEntry[i].peRed = (BYTE)i;
                        lpPal->palPalEntry[i].peGreen = (BYTE)0;
                        lpPal->palPalEntry[i].peBlue = (BYTE)0;
                        lpPal->palPalEntry[i].peFlags = (BYTE)PC_EXPLICIT;
                }

                /* create the palette and get handle to it */
                bResult = m_Pal.CreatePalette(lpPal);
                ::GlobalUnlock((HGLOBAL) hLogPal);
                ::GlobalFree((HGLOBAL) hLogPal);
        }

        return bResult;
}


//////////////////////////////////////////////////////////////////////////
// CBridgeView Painting functions
// In order to display a live video window on the screen, the MIC II v2.2
// software must select a color (or group of colors) to be transparent
// to the video.
// In this case, we would like the entire client area to be painted with
// the transparent color.
```

```
void CBridgeView::TransparentPaint(CDC* pDC)
// Called from the OnDraw() function
{
        CBrush          hB;
        CRect           ClientRect;
        UINT            nR;

        if(m_goodPal){
                if(pDC->SelectPalette(&m_Pal, 0)){
                        nR = pDC->RealizePalette();
                        hB.CreateSolidBrush(PALETTEINDEX(TransColorIndex));

                        pDC->SelectObject(&hB);

                        GetClientRect(&ClientRect);

                        pDC->FillRect(&ClientRect, &hB);
                }
        }

}



////////////////////////////////////////////////////////////////////////////
// CBridgeView Video display functions
// These functions call the MIC functions to control the video display

/*****************************************************************************
;+
Function Name  :  AssignDVA
Inputs         :
Outputs        :
Side Effects   :
Description    :        Assign the DVA to our application, if any of the assigns
                                fail deassign any already assigned.
                                The error returned from MIC should really be interpreted
                                to determine the true cause of the failed assignment.
;-
*****************************************************************************/
BOOL CBridgeView::AssignDVA()
{
        WORD wMICErr;
        BOOL bRet;
        char szMICRspBuf[256];


        bRet = FALSE;
        wMICErr = MICWrRdStr(micID, (LPSTR) "FStore $VWin Assign DVA", szMICRspBuf,
256);
        if (wMICErr == MIC2_OK)
        {
                wMICErr = MICWrRdStr(micID, (LPSTR) "Graphics $VWin Assign DVA",
szMICRspBuf, 256);
                if (wMICErr == MIC2_OK)
                {
```

```
                        wMICErr = MICWrRdStr(micID, (LPSTR) "Audio $VWin Assign DVA",
szMICRspBuf, 256);
                        wMICErr = MICWrRdStr(micID, (LPSTR) "Video $VWin Assign DVA",
szMICRspBuf, 256);
                        if (wMICErr == MIC2_OK)
                        {
                                bRet = TRUE;
                        }
                        else
                        {
                                /* assign failed */
                                MICWriteStr(micID, (LPSTR) "Graphics $VWin Deassign");
                                MICWriteStr(micID, (LPSTR) "FStore $VWin Deassign");
                        }
                }
                else
                {
                        /* assign failed */
                        MICWriteStr(micID, (LPSTR) "Graphics $VWin Deassign");
                }
        }
        else
        {
                /* assign failed */
        }
        return (bRet);
}

/*****************************************************************************
;+
Function Name  :  DeassignDVA
Inputs         :
Outputs        :
Side Effects   :
Description    :        Deassign the DVA from our application
;-
*****************************************************************************/
void CBridgeView::DeassignDVA()
{
        /* should really do some error checking here */
        MICWriteStr(micID, (LPSTR) "Audio $VWin Deassign");
        MICWriteStr(micID, (LPSTR) "Video $VWin Deassign");
        MICWriteStr(micID, (LPSTR) "Graphics $VWin Deassign");
        MICWriteStr(micID, (LPSTR) "FStore $VWin Deassign");
}


/*****************************************************************************
;+
Function Name  :  MoveVideoWindow
Inputs         :
Outputs        :
Side Effects   :        This code compensates for a bug on the DVA-4000 board
                                which causes video coruption when the image is
displayed
```

```
                                        off the bottom right of the screen. This code is only valid
                                        if inputs are enables.
Description    :        Moves the Video Window (In screen coordinates)
;-
*****************************************************************************/
void CBridgeView::MoveVideoWindow()
// Called from the OnDraw() function and OnMove()
{
        CRect           ClientRect;
        CPoint          topLeft;
        WORD            width, height, top, left;

        char            szMICCmdBuf[256];

        GetClientRect(&ClientRect);
        ClientToScreen(&ClientRect); // Convert to screen coord needed for MIC
        width = (WORD)ClientRect.Width();
        height = (WORD)ClientRect.Height();
        topLeft = ClientRect.TopLeft();
        top = (WORD)topLeft.y;
        left = (WORD)topLeft.x;

        /*
                Hold FStore so all FStore commands are sent to the DVA at
                the same time to minimize screen disturbance.
        */
        MICWriteStr(micID, (LPSTR) "FStore $VWin Hold");
        wsprintf((LPSTR) szMICCmdBuf, (LPSTR) "FStore $VWin DPos 3 %d %d", left, top);
        MICWriteStr(micID, (LPSTR) szMICCmdBuf);

        wsprintf((LPSTR) szMICCmdBuf, (LPSTR) "FStore $VWin DSize 3 %d %d",
                        width, height);
        MICWriteStr(micID, (LPSTR) szMICCmdBuf);

        /* send all the commands down to the DVA */
        MICWriteStr(micID, (LPSTR) "FStore $VWin Release");
}

////////////////////////////
// micInq()
// Send an inquiry message to the MIC
// Needed so that other classes can send MIC commands
void CBridgeView::micInq(LPSTR szMICCmdBuf, LPSTR szMICRspBuf, WORD bufSize)
{
        MICWrRdStr(micID, szMICCmdBuf, szMICRspBuf, bufSize);
}


////////////////////////////////
// micWrite()
// Send a MIC command
// Needed so that other classes can send MIC commands
WORD CBridgeView::micWrite(LPSTR szMICCmdBuf)
{
        return(MICWriteStr(micID, szMICCmdBuf));
}
```

```
/*****************************************************************************
;+
Function Name  :  DeInitVideoWindow
Inputs         :
Outputs        :
Side Effects   :
Description    :
;-
*****************************************************************************/
void CBridgeView::DeInitVideoWindow()
{
        /* Remove the transparent colour */
        MICWriteStr(micID, (LPSTR) "Graphics $VWin Transparent None");
        /* Turn video off */
        MICWriteStr(micID, (LPSTR) "Video $VWin Off");
}


//////////////////////////
void CBridgeView::initMic()
{
        WORD result;
        char    szMICCmdBuf[256];

        // Open the MIC
        if((result = MICOpen((LPDWORD)&micID)) != MIC2_OK)
                micID = 0;

        if (micID){
                result = MICWriteStr(micID, (LPSTR)"SYSTEM RESET");
                if(AssignDVA()){
                        result = MICWriteStr(micID, (LPSTR) "FStore $VWin Mode Normal");
                        result = MICWriteStr(micID, (LPSTR)"VIDEO $VWin ON");
                        result = MICWriteStr(micID, (LPSTR)"Audio $VWin OFF");

                        /* set the transparent color */
                        wsprintf((LPSTR) szMICCmdBuf, (LPSTR) "Graphics $VWin
Transparent  %d", TransColorIndex);
                        result = MICWriteStr(micID, (LPSTR) szMICCmdBuf);

                }
        }


}

//////////////////////////
// Freeze the video input
void CBridgeView::freezeVideo(BOOL freezeUnfreeze)
{// if freezeUnfreeze = TRUE then freeze video
        WORD result;
        if(freezeUnfreeze){
                result = MICWriteStr(micID, (LPSTR) "FStore $VWin INPUT 3 DISABLE");
                //result = MICWriteStr(micID, (LPSTR) "FStore $VWin Mode FREEZEMAN");
```

```
        }else{
                result = MICWriteStr(micID, (LPSTR) "FStore $VWin INPUT 3 ENABLE");
                //result = MICWriteStr(micID, (LPSTR) "FStore $VWin Mode Normal");
        }

}


//////////////////////////
void CBridgeView::closeMic()
{
        WORD result;

        if (micID)
                DeInitVideoWindow();
                DeassignDVA();
                result = MICClose((LPDWORD)&micID);

}

/////////////////////////////////////////////////////////////////////
// CBridgeView Communication functions
// Keeping the communication functions in the view will make it easier
// to assure that only one Comm port is opened and used as the
// VISCA port.
//

int CBridgeView::m_InitComm(){

        idComm = CommErr = OpenComm("COM1", 256, 256);
        if (CommErr <0) {
          AfxMessageBox("Could not open", MB_OKIMB_ICONINFORMATION,0);
        }

        if (CommErr >= 0) {
          CommErr = BuildCommDCB("COM1:9600,n,8,1", &dcb);
          if (CommErr < 0) {
            AfxMessageBox("DCB error", MB_OKIMB_ICONINFORMATION,0);
          }
        }

        dcb.fNull = 0; // Disable input null stripping
        dcb.fDtrDisable = 0; // Enable DTR on Init
        dcb.fBinary = 1; // Enable Binary mode;
        dcb.fRtsDisable = 1; // Disable RTS
        dcb.fOutX = 0;  // Disable XON/XOFF
        dcb.fInX = 0;
        dcb.fDtrflow = 0; // Disable DTR flow control
        dcb.fRtsflow = 0; // Disable RTS flow control
        dcb.EofChar = (char)VS_TERM;  // Use End of Message
        dcb.fChEvt = 1; // Signal event when EvtChr is received
        dcb.EvtChar = (char)VS_TERM; // VISCA end-of-message char

        if (CommErr == 0) {
          CommErr = SetCommState(&dcb);
          if (CommErr < 0) {
```

```
            AfxMessageBox("SetComm State error", MB_OKIMB_ICONINFORMATION,0);
          }
        }

        if (CommErr == 0){
                SetCommEventMask(idComm, EV_RXFLAG);
                CommErr = EnableCommNotification(idComm,GetSafeHwnd() , 1, 1);
        }

#ifdef _DEBUG
                afxDump << "m_InitComm \n";
                if(CommErr == 0)
                        afxDump << "EnableCommErr \n";
#endif

        m_FlushComm(OUTPUT_Q);
        m_FlushComm(INPUT_Q);
        return(CommErr);
}

////////////////
// Write string to port
// This writes the VISCA message and sets a timer.
int     CBridgeView::m_WriteComm(char* pS, UINT length)
{  UINTi;
        char    oBuf[30];

        if (CommErr >= 0){
                CommErr = WriteComm(idComm, pS,length);
                if (CommErr < 0) {
                        GetCommError(idComm, NULL);
                AfxMessageBox("Write Error", MB_OKIMB_ICONINFORMATION,0);
                }else{
                        m_complete = FALSE;
                        // Set a timer and then wait for message
                        // If timer message comes, then timedOut
                        // else CommNotify got a message.
                        // Note: Timeout is specified in milliseconds


                        CString oS("Sent ");
                        for(i = 0;i<length;i++, pS++){
                                oS += _itoa((int)*pS, oBuf, 16);
                                oS += " : ";
                        }
#ifdef _DEBUG
                        afxDump << oS << "\n";
                        char            buf[20];
                        COMSTAT                 cstat;
        int                     result;
        result = GetCommError(idComm, (COMSTAT FAR*) &cstat);
        afxDump << "Comstat result " << _itoa(result,buf, 16) << "\n";
#endif
                        /**
                        AfxMessageBox(oS, MB_OKIMB_ICONINFORMATION,0);
                        **/
```

```
                    SetTimer(VS_TIMEOUT_ID, VS_TIMEOUT * 1000,NULL);
                }
        }

        return(CommErr);
}

//////////
// Flush the port
int     CBridgeView::m_FlushComm(UINT queueNum)
{
// queueNum= 0 -> transmission.
// queueNum= 1 -> receive.
        CommErr = FlushComm(idComm, queueNum);
                if (CommErr != 0) {
                        GetCommError(idComm, NULL);
            AfxMessageBox("Flush Error", MB_OKIMB_ICONINFORMATION,0);
          }

        return(CommErr);
}


//////////
// Read the port
int     CBridgeView::m_ReadComm(char* resultBuf, UINT numChar)
{
        COMSTAT             stat;
        int                 nChar;

        nChar = ReadComm(idComm, resultBuf, numChar);

        CommErr = GetCommError(idComm, &stat);
        if (CommErr < 0) {
            AfxMessageBox("Read Error", MB_OKIMB_ICONINFORMATION,0);
            return(CommErr);
        }else return(nChar);
}


/////////////////
/// Close the port
int     CBridgeView::m_CloseComm(){
        if (idComm >= 0) {
                CommErr = CloseComm(idComm);
                if (CommErr < 0) {
            AfxMessageBox("Close Error", MB_OKIMB_ICONINFORMATION,0);
          }
        }

        return(CommErr);
}

////////////////////////////////////////////////////////////////
// IDS Startup code
// Performs the initialization and starts up real time IDS
```

```
// Add a timeout in case IDS is not available
void CBridgeView::startUpIDS()
{
#define         WAIT10          10000
        int     status;
        DWORD           pauseUntil;

        pauseUntil = WAIT10 + GetTickCount();

        do {
                strcpy(idsMessage, "status"); // Request status

                if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG
                        afxDump << "send failed \n";
#endif
                }

                if(recv(clientSock, (LPSTR)idsMessage, MESSAGE_SIZE, 0) ==
SOCKET_ERROR){
                        AfxMessageBox("recv failed",MB_OK, 0);
                }else  // Convert message string to int
                        status = atoi(idsMessage);

                if(pauseUntil < GetTickCount()){ // Timed out
                        AfxMessageBox("Reset IDS box and try again.", MB_OK |
MB_ICONEXCLAMATION, 0);
                        break;
                }

        }while(status!= READY);

        strcpy(idsMessage, "init");

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }

        strcpy(idsMessage, "histo(0xbf, 0x40, 7)");

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }

        strcpy(idsMessage, "spread(1)");

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
```

```
#endif
        }

   strcpy(idsMessage, "realtime");

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }

}


////////////////////////////////////////////////////////////////
// VISCA Message processing
//////////////////
// m_WriteMessage
// If m_complete == TRUE and there are no messages in the
// messageQ, then call m_WriteComm to write the message.
// Else, add the message to the messageQ for it to be picked out
// later by getVSMessage.
void CBridgeView:: m_WriteMessage(char* pMessage, UINT length)
{  UINT         i;

        if(m_complete && messageQ.IsEmpty())
                m_WriteComm(pMessage,  length);
        else {
                // Copy the pMessage into a CByteArray
                CByteArray* pCBA = new CByteArray();
                for(i=0; i< length; i++){
                        pCBA->Add((BYTE) pMessage[i]);
                }
                messageQ.AddTail(pCBA);
        }

}

//////////////////////
// getVSMessage
// If the messageQ is not empty, get the message at the front
// of the messageQ and call m_WriteComm
void CBridgeView::getVSMessage(void)
{       char    pMessage[VS_MAXPACKET];
        UINT    length, i;
        CByteArray*     pCBA;

        if(FALSE == messageQ.IsEmpty()){
                pCBA = (CByteArray*)messageQ.RemoveHead();  // Gets the head and
removes it
                length = pCBA->GetSize();
                for(i=0;i<length; i++){
                        // Copy into pMessage
                        pMessage[i] =  pCBA->GetAt(i);
                }
```

```
                // Remove all elements from pCBA
                pCBA->RemoveAll();
                m_WriteComm(pMessage,  length);
                delete []pCBA;
        }

}


////////////////
// m_ReadMessage
void CBridgeView:: m_ReadMessage(void)
{// Messages are terminated with VS_TERM character.
        char                    rBuf[50];
        char                    in[1];
        int                             nChar = 0,i;
        CTime                   lineTime;
        CTimeSpan               ts;
        BOOL                    timedOut = FALSE, reading = TRUE;


        lineTime = CTime::GetCurrentTime();
        while (reading){
                if((ts=CTime::GetCurrentTime() - lineTime).GetTotalSeconds() <
VS_TIMEOUT){
                        if((m_ReadComm(in,1)) == 1){ // Got the char
                                rBuf[nChar] = in[0];       // Append it to the receive string
                                nChar += 1;
                                // Check for VS_TERM
                                if((BYTE)in[0]  == VS_TERM)
                                        reading = FALSE;
                        }
                } else {
                AfxMessageBox("VISCA Timeout", MB_OKIMB_ICONINFORMATION,0);
                        timedOut = TRUE;
                        m_FlushComm(INPUT_Q);
                        reading = FALSE;
                }
        }

        if(timedOut == FALSE){
                // Copy message into m_LastVISCAMsg
#ifdef _DEBUG
                        int             j;
                        char    oBuf[30];
                        CString oS("Received ");
                        for(j = 0;j<nChar;j++){
                                oS += _itoa((int)rBuf[j], oBuf, 16);
                                oS += " : ";
                        }
                        afxDump << oS << "\n";
#endif
                for(i=0; i<nChar; i++) m_LastVISCAMsg[i] = rBuf[i];
                m_ParseMessage(rBuf, nChar);
        }
```

```
}

////////////////////////////////
// m_ParseMessage
// Parse the result messages
//
void    CBridgeView::m_ParseMessage(char* sReceive, UINT nChar)
{
        char    oBuf[20];
        CString oS;
        UINT            i;
        WORD errorCode;

        switch (sReceive[1] & 0xf0) {
                case VS_COMP:
                        m_complete = TRUE;

                        if(nChar > 3) // Don't parse simple completion messages
                        // Post a message to the appropriate dialog
                        // If message is from device 1 (case 0x10), send to VCRDlg
                        // If message is from device 2 (case 0x20), send to CameraDlg
                        // Note sReceive[0] is the VISCA address byte
                        switch (sReceive[0] & 0x70){
                                case 0x10:
                                        if(pVSCADlg->GetSafeHwnd() != NULL)
                                                pVSCADlg->SendMessage(WM_VISCAINQ
,nChar, (long)((LPSTR)m_LastVISCAMsg));
                                        else if(pVidSDlg->GetSafeHwnd() != NULL)
                                                pVidSDlg->SendMessage(WM_VISCAINQ
,nChar, (long)((LPSTR)m_LastVISCAMsg));
                                        break;
                                case 0x20:
                                        CameraDlg.SendMessage(WM_VISCAINQ ,nChar,
(long)((LPSTR)m_LastVISCAMsg));
                                        break;
                                default:

                                        oS = "Complete ";
                                        for(i = 0;i<nChar;i++){
                                                oS += _itoa((int)sReceive[i], oBuf, 16);
                                                oS += " : ";
                                        }
                                        AfxMessageBox(oS,
MB_OKIMB_ICONINFORMATION,0);

                                        break;
                        }
                        break;
                case VS_ERROR:
                        // There are 17 error conditions.
                        // The messages are stored as string resources.
                        // Post a message to the appropriate dialog
                        // If message is from device 1 (case 0x10), send to VCRDlg
                        // If message is from device 2 (case 0x20), send to CameraDlg
                        // Note sReceive[0] is the VISCA address byte
                        switch (sReceive[0] & 0x70){
```

```
                                case 0x10:
                                        if(pVSCAdlg->GetSafeHwnd() != NULL)
                                                pVSCAdlg->SendMessage(WM_VISCAERR
,nChar, (long)((LPSTR)m_LastVISCAMsg));
                                        else if(pVidSDlg->GetSafeHwnd() != NULL)
                                                pVidSDlg->SendMessage(WM_VISCAERR
,nChar, (long)((LPSTR)m_LastVISCAMsg));
                                        break;
                                case 0x20:
                                        CameraDlg.SendMessage(WM_VISCAERR ,nChar,
(long)((LPSTR)m_LastVISCAMsg));
                                        break;
                                default:
                                        errorCode = (WORD)sReceive[2];
                                        oS.LoadString((UINT)(VS_STRINGTABLE |
(UINT)errorCode));
                                        oS += "  ";
                                        for(i = 0;i<nChar;i++){
                                                oS += _itoa((int)sReceive[i], oBuf, 16);
                                                oS += " : ";
                                        }
                                        AfxMessageBox(oS,
MB_OKIMB_ICONINFORMATION,0);
                                        break;
                        }

                        m_complete = TRUE;
                        break;
                case VS_ACK:
                        // Don't do anything when we get an ACK.
                        // All commands are sent for immediate execution.
                        //AfxMessageBox("ACK", MB_OKIMB_ICONINFORMATION,0);
                        break;
                default:
                        // Broadcast messages
                        // Capture the address message to make sure both devices
                        // are connected to the net.
                        /**
                        oS = "Default ";
                                for(i = 0;i<nChar;i++){
                                oS += _itoa((int)sReceive[i], oBuf, 16);
                                oS += " : ";
                        }
                        AfxMessageBox(oS, MB_OKIMB_ICONINFORMATION,0);
                        **/
                        m_complete = TRUE;
                        break;
        }


}


//////////////////////
// VISCAInit
void CBridgeView::m_VISCAInit(void)
```

```
{
        char    pS[VS_MAXPACKET];

        // Initialize the m_complete
        m_complete = TRUE;

        // Send CLEAR
        pS[0] = VS_BROADCAST;
        pS[1] = VS_COMMAND1;
        pS[2] = VS_INTERFACE;
        pS[3] = (BYTE)0x01;
        pS[4] = VS_TERM;

        m_WriteMessage(pS, 5);

        // Send Address
        pS[0] = VS_BROADCAST;
        pS[1] = VS_ADDRESS;
        pS[2] = (BYTE)0x01;
        pS[3] = VS_TERM;

        m_WriteMessage(pS, 4);
}

////////////////////////////
/// SelectInput
void CBridgeView::m_SelectInput(char inLine)
{
        char    pS[VS_MAXPACKET];

        // Send Device Input select
        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x13;  // MD_InputSelect

        // Which video input, S-video, Line1, or Line2
        switch(inLine) {
                case 'S':
                        pS[4] = (BYTE)0x21;  // S-Video1
                        break;
                case '1':
                        pS[4] = (BYTE)0x11;  // Video1 line 1
                        break;
                case '2':
                        pS[4] = (BYTE)0x12;  // Video1 line 2
                        break;
        }


        //pS[5] = (BYTE)0x11;   // Audio line 1
        pS[5] = (BYTE)0x00;     // Muted Audio Input -- Audio from camera not
                                                // sent down. Use audio track for microphone
                                                // input so user can annotate video.
        pS[6] = VS_TERM;
```

```
        m_WriteMessage(pS, 7);


}


////////////////////////////////////
// imSave()
// Saves the DVA file saved in the RAM disk as a DIB file
// using the MIC Image Load and Image Save commands
// If dib is 1 save as dib otherwise save as dva
// If autonaming in effect, then compose name from base name,sequence number,
// and format extension.
void CBridgeView::imSave(BOOL dib)
{
#define         FILEDIAL        201

        WORD            result;
        char            *str;

        CVBControl      fileDialog;
        int                     fileIOErr = 0;
        char            seqStr[10];
        CString         fileName;

        //
        if(m_saveAutoNameButton){
                switch(m_saveDestination){
                        case HardDisk:
                                fileName = HDDIR;
                                break;
                        case RemovableDisk:
                                fileName = RDDIR;
                                break;
                        case FloppyDisk:
                                fileName = FDDIR;
                                break;
                }
                fileName += m_saveBaseName;
                fileName += itoa(m_saveSequenceNum, seqStr, 10);
                if(dib)
                        fileName += ".DIB";
                else
                        fileName += ".DVA";
                // Use the MIC Image Load
                CString CmdStr("Image Load ");
                CmdStr += '"';
                CmdStr += "E:\\TMPIMG.DVA"; // Have to use double \ because \ is an escape
char
                CmdStr += '"';
                CmdStr += " DVA";
                CmdStr.MakeUpper(); // File names have to be upper case
                str = CmdStr.GetBuffer(256);
                result = MICWriteStr(micID, (LPSTR)str );
```

```
                // Save as a true color DVA or DIB file
                CString SavStr("Image Save ");
                SavStr += '"';
                SavStr += fileName;
                SavStr += '"';

                if(dib)
                        SavStr += " DIB TRUE";
                else
                        SavStr += " DVA NTSC FRAME";

                SavStr.MakeUpper(); // File names have to be upper case
                str = SavStr.GetBuffer(256);
                result = MICWriteStr(micID, (LPSTR)str );
                // Increment the sequence num
                m_saveSequenceNum++;

        }else {
                // Create the VBX file dialog if not automatic
                CRect  dRect(0,0,100,100);
                result =
(WORD)fileDialog.Create((LPCSTR)"CMDIALOG.VBX;CommonDialog;Save Image",
                        // The name CommonDialog MUST be used or it won't create
                        WS_CHILD, dRect, this, FILEDIAL);
                        // The dRect is a dummy since the CommonDialog VBX determines the
rect.
                if (result){
                        if(dib)
                                fileDialog.SetStrProperty("DefaultExt", "DIB");
                        else
                          fileDialog.SetStrProperty("DefaultExt", "DVA");
                        fileDialog.SetStrProperty("DialogTitle", "Save the Image");
                        fileDialog.SetStrProperty("Filter",
                                "VideoLogic DVA (*.DVA)|*.DVA|Windows DIB (*.DIB)|*.DIB|All
Files (*.*)|*.*");
                        if(dib)
                                fileDialog.SetNumProperty("FilterIndex", 2);
                        else
                                fileDialog.SetNumProperty("FilterIndex", 1);
                        fileDialog.SetNumProperty("Flags", OFN_OVERWRITEPROMPT |
OFN_HIDEREADONLY);
                        switch(m_saveDestination){
                                case HardDisk:
                                        fileDialog.SetStrProperty("InitDir","C:\\Bridge"); // Have
to use double \ because \ is an escape char
                                        break;
                                case RemovableDisk:
                                        fileDialog.SetStrProperty("InitDir","D:\\");
                                        break;
                                case FloppyDisk:
                                        fileDialog.SetStrProperty("InitDir","A:\\");
                                        break;
                        }
                        fileDialog.SetNumProperty("CancelError",TRUE);

                        // At last lets open the dialog
```

```
                        fileDialog.SetNumProperty("Action", 2);

                        if(fileDialog.m_nError == 0 ){
                                RestoreWaitCursor();
                                // Use the MIC Image Load
                                CString CmdStr("Image Load ");
                                CmdStr += '"';
                                CmdStr += "E:\\TMPIMG.DVA"; // Have to use double \ because
\ is an escape char
                                CmdStr += '"';
                                CmdStr += " DVA";
                                CmdStr.MakeUpper(); // File names have to be upper case
                                str = CmdStr.GetBuffer(256);
                                result = MICWriteStr(micID, (LPSTR)str );

                                // Save as a true color DVA or DIB file
                                CString SavStr("Image Save ");
                                SavStr += '"';
                                SavStr += fileDialog.GetStrProperty("FileName");
                                SavStr += '"';

                                if(dib)
                                        SavStr += " DIB TRUE";
                                else
                                        SavStr += " DVA NTSC FRAME";

                                SavStr.MakeUpper(); // File names have to be upper case
                                str = SavStr.GetBuffer(256);
                                result = MICWriteStr(micID, (LPSTR)str );

                        } else
                                AfxMessageBox("Canceled image save.",
MB_OK|MB_ICONINFORMATION,0);
                } else
                        AfxMessageBox("File Save Error",
MB_OK|MB_ICONINFORMATION,0);
        }

}

//////////////////////////////////////////
// Byte to BCD
// The time and date functions use BCD (Binary coded decimal)
// Only for numbers up to 99.
BYTE   CBridgeView::byte2bcd(BYTE bytenum)
{
        BYTE   result, ones, tens;
        div_t   div_result;

   div_result = div(bytenum, 10);
   tens = (div_result.quot & 0x0f);
   ones = (div_result.rem & 0x0f);
   result = (BYTE)((tens << 4) | ones);

   return(result);
```

```
}

//////////////////////////////////////////
// Set the date and time of the VCR
void CBridgeView::m_setVCRDateTime()
{
        char    pS[VS_MAXPACKET];
        int             year10s, year1s, month, day, hour, minute, second;
        div_t   div_result;

        CTime ct = CTime::GetCurrentTime();

        div_result = div((ct.GetYear() % 100) , 10); // Just want the 10's and 1's
        year10s = div_result.quot;
        year1s = div_result.rem;
        month = ct.GetMonth();
        day = ct.GetDay();
        hour = ct.GetHour();
        minute = ct.GetMinute();
        second = ct.GetSecond();

        // Send Date Set
        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x16;  // MD_ClockSet
        pS[4] = (BYTE)0x41;     // Date Set
        pS[5] = (BYTE) year10s;
        pS[6] = (BYTE) year1s;
        pS[7] = (BYTE) byte2bcd((BYTE) month);
        pS[8] = (BYTE) byte2bcd((BYTE) day);
        pS[9] = VS_TERM;


        m_WriteMessage(pS,  10);


        // Send Time Set
        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x16;  // MD_ClockSet
        pS[4] = (BYTE)0x42;     // Time Set
        pS[5] = (BYTE) byte2bcd((BYTE)hour);
        pS[6] = (BYTE) byte2bcd((BYTE)minute);
        pS[7] = (BYTE) byte2bcd((BYTE)second);
        pS[8] = (BYTE) 0;
        pS[9] = VS_TERM;


        m_WriteMessage(pS,  10);

}
```

```
//////////////////////////////////////////////////////////////////////////
// CBridgeView message handlers

void CBridgeView::OnVISCAActive()
{
        CVSCAdlg        VCRDlg;
        pVSCAdlg = &VCRDlg;

        CMainFrame* parent = (CMainFrame*)GetParentFrame();

        // Kill the joystick timer
        parent->KillJoy();

        m_InitComm();

    // Pass the pointer to this class to the dialog class
    // so that it can access the communication member functions.
        VCRDlg.SetViewPtr(this);

        // Initialize the VISCA net
        m_VISCAInit();

        // Set VCR Date and Time
        m_setVCRDateTime();

        if (VCRDlg.DoModal() == IDOK)
        {
                m_CloseComm();
        }

        pVSCAdlg = NULL;
        // Start the joystick timer
        parent->StartJoy();
}

//////////////////////////////////////////////////////////////////////////

void CBridgeView::OnCAMERAActive()
{
        m_InitComm();

    // Pass the pointer to this class to the dialog class
    // so that it can access the communication member functions.
        CameraDlg.SetViewPtr(this);

        // Initialize the VISCA net
        m_VISCAInit();

        if (CameraDlg.DoModal() == IDOK)
        {
                m_CloseComm();
        }
```

```
}

void CBridgeView::OnVideoSelect()
// Video selection dialog activation
{
        CVidSDlg        VidSDlg;
        // Set a pointer so that we can send messages to it
        pVidSDlg = &VidSDlg;

        m_InitComm();

   // Pass the pointer to this class to the dialog class
   // so that it can access the communication member functions.
        VidSDlg.SetViewPtr(this);

        // Initialize the VISCA net
        m_VISCAInit();

        if (VidSDlg.DoModal() == IDOK)
        {
                m_CloseComm();
        }

        pVidSDlg = NULL;

}



void CBridgeView::OnUserOptions()
{
        CUsrOptDg       UsrOptDlg;

        int                     val;

        UsrOptDlg.DIBButton = m_saveFormat;
        UsrOptDlg.autoNameButton  = m_saveAutoNameButton;
        UsrOptDlg.baseName = m_saveBaseName;
        UsrOptDlg.sequenceNum = m_saveSequenceNum;
        UsrOptDlg.destination = m_saveDestination;

        if (UsrOptDlg.DoModal() == IDOK)
        {
#ifdef _DEBUG
                afxDump << UsrOptDlg.baseName;
                val = UsrOptDlg.DIBButton;
#endif
                m_saveFormat = UsrOptDlg.DIBButton;
                m_saveAutoNameButton = UsrOptDlg.autoNameButton;
                m_saveBaseName = UsrOptDlg.baseName;
                m_saveSequenceNum = UsrOptDlg.sequenceNum;
                m_saveDestination = UsrOptDlg.destination;
        }

}
```

```
//////////////////////////
// The OnCommNotify function monitors the comm port
LRESULT CBridgeView::OnCommNotify(WPARAM wparam, LPARAM lparam)
{
        if(LOWORD(lparam) & CN_EVENT){
                GetCommEventMask(idComm, EV_RXFLAG);
                m_ReadMessage();
                if(m_complete){
                        KillTimer(VS_TIMEOUT_ID);
                        // dispatch any waiting messages
                        getVSMessage();
                }


        }

        if(LOWORD(lparam) == CN_TRANSMIT){
        //AfxMessageBox("Flush", MB_OKIMB_ICONINFORMATION,0);
        //m_FlushComm(OUTPUT_Q);

        }

        if(LOWORD(lparam) & CN_RECEIVE){
#ifdef _DEBUG
                afxDump << "Got char on CN_RECEIVE \n";
#endif
        }

 return ( TRUE ) ;
}


//////////////////////////////////
// Respond to WM_TIMER messages

void CBridgeView::OnTimer(UINT nIDEvent)
{

        if(nIDEvent == VS_TIMEOUT_ID){
                KillTimer(nIDEvent);

#ifdef _DEBUG
                afxDump << "Packet timeout \n";

#endif

                // Try to read the queue
    m_ReadMessage();
                if(m_complete){
                        // dispatch any waiting messages
                        getVSMessage();
                }else {
                        // Set m_complete to true
                        m_complete = TRUE;
                        // dispatch any waiting messages
```

```
                getVSMessage();
        }

#ifdef _KILLALL
        // Kill all messages in the queue
        CByteArray*     pCBA;

        // If there are any messages in the messageQ, delete them.
        while(messageQ.GetCount() > 0){
                pCBA = (CByteArray*)messageQ.RemoveHead();  // Gets the head and
removes it
                // Remove all elements from pCBA
                pCBA->RemoveAll();
                delete []pCBA;
        }

        messageQ.RemoveAll(); // Just to make sure;
#endif


    }

     CView::OnTimer(nIDEvent);

}

//////////////////////////////
// Handle the Clicking of the Snap Shot button
// Actions:
// Set DPOS to (0,0) and DSIZE to FULL
// Save image to temporary RAM disk file
// Restore old DPOS and DSIZE
// Ask for name of file or CANCEL
// Copy temporary file to disk as DIB.
void CBridgeView::OnSnapShot()
{
    WORD result;
    char    *str;

    struct _diskfree_t      freeStruct;
    DWORD       space;
    DWORD length;
    unsigned int    spaceResult;

    // Set DPOS
    result = MICWriteStr(micID, (LPSTR)"FStore $VWin DPOS 3 0 0 ABSOLUTE");

    // Set DSIZE
    result = MICWriteStr(micID, (LPSTR)"FStore $VWin DSIZE 3 FULL");

    // Set the cursor to an hourglass
    BeginWaitCursor();

    CString CmdStr("FStore $VWin SAVE 3 ");
    CmdStr += '"';
    CmdStr += "E:\\TMPIMG.DVA";  // Have to use double \ because \ is an escape char
```

```
        CmdStr += '"';
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
    // Save image to temporary RAM disk file
        result = MICWriteStr(micID, (LPSTR)str );

        // Restore Image to window size
        MoveVideoWindow();

        // Check to see that there is enough space on the destination device
        if(m_saveFormat) // 24-bit DIB
                length = 922000;
        else {
                CFile inFile("E:\\TMPIMG.DVA",CFile::modeRead|CFile::shareExclusive);
                length = inFile.GetLength();
                length += 1000; // A little extra
        }
        switch(m_saveDestination){
                case HardDisk:
                        spaceResult = _dos_getdiskfree(3, &freeStruct); // C: is disk 3
                        break;
                case RemovableDisk:
                        spaceResult = _dos_getdiskfree(4, &freeStruct); // D: is disk 4
                        break;
                case FloppyDisk:
                        spaceResult = _dos_getdiskfree(1, &freeStruct); // A: is disk 1
                        break;
        }
        space = (DWORD)(freeStruct.avail_clusters) * (DWORD)(freeStruct.sectors_per_cluster)
*
                        (DWORD)(freeStruct.bytes_per_sector);

        if((spaceResult==0) &&(space > length)){   // DVA Format
                // Save the image to a disk file
                imSave(m_saveFormat);    // Set by user options
        } else {
                switch(m_saveDestination){
                        case HardDisk:
                                AfxMessageBox("Not enough disk space on Hard Drive.\
Move images to floppy or removable disk.", MB_OK | MB_ICONEXCLAMATION, 0);
                                break;
                        case RemovableDisk:
                                AfxMessageBox("Not enough disk space on Removable Drive.\
Insert new removable disk.", MB_OK | MB_ICONEXCLAMATION, 0);
                                break;
                        case FloppyDisk:
                                AfxMessageBox("Not enough disk space on Floppy Disk.\
Insert new floppy disk.", MB_OK | MB_ICONEXCLAMATION, 0);
                                break;
                }

        }

        // Restore old cursor
        EndWaitCursor();
```

```
}


void CBridgeView::OnIds()
{
        CIDSDlg                 IDSDlg;


        IDSDlg.clientSock = clientSock;

        IDSDlg.imExSock = imExSock;

        // Pass the pointer to this class to the dialog class
  // so that it can access the communication member functions.
        IDSDlg.SetViewPtr(this);

        if (IDSDlg.DoModal() == IDOK)
        {

        }


}

//////////////////////////////////////////////////////////////

void CBridgeView::OnImageProc()
{
        CImProcDlg              ImProcDlg;

        // Pass the pointer to this class to the dialog class
  // so that it can access the communication member functions.
        ImProcDlg.SetViewPtr(this);

        if (ImProcDlg.DoModal() == IDOK)
        {

        }


}

//////////////////////////////////////////////////////////////

void CBridgeView::OnUserHelp()
{
        // TODO: Add your command handler code here

}

void CBridgeView::OnUserLevel()
{
        // TODO: Add your command handler code here

}
```

```
SOCKET CBridgeView::openPort(u_short portNum)
{
   WORD              wReqVer;
   int               err;
   WSADATA           wsaData;
   SOCKET            clientSock;

        wReqVer = MAKEWORD(1,1);

        // Check version of DLL
        err = WSAStartup(wReqVer, &wsaData);
        if(err != 0) {
#ifdef _DEBUG
                afxDump << "WSA Startup failed \n";
#endif
                return(FALSE);
        }

        memset(&serverAddr, 0, sizeof(serverAddr));
        memset(&clientAddr, 0, sizeof(clientAddr));

        clientSock = socket(PF_INET, SOCK_STREAM, 0);

        if(clientSock == INVALID_SOCKET){
#ifdef _DEBUG
                afxDump << "socket failed \n";
#endif
                return(FALSE);
        }



        // Bind the internet address and the socket
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(portNum);

        if((serverAddr.sin_addr.s_addr = inet_addr((LPSTR)HOSTNAME)) == INADDR_NONE ){
/* resolve destination host name */
#ifdef JOJO
if ((dstIP = (inet_addr(host))/*.s_addr*/ ) == INADDR_NONE) {
          if ((dstHost = gethostbyname(host)) == NULL)
                  return(FALSE);
          dstIP = *((LPin_name)(dstHost->h_addr));
        }

        if((serverAddr.sin_addr.s_addr = *((LPIN_ADDR)(gethostbyname((LPSTR)"robin")-
>h_addr))) == INADDR_NONE ){

#endif  // JOJO

#ifdef _DEBUG
                afxDump << "invalid host  \n";
#endif
           closesocket(clientSock);
```

```
            return(FALSE);
        }
#ifdef _DEBUG
            afxDump << "Server addr " << inet_ntoa(serverAddr.sin_addr) << "\n";
#endif


        if(connect(clientSock, (LPSOCKADDR)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR){

#ifdef _DEBUG
            netErr = WSAGetLastError();
            printErr(netErr);
            afxDump << "connect failed \n";
#endif
            closesocket(clientSock);
            return(FALSE);
        }

#ifdef _DEBUG
            afxDump << "Connected \n";
#endif

        return(clientSock);



}


//////////////////////////
void CBridgeView::closePort(SOCKET clientSock)
{

  closesocket(clientSock);
//      WSACleanup();

}

void CBridgeView::cleanupWSA()
{
        WSACleanup();
}


//////////////////////////


//////////////////////////
void CBridgeView::printErr(int nErr)
{
#ifdef _DEBUG
        switch(nErr){
            case WSANOTINITIALISED:
                afxDump << "WSANOTINITIALISED \n";
                break;
```

```
                case WSAENETDOWN:
                        afxDump << "WSAENETDOWN \n";
                        break;
                case WSAEADDRINUSE:
                        afxDump << "WSAEADDRINUSE \n";
                        break;
                case WSAEINTR:
                        afxDump << "WSAEINTR \n";
                        break;
                case WSAEINPROGRESS:
                        afxDump << "WSAEINPROGRESS \n";
                        break;
                case WSAEADDRNOTAVAIL:
                        afxDump << "WSAEADDRNOTAVAIL \n";
                        break;
                case WSAEAFNOSUPPORT:
                        afxDump << "WSAEAFNOSUPPORT \n";
                        break;
                case WSAECONNREFUSED:
                        afxDump << "WSAECONNREFUSED \n";
                        break;
                case WSAEDESTADDRREQ:
                        afxDump << "WSAEDESTADDRREQ \n";
                        break;
                case WSAEFAULT:
                        afxDump << "WSAEFAULT \n";
                        break;
                case WSAEINVAL:
                        afxDump << "WSAEEINVAL \n";
                        break;
                case WSAEISCONN:
                        afxDump << "WSAEISCONN \n";
                        break;
                case WSAEMFILE:
                        afxDump << "WSAEMFILE \n";
                        break;
                case WSAENETUNREACH:
                        afxDump << "WSAENETUNREACH \n";
                        break;
                case WSAENOBUFS:
                        afxDump << "WSAENOBUFS \n";
                        break;
                case WSAENOTSOCK:
                        afxDump << "WSAENOTSOCK \n";
                        break;
                case WSAETIMEDOUT:
                        afxDump << "WSAETIMEDOUT \n";
                        break;
                case WSAEWOULDBLOCK:
                        afxDump << "WSAEWOULDBLOCK \n";
                        break;

        }
#endif
}
```

```
void CBridgeView::pause(DWORD val)
{
        DWORD           pauseUntil;

        pauseUntil = val + GetTickCount();

        while(pauseUntil >= GetTickCount());
}
```

```
#include "winsock.h"


// bridgvw.h : interface of the CBridgeView class
//

#define PALVERSION  0x300
#define TransColorIndex         50

#define MESSAGE_SIZE        50

class CVidSDlg;
class CVSCAdlg;

////////////////////////////////////////////////////////////////////////

class CBridgeView : public CView
{
protected: // create from serialization only
        CBridgeView();
        DECLARE_DYNCREATE(CBridgeView)

// Attributes
public:
        CBridgeDoc* GetDocument();

        // Communications support
        DCB             dcb;
        int                     idComm;
        int                     CommErr;
#ifdef _DEBUG
        void            DumpDCB (DCB* pDCB);
#endif

        // Communications Support Functions
        int                     m_InitComm();
        int                     m_WriteComm(char* pS, UINT length);
        int                     m_ReadComm(char* resultBuf,UINT numChar);
        int                     m_FlushComm(UINT queueNum);
        int                     m_CloseComm();

        // VISCA Message processing
        CPtrListmessageQ;
        void            getVSMessage(void);
        BOOL            m_ack, m_complete;
        char            m_LastVISCAMsg[100];
        void            m_WriteMessage(char* pMessage, UINT length);
        void            m_ReadMessage(void);
        void            m_ParseMessage(char* sReceive, UINT nChar);
        void            m_VISCAInit(void);
        void            m_SelectInput(char S_or_L);
        void            m_setVCRDateTime(void);
        BYTE            byte2bcd(BYTE bytenum);

        // MIC support
        CPalette        m_Pal;
```

```
        DWORD           micID;

        BOOL        m_goodPal;

        BOOL        CreateVideoPalette();

        BOOL        AssignDVA();
        void        DeassignDVA();
        void        DeInitVideoWindow();
        void        MoveVideoWindow();
        void        micInq(LPSTR szMICCmdBuf, LPSTR szMICRspBuf, WORD size);
        WORD        micWrite(LPSTR szMICCmdBuf);

        void        freezeVideo(BOOL freezeUnfreeze);

        void        imSave(BOOL dib);

        void        initMic();
        void        closeMic();

        // User Option Dialog support
        BOOL        m_saveFormat;
        BOOL        m_saveAutoNameButton;
        CString     m_saveBaseName;
        unsigned int        m_saveSequenceNum;
        unsigned int        m_saveDestination;

        // IDS Ether support
        SOCKET              clientSock, imExSock;
        void        printErr(int nErr);
        void        pause(DWORD val);
        SOCKET              openPort(u_short portNum);
        void        closePort(SOCKET clientSock);
        void        cleanupWSA();
        SOCKADDR_IN                 serverAddr, clientAddr;
        int                         netErr;
        void        startUpIDS(void);
        char        idsMessage[MESSAGE_SIZE];


// Operations
public:

// Implementation
public:
        virtual ~CBridgeView();
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Printing support
protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
```

```
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Generated message map functions
public:
        CVidSDlg*       pVidSDlg;
        CVSCAdlg*       pVSCAdlg;
        //{{AFX_MSG(CBridgeView)
        afx_msg void OnVISCAActive();
        afx_msg void OnCAMERAActive();
        afx_msg void OnVideoSelect();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnSnapShot();
        afx_msg void OnImageProc();
        afx_msg void OnUserHelp();
        afx_msg void OnUserLevel();
        afx_msg void OnUserOptions();
        afx_msg void OnIds();
        //}}AFX_MSG
        afx_msg LRESULT OnCommNotify(WPARAM wparam, LPARAM lparam);

        DECLARE_MESSAGE_MAP()

        // Painting Support
        void TransparentPaint(CDC* pDC);
};

#ifndef _DEBUG          // debug version in bridgvw.cpp
inline CBridgeDoc* CBridgeView::GetDocument()
  { return (CBridgeDoc*) m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
// camdlg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "camdlg.h"

#include "bridgdoc.h"

#include "bridgvw.h"

#include "viscadev.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

static UINT NEAR WM_VISCAINQ = RegisterWindowMessage((LPCSTR)"VISCA_INQ");

/////////////////////////////////////////////////////////////////////////////
// camdlg dialog

CCamDlg::CCamDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CCamDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CCamDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}

void CCamDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CCamDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCamDlg, CDialog)
        //{{AFX_MSG_MAP(CCamDlg)
        ON_VBXEVENT(VBN_CLICK, IDC_Zoom, OnClickZoom)
        ON_VBXEVENT(VBN_CLICK, IDC_Wide, OnClickWide)
        ON_VBXEVENT(VBN_CLICK, IDC_CHECKDataScreen, OnClickCHECKDataScreen)
        ON_REGISTERED_MESSAGE(WM_VISCAINQ, OnViscaInq)
        ON_VBXEVENT(VBN_CLICK, IDC_Far, OnClickFar)
        ON_VBXEVENT(VBN_CLICK, IDC_Near, OnClickNear)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Helper functions

void CCamDlg::SetViewPtr(CBridgeView* p)
{
```

```
        pBridgeView = p;

}


////////////////////////////////////////////////////////////////////////
// CCamDlg message handlers


void CCamDlg::OnClickZoom(UINT, int, CWnd*, LPVOID)
{
        static int state = 0;

        char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x02);    // Send to Device 2
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x07;  // MD_CameraZoom

        if(state ==0){
                state = 1;
                pS[4] = (BYTE)0x02;  // Motor tele
        } else {
                state = 0;
                pS[4] = (BYTE)0x00;  // Motor stop
        }
        pS[5] = VS_TERM;


        pBridgeView->m_WriteMessage(pS, 6);
}

void CCamDlg::OnClickWide(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];
        static int      state = 0;


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x02);    // Send to Device 2
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x07;  // MD_CameraZoom

        if(state ==0){
                state = 1;
                pS[4] = (BYTE)0x03;  // Motor wide
        } else {
                state = 0;
                pS[4] = (BYTE)0x00;  // Motor stop
        }
        pS[5] = VS_TERM;
```

```
        pBridgeView->m_WriteMessage(pS, 6);

}

void CCamDlg::OnClickCHECKDataScreen(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];

        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x02);    // Send to Device 2
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x10;  // Toggle Display ON/OFF
        pS[4] = (BYTE)0x45;  // Motor stop
        pS[5] = VS_TERM;


        pBridgeView->m_WriteMessage(pS, 6);

}

////////////////////////////////////
// OnViscaInq(WPARAM wparam, LPARAM lparam)
// When an inquiry message completes, VSCAView sends a message
// with the result of the inquiry if it completed.
// It is up to this function to parse the message and perform
// desired action.
LRESULT CCamDlg::OnViscaInq(WPARAM wparam, LPARAM lparam)
{


   return((long)0);
}

#define FOCUSCONTROL
#ifdef FOCUSCONTROL
void CCamDlg::OnClickNear(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];
        static int      state = 0;


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x02);    // Send to Device 2
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x08;  // MD_CameraFocus

        if(state ==0){
                state = 1;
                pS[4] = (BYTE)0x03;  // Motor near
        } else {
                state = 0;
                pS[4] = (BYTE)0x00;  // Motor stop
        }
        pS[5] = VS_TERM;
```

```
        pBridgeView->m_WriteMessage(pS, 6);

}


void CCamDlg::OnClickFar(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];
        static int      state = 0;


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x02);    // Send to Device 2
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x08;  // MD_CameraFocus

        if(state ==0){
                state = 1;
                pS[4] = (BYTE)0x02;  // Motor far
        } else {
                state = 0;
                pS[4] = (BYTE)0x00;  // Motor stop
        }
        pS[5] = VS_TERM;


        pBridgeView->m_WriteMessage(pS, 6);


}
#endif
```

```
// camdlg.h : header file
//

class CBridgeView;

/////////////////////////////////////////////////////////////////////////////
// camdlg dialog

class CCamDlg : public CDialog
{
// Construction
public:
        CCamDlg(CWnd* pParent = NULL);      // standard constructor

// Called by View
        CBridgeView*            pBridgeView;
        void SetViewPtr(CBridgeView* p);

// Dialog Data
        //{{AFX_DATA(CCamDlg)
        enum { IDD = IDD_CAMERAdlg };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CCamDlg)
        afx_msg void OnClickZoom(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickWide(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickCHECKDataScreen(UINT, int, CWnd*, LPVOID);
        afx_msg LRESULT OnViscaInq(WPARAM wparam, LPARAM lparam);
        afx_msg void OnClickFar(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickNear(UINT, int, CWnd*, LPVOID);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()



public:
        // VISCA Support
};
```

```
// idsadvdg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "idsdlg.h"
#include "idsadvdg.h"


#include "knifevbx.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIDSAdvDg dialog

CIDSAdvDg::CIDSAdvDg(CWnd* pParent /*=NULL*/)
        : CDialog(CIDSAdvDg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CIDSAdvDg)
        m_Cont_Val = NULL;
        //}}AFX_DATA_INIT
}

void CIDSAdvDg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CIDSAdvDg)
        DDX_VBControl(pDX, IDC_CONT_VAL, m_Cont_Val);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CIDSAdvDg, CDialog)
        //{{AFX_MSG_MAP(CIDSAdvDg)
        ON_VBXEVENT(VBN_CLICK, IDC_IDS_RESET, OnIdsReset)
        ON_WM_HSCROLL()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIDSAdvDg Functions


/////////////////////////////////////////////////////////////////////////////
// CIDSAdvDg message handlers

void CIDSAdvDg::OnIdsReset(UINT, int, CWnd*, LPVOID)
{
        // Init, end old task, start new task;
        strcpy(idsMessage, "init");

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG
```

```
                afxDump << "send failed \n";
#endif
        }

        pIDSDlg->SetHisto(Contrast);    // Set Contrast

        strcpy(idsMessage, "endreal");    // End old task

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }

        strcpy(idsMessage, "realtime");       // Start new task

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }


}


#define         SCROLL_LO   1
#define         SCROLL_HI   8
////////////////////////////////
// This handles the edge contrast scroll bar on the dialog
// Because the scroll bar pointer is passed as an argument,
// I cannot set some of the parameters of the scroll bar.
// Since it is a control scroll its range is 0. I want to set it
// to 0 to 255, but it jumps to 255 when you try to move the
// control box the first time. Add an if statement to watch for
// big nPos
void CIDSAdvDg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{

        char    sCont[3];

        pScrollBar->SetScrollRange(SCROLL_LO, SCROLL_HI, TRUE);

        switch(nSBCode){
                case SB_LINERIGHT:
                        if(Contrast < SCROLL_HI) {
                                Contrast++;
                                pScrollBar->SetScrollPos((int) Contrast, TRUE);
                                m_Cont_Val->SetStrProperty("Caption", _itoa(Contrast, sCont,
10));
                        }
                        break;
                case SB_LINELEFT:
```

```
                        if(Contrast > SCROLL_LO) {
                                Contrast--;
                                pScrollBar->SetScrollPos((int) Contrast, TRUE);
                                m_Cont_Val->SetStrProperty("Caption", _itoa(Contrast, sCont,
10));
                        }
                        break;
                case SB_PAGERIGHT:
                        break;
                case SB_PAGELEFT:
                        break;
                case SB_THUMBPOSITION:
                        if((nPos >= SCROLL_LO) && (nPos <= SCROLL_HI)){
                                pScrollBar->SetScrollPos((int) nPos, TRUE);
                                Contrast = nPos;
                                m_Cont_Val->SetStrProperty("Caption", _itoa(nPos, sCont, 10));

                        } else  pScrollBar->SetScrollPos(SCROLL_LO, TRUE);
                        break;
                default:
                        break;
        }

        pIDSDlg->SetHisto(Contrast);



        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

BOOL CIDSAdvDg::OnInitDialog()
{
        CScrollBar*     ContScroll;
        char    sCont[3];

        CDialog::OnInitDialog();

        // Use GetDlgItem to get the scroll bar handle and init the scroll bar.
        if((ContScroll = (CScrollBar*)GetDlgItem(IDC_SCROLLBAR1)) != NULL){
                ContScroll->SetScrollRange(SCROLL_LO, SCROLL_HI, FALSE);
                ContScroll->SetScrollPos(Contrast, TRUE);
                m_Cont_Val->SetStrProperty("Caption", _itoa(Contrast, sCont, 10));
        } else
                AfxMessageBox("Didn't init", MB_OK, 0);

        return TRUE;  // return TRUE  unless you set the focus to a control
}
```

```
// idsadvdg.h : header file
//

#include        "winsock.h"

#define         MESSAGE_SIZE        50
#define         DEFAULT_CONTRAST 6

static UINT             Contrast = DEFAULT_CONTRAST; // Edge Contrast

class   CIDSDlg;

/////////////////////////////////////////////////////////////////////////////
// CIDSAdvDg dialog

class CIDSAdvDg : public CDialog
{
// Construction
public:
        CIDSAdvDg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CIDSAdvDg)
        enum { IDD = IDD_IDS_ADV_DLG };
        CVBControl*     m_Cont_Val;
        //}}AFX_DATA

        SOCKET                  clientSock;
        SOCKET                  imExSock;
        // The ids control messages will be MESSAGE_SIZE long
        char            idsMessage[MESSAGE_SIZE];

        //void          SetHisto(UINT optionNum);
        CIDSDlg*        pIDSDlg;


// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CIDSAdvDg)
        afx_msg void OnIdsReset(UINT, int, CWnd*, LPVOID);
        afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// idsdlg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "idsdlg.h"
#include "string.h"

#include "idsadvdg.h"

#include "knifevbx.h"
#include "bridgdoc.h"
#include "bridgvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIDSDlg dialog

CIDSDlg::CIDSDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CIDSDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CIDSDlg)
        m_processPercent = NULL;
        m_PicBuf1 = NULL;
        m_hShift = 0;
        m_vShift = 0;
        //}}AFX_DATA_INIT
}

void CIDSDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CIDSDlg)
        DDX_VBControl(pDX, IDC_PROCESS_PERCENT, m_processPercent);
        DDX_VBControl(pDX, IDC_PICBUF1, m_PicBuf1);
        //}}AFX_DATA_MAP
        /*DDX_Text(pDX, IDC_hShiftEdit, m_hShift);
        DDV_MinMaxInt(pDX, m_hShift, -128, 128);
        DDX_Text(pDX, IDC_vShiftEdit, m_vShift);
        DDV_MinMaxInt(pDX, m_vShift, -128, 128); */
}

BEGIN_MESSAGE_MAP(CIDSDlg, CDialog)
        //{{AFX_MSG_MAP(CIDSDlg)
        ON_VBXEVENT(VBN_CLICK, IDC_IDS_ADV, OnIdsAdv)
        ON_VBXEVENT(VBN_CLICK, IDC_PROC_IMG, OnProcImg)
        ON_VBXEVENT(VBN_CLICK, IDC_PROC_SAV_IMG, OnProcSavImg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
```

```
// CIDSDlg utility functions
void CIDSDlg::ZeroMsgBuf(void)
{
	int		i;

	for(i=0;i<MESSAGE_SIZE; i++) idsMessage[i] = NULL;
}

void CIDSDlg::SetViewPtr(CBridgeView* p)
{

	pBridgeView = p;

}

//////////////////////////////////////////////////

void CIDSDlg::VHShift(int i,char* lineBuf)
// Using member variables m_hShift and m_vShift, adjust output image.
{
	int		j,vs;
	long	scanLine[512];
	//char	sBuf[512];

	/* Shift the lines horizontally */
	if(m_hShift < 0){
		for(j=0; j< 512 + m_hShift; j++) scanLine[j- m_hShift] = (unsigned long)lineBuf[j];
		for(j=512+m_hShift;j<512;j++) scanLine[j-(512+m_hShift)] = (unsigned
long)lineBuf[j];
	}

	if(m_hShift > 0){
		for(j=0; j< 512 - m_hShift; j++) scanLine[j] = (unsigned long)lineBuf[j +
m_hShift];
		for(j=512-m_hShift;j<512;j++) scanLine[j] = (unsigned long)lineBuf[j - (512 -
m_hShift)];
	}

	if(m_hShift == 0) for(j=0;j<512;j++) scanLine[j] = (unsigned long)lineBuf[j];

	/* Shift the whole image vertically */

	if(m_vShift < 0)
		if((i + m_vShift) >= 0) vs = i + m_vShift;
		else vs = 512 + m_vShift + i;

	if(m_vShift > 0)
		if((i + m_vShift) < 512) vs = i + m_vShift;
		else vs = (m_vShift + i) - 512;

	if(m_vShift == 0)  vs = i;


	imkPutScanLine((short)m_PicBuf1->GetNumProperty("FullPicture"),
		(int)vs, (LPPE)scanLine);
```

```
}

//////////////////////////////////////////////////////////

void CIDSDlg::SetHisto(UINT optionNum)
{
        // The option numbers are based on the original selections
        // used by S. Strang
        switch(optionNum){
                case 1: strcpy(idsMessage, "histo(0x80, 0x7f, 1)");      break;
                case 2: strcpy(idsMessage, "histo(0x81, 0x7e, 2)");      break;
                case 3: strcpy(idsMessage, "histo(0x83, 0x7c, 3)");      break;
                case 4: strcpy(idsMessage, "histo(0x87, 0x78, 4)");      break;
                case 5: strcpy(idsMessage, "histo(0x8f, 0x70, 5)");      break;
                case 6: strcpy(idsMessage, "histo(0x9f, 0x60, 6)");      break;
                case 7: strcpy(idsMessage, "histo(0xbf, 0x40, 7)"); break;
                case 8: strcpy(idsMessage, "histo(0xcf, 0x60, 6)");      break;
        }

        if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
        }
}

//////////////////////////////////////////////////////////


void CIDSDlg::IdsSend()
// Move image from IDS to the Host computer via the imExSock ethernet socket.
{
        char    lineBuf[512];
        long    palent[256], scanLine[512];
        unsigned long           i,j;
        PALETTEENTRY        pal;

        sendIdsMsg("sendfrm");

        pal.peRed = 0;
        pal.peGreen = 0;
        pal.peBlue = 0;
        pal.peFlags = NULL;

        /* Init the image */
        m_PicBuf1->SetNumProperty("FullPicture",
                imkInit(8,512,512,(VBCOLOR)pal));

        /* Set palette entries to be uniform*/
        for(i=0;i<256;i++) palent[i] = (unsigned long) RGB(i,i,i);

        m_PicBuf1->SetNumProperty("Palette",
                (short)imkSetPalette(256, (LPDWORD)palent));

        for(i=0;i<512;i++){
```

```
                if(recv(imExSock, (LPSTR)lineBuf, 512, 0) == SOCKET_ERROR){
                        AfxMessageBox("recv failed", MB_OK, 0);
                } else {
                        /* Load the scanLines */
                        /* Permit the horiz and vert shifting of result image */
                //      VHShift((int)i,lineBuf);
                                for(j=0;j<512;j++) scanLine[j] = (unsigned long)lineBuf[j];

                                imkPutScanLine((short)m_PicBuf1-
>GetNumProperty("FullPicture"),
                                        (int)i, (LPPE)scanLine);
                }
        }

        // Only write a subset of the image that contains real pixels
        m_PicBuf1->SetNumProperty("TopSelect", 6);
        m_PicBuf1->SetNumProperty("LeftSelect", 0);
        m_PicBuf1->SetNumProperty("HeightSelect", 480);
        m_PicBuf1->SetNumProperty("WidthSelect", 511);

        /* Write to file */
        // IMKSTORE) (LPSTR string, short hdib, int format, int compression);
        imkStore((LPSTR)"E:\\idsout.dib",
                (short)m_PicBuf1->GetNumProperty("Selection"),
                IMK_DIB, 0);


}

//////////////////////////////////////////////////////

void CIDSDlg::IdsGet()
// Move image from Host to IDS via the imExSock ethernet socket.
{
        long    scanLine[512];
        char    charScanLine[512]; /* Must convert long line to char line */
        int             lineNum=0,i;
        LONG  palette[256];

        sendIdsMsg("getfrm");

        /* Load the image */
        m_PicBuf1->SetNumProperty("FullPicture",
                imkLoad("E:\\GRAYIMG.DIB", 0, IMK_DIB));
        /* Get the palette */
        imkGetPalette((short)m_PicBuf1->GetNumProperty("FullPicture"),
                (LPPE)palette);

        /* Get and send each scan line */
        for(lineNum = 0; lineNum < 512; lineNum++) {
                imkGetScanLine((short)m_PicBuf1->GetNumProperty("FullPicture"),
                        lineNum, (LPPE)scanLine);
                /* Convert scanLine to char */
                for(i=0;i<512;i++){
                        charScanLine[i] = (char)palette[(int)scanLine[i]];
                }
```

```
                if(send(imExSock, (LPSTR)charScanLine, 512, 0) == SOCKET_ERROR){
#ifdef _DEBUG

                afxDump << "send failed \n";
#endif
                }

        }

}

//////////////////////////////////////////////////////////

void CIDSDlg::saveAsGrayDIBFromLive()
// In order to process single frames on the IDS, the image must be first
// captured as a DVA image then converted to a gray 512x512 DIB.
{
        WORD            result;
        char            *str;
  CString               CmdStr;
        int                     fileIOErr = 0;



        CmdStr = "FStore $VWin SAVE 3 ";
        CmdStr += '"';
        CmdStr += "E:\\TMPIMG.DVA";  // Have to use double \ because \ is an escape char
        CmdStr += '"';
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
  // Save image to temporary RAM disk file
        result = pBridgeView->micWrite((LPSTR)str );


        // Use the MIC Image Load to convert DVA to gray 512x512 DIB
        CmdStr = "Image Load ";
        CmdStr += '"';
        CmdStr += "E:\\TMPIMG.DVA"; // Have to use double \ because \ is an escape char
        CmdStr += '"';
        CmdStr += " DVA";
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        // Size the image to be 512 x 512 pixels needed for IDS
        CmdStr = "Image OSIZE 512 512";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        // Save as a gray DIB file
        CmdStr = "Image Save ";
        CmdStr += '"';
        CmdStr += "E:\\GRAYIMG.DIB";
        CmdStr += '"';
```

```
        CmdStr += " DIB GRAY";
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

//              AfxMessageBox("File Save Error", MB_OKIMB_ICONINFORMATION,0);

}

////////////////////////////////////////////////////////////

void CIDSDlg::pause(DWORD val)
// Pause val number of milliseconds
{
        DWORD       pauseUntil;

        pauseUntil = val + GetTickCount();

        while(pauseUntil >= GetTickCount());
}

////////////////////////////////////////////////////////////


void CIDSDlg::sendIdsMsg(char* msg)
// Send the message to the IDS computer via clientSock ethernet socket.
// The message must be copied to idsMessage since we have to send fixed
// length messages of MESSAGE_SIZE long.
{

        strcpy(idsMessage, msg);

                if(send(clientSock, idsMessage, MESSAGE_SIZE, 0) == SOCKET_ERROR){
        #ifdef _DEBUG

                        afxDump << "send failed \n";
        #endif
        }

}

////////////////////////////////////////////////////////////////////////
// CIDSDlg message handlers


BOOL CIDSDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Hide the Image Knife VBX, we'll use the main video window for results
        m_PicBuf1->SetNumProperty("Visible", FALSE);

        // Hide process percent bar
        m_processPercent->SetNumProperty("Visible", FALSE);

        return TRUE;  // return TRUE  unless you set the focus to a control
```

```
}

//////////////////////////////////////////////////////////

void CIDSDlg::OnIdsAdv(UINT, int, CWnd*, LPVOID)
{
        CIDSAdvDg               IDSAdvDg;


        IDSAdvDg.clientSock = clientSock;

        IDSAdvDg.imExSock = imExSock;

        IDSAdvDg.pIDSDlg = this;

        if (IDSAdvDg.DoModal() == IDOK)
        {
        }

}

//////////////////////////////////////////////////////////
void CIDSDlg::displayResult()
// Display the resulting IDS image.
// The image must be scaled to fit the display
{
        char    *str;
        int             result;
        // Since we can only directly display DVA images,
        // First load the DIB image and write it to RAM disk as DVA
        // then use FStore to display it.
        CString ImLdStr("Image Load ");
        ImLdStr += '"';
        ImLdStr += "E:\\IDSOUT.DIB";
        ImLdStr += '"';
        ImLdStr += " DIB";
        str = ImLdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        // Set the output size to  640x480 instead of 512x512
        CString ImOSizeStr("Image Osize 640 480");
        str = ImOSizeStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );
        // Save to RAM disk
        CString ImSvStr("Image Save ");
        ImSvStr += '"';
        ImSvStr += "E:\\IDSOUT.DVA";
        ImSvStr += '"';
        ImSvStr += " DVA NTSC FRAME";
        str = ImSvStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );
        // Finally load from RAM disk
        CString FSStr("FStore $VWin Load 3 ");
        FSStr += '"';
        FSStr += "E:\\IDSOUT.DVA";
        FSStr += '"';
```

```
        str = FSStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );
        //pBridgeView->MoveVideoWindow();


}

//////////////////////////////////////////////////////

void CIDSDlg::processImage()
// This function processes one frame through the IDS processor.
// The individual steps are:
//              End realtime IDS mode and freeze input
//              Size and save a gray scale image to RAM disk
//              Send the image to the IDS
//              Perform a oneshot operation while waiting 4 frame times on host
//              Send the result back to host
//              Restart realtime
//              Freeze display
//              Display result
{

        this->UpdateData(TRUE);

        // Show process percent bar
        m_processPercent->SetNumProperty("Visible", TRUE);

        m_processPercent->SetNumProperty("FloodPercent", 0);

  sendIdsMsg("endreal"); // End realtime mode
  sendIdsMsg("init");   // Initialize
  SetHisto(Contrast);
  m_processPercent->SetNumProperty("FloodPercent", 10);
  pause((DWORD) 30 * 2);      // Pause 2 frames
  sendIdsMsg("freeze"); // Freeze input
  pBridgeView->freezeVideo(TRUE); // Freeze display
  m_processPercent->SetNumProperty("FloodPercent", 20);

  if(m_UseSavedImage)
              saveAsGrayDIBFromSaved();
  else
        saveAsGrayDIBFromLive(); // Size and save gray scale image
  m_processPercent->SetNumProperty("FloodPercent", 40);
        IdsGet(); // Moves image from Host to IDS
        m_processPercent->SetNumProperty("FloodPercent", 50);
  pause((DWORD) 3000);      // Pause 3 sec
        sendIdsMsg("oneshot");          // Perform One Shot

        pause((DWORD) 30 * 10);       // Pause 4 frames
        m_processPercent->SetNumProperty("FloodPercent", 60);

        IdsSend(); // Moves image from IDS to Host
        m_processPercent->SetNumProperty("FloodPercent", 70);

        sendIdsMsg("init");   // Initialize
  SetHisto(Contrast);
```

```
        sendIdsMsg("realtime"); // Start realtime
    m_processPercent->SetNumProperty("FloodPercent", 80);

        displayResult(); // Display result
        m_processPercent->SetNumProperty("FloodPercent", 100);

        // Hide process percent bar
        m_processPercent->SetNumProperty("Visible", FALSE);

}

//////////////////////////////////////////////////////////
void CIDSDlg::saveAsGrayDIBFromSaved()
{
#define         FILEDIAL        201

        WORD            result;
        char            *str;
        HCURSOR                 hCursorOld;
        CString         CmdStr;

        CVBControl      fileDialog;
        int                     fileIOErr = 0;

        // Set DPOS
        result = pBridgeView->micWrite((LPSTR)"FStore $VWin DPOS 3 0 0 ABSOLUTE");

        // Set DSIZE
        result = pBridgeView->micWrite((LPSTR)"FStore $VWin DSIZE 3 FULL");

        // Create the VBX file dialog
        CRect  dRect(0,0,100,100);
        result = (WORD)fileDialog.Create((LPCSTR)"CMDIALOG.VBX;CommonDialog;Save
Image",
                // The name CommonDialog MUST be used or it won't create
                WS_CHILD, dRect, this, FILEDIAL);
                // The dRect is a dummy since the CommonDialog VBX determines the rect.
        if (result){
                fileDialog.SetStrProperty("DefaultExt", "DVA");
                fileDialog.SetStrProperty("DialogTitle", "Display Image");
                fileDialog.SetStrProperty("Filter",
                        "VideoLogic DVA (*.DVA)|*.DVA|Windows DIB (*.DIB)|*.DIB|All Files
(*.*)|*.*");
                fileDialog.SetNumProperty("FilterIndex", 1);
                fileDialog.SetNumProperty("Flags", OFN_OVERWRITEPROMPT);
                fileDialog.SetStrProperty("InitDir","C:\\Bridge"); // Have to use double \ because \
is an escape char
                fileDialog.SetNumProperty("CancelError",TRUE);

                // At last lets open the dialog
                fileDialog.SetNumProperty("Action", 1);

                if(fileDialog.m_nError == 0){
                        // Set the cursor to an hourglass
                        hCursorOld = SetCursor(LoadCursor(NULL, IDC_WAIT));
```

```
        // Get fileName and extension
        CString fileName(fileDialog.GetStrProperty("FileName"));
        CString ext(fileName.Right(3));

    // Switch based on fileName extension
    if(_strnicmp(ext.GetBuffer(4), "dva", 3) == 0){
            // DVA type. Use the MIC FStore Load
            CmdStr = "FStore $VWin Load 3 ";
            CmdStr += '"';
            CmdStr += fileName;
            CmdStr += '"';
            CmdStr.MakeUpper(); // File names have to be upper case
            str = CmdStr.GetBuffer(256);
            result = pBridgeView->micWrite((LPSTR)str );
            // Use the MIC Image Load to convert DVA to gray 512x512 DIB
            CmdStr = "Image Load ";
            CmdStr += '"';
            CmdStr += fileName;
            CmdStr += '"';
            CmdStr += " DVA";
            CmdStr.MakeUpper(); // File names have to be upper case
            str = CmdStr.GetBuffer(256);
            result = pBridgeView->micWrite((LPSTR)str );
        }else if(_strnicmp(ext.GetBuffer(4), "dib", 3) == 0){
            // Since we can only directly display DVA images,
            // First load the DIB image and write it to RAM disk as DVA
            // then use FStore to display it.
            CString ImLdStr("Image Load ");
            ImLdStr += '"';
            ImLdStr += fileName;
            ImLdStr += '"';
            ImLdStr += " DIB";
            ImLdStr.MakeUpper(); // File names have to be upper case
            str = ImLdStr.GetBuffer(256);
            result = pBridgeView->micWrite((LPSTR)str );
            // Save to RAM disk
            CString ImSvStr("Image Save ");
            ImSvStr += '"';
            ImSvStr += "E:\\tmpib2va.dva";
            ImSvStr += '"';
            ImSvStr += " DVA NTSC FRAME";
            ImSvStr.MakeUpper(); // File names have to be upper case
            str = ImSvStr.GetBuffer(256);
            result = pBridgeView->micWrite((LPSTR)str );
            // Finally load from RAM disk
            CString FSStr("FStore $VWin Load 3 ");
            FSStr += '"';
            FSStr += "E:\\tmpib2va.dva";
            FSStr += '"';
            FSStr.MakeUpper(); // File names have to be upper case
            str = FSStr.GetBuffer(256);
            result = pBridgeView->micWrite((LPSTR)str );

            // Use the MIC Image Load to convert DVA to gray 512x512 DIB
            CmdStr = "Image Load ";
            CmdStr += '"';
```

```
                    CmdStr += "E:\\tmpib2va.dva";
                    CmdStr += '"';
                    CmdStr += " DVA";
                    CmdStr.MakeUpper(); // File names have to be upper case
                    str = CmdStr.GetBuffer(256);
                    result = pBridgeView->micWrite((LPSTR)str );

                }else
                    AfxMessageBox("Wrong Image Type. Should be DVA or DIB.",
MB_OKIMB_ICONINFORMATION,0);

                // Now save the image as a gray dib
                // Size the image to be 512 x 512 pixels needed for IDS
                CmdStr = "Image OSIZE 512 512";
                CmdStr.MakeUpper();
                str = CmdStr.GetBuffer(256);
                result = pBridgeView->micWrite((LPSTR)str );

                // Save as a gray DIB file
                CmdStr = "Image Save ";
                CmdStr += '"';
                CmdStr += "E:\\GRAYIMG.DIB";
                CmdStr += '"';
                CmdStr += " DIB GRAY";
                CmdStr.MakeUpper(); // File names have to be upper case
                str = CmdStr.GetBuffer(256);
                result = pBridgeView->micWrite((LPSTR)str );

                // Restore the old cursor
                SetCursor(hCursorOld);

            } else
                AfxMessageBox("Canceled display image.",
MB_OKIMB_ICONINFORMATION,0);
        } else
            AfxMessageBox("Image Display Error", MB_OKIMB_ICONINFORMATION,0);


}

//////////////////////////////////////////////////////////////
void CIDSDlg::OnProcSavImg(UINT, int, CWnd*, LPVOID)
{
        m_UseSavedImage = TRUE;
        processImage();

}

//////////////////////////////////////////////////////////////
void CIDSDlg::OnProcImg(UINT, int, CWnd*, LPVOID)
{
        m_UseSavedImage = FALSE;
```

```
        processImage();
    }
```

```
// idsdlg.h : header file
//

#include        "winsock.h"

// Forward class declaration
class   CBridgeView;

/////////////////////////////////////////////////////////////////////////
// Define Message size and status codes
#define         MESSAGE_SIZE        50
#define         RAW                         100
#define         INITIAL             101
#define         DECOMPRESSING                       102
#define         DONE_DECOMP                         103
#define         LOADING                     104
#define         READY               105
#define         RUNNING                     106
#define         SENDING                     107
#define         GETTING                     108


/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////
// CIDSDlg dialog

class CIDSDlg : public CDialog
{
// Construction
public:
        CIDSDlg(CWnd* pParent = NULL);      // standard constructor

// Dialog Data
        //{{AFX_DATA(CIDSDlg)
        enum { IDD = IDD_IDS_DLG };
        CVBControl*     m_processPercent;
        CVBControl*     m_PicBuf1;
        int             m_hShift;
        int             m_vShift;
        //}}AFX_DATA

        SOCKET              clientSock;
        SOCKET              imExSock;

        //static UINT           m_Contrast; // Edge Contrast

        // The ids control messages will be MESSAGE_SIZE long
        char            idsMessage[MESSAGE_SIZE];
        void            IdsGet(void);
        void            IdsSend(void);

        // MIC Support
        void            SetViewPtr(CBridgeView* p);
        CBridgeView*    pBridgeView;
```

```
        BOOL        m_UseSavedImage;
        void        saveAsGrayDIBFromSaved(void);
        void        saveAsGrayDIBFromLive(void);
        void        processImage(void);
        void        displayResult(void);

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CIDSDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnIdsAdv(UINT, int, CWnd*, LPVOID);
        afx_msg void OnProcImg(UINT, int, CWnd*, LPVOID);
        afx_msg void OnProcSavImg(UINT, int, CWnd*, LPVOID);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()


        // Utility functions
        void        ZeroMsgBuf(void);
        void        pause(DWORD val);
        void        sendIdsMsg(char* msg);
        void        VHShift(int i,char* lineBuf);
public:
        void        SetHisto(UINT optionNum);
};
```

```
// improcdl.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "improcdl.h"

#include "bridgdoc.h"
#include "bridgvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CImProcDlg dialog

CImProcDlg::CImProcDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CImProcDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CImProcDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}

void CImProcDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CImProcDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CImProcDlg, CDialog)
        //{{AFX_MSG_MAP(CImProcDlg)
        ON_VBXEVENT(VBN_CLICK, IDC_UpLeft, OnUpLeft)
        ON_VBXEVENT(VBN_CLICK, IDC_LowerLeft, OnLowerLeft)
        ON_VBXEVENT(VBN_CLICK, IDC_LowerRight, OnLowerRight)
        ON_VBXEVENT(VBN_CLICK, IDC_UpRight, OnUpRight)
        ON_VBXEVENT(VBN_CLICK, IDC_Center, OnCenter)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Support functions
void CImProcDlg::SetViewPtr(CBridgeView* p)
{

        pBridgeView = p;

}


/////////////////////////////////////////////////////////////////////////////
void CImProcDlg::SaveAndLoad()
```

```
// Save the current image to RAM disk and Load it for conversion
{
        WORD            result;
        char            *str;
   CString              CmdStr;

        CmdStr = "FStore $VWin SAVE 3 ";
        CmdStr += '"';
        CmdStr += "E:\\TMPIMG.DVA";  // Have to use double \ because \ is an escape char
        CmdStr += '"';
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
   // Save image to temporary RAM disk file
        result = pBridgeView->micWrite((LPSTR)str );


        // Use the MIC Image Load
        CmdStr = "Image Load ";
        CmdStr += '"';
        CmdStr += "E:\\TMPIMG.DVA"; // Have to use double \ because \ is an escape char
        CmdStr += '"';
        CmdStr += " DVA";
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

}

//////////////////////////////////////////////////////////////////////
void CImProcDlg::SaveAndDisplay()
// Save the image that has been zoomed and display it
{
        WORD            result;
        char            *str;
   CString              CmdStr;

        // Size the output image to be full screen
        CmdStr = "Image OSIZE FULL";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        // Save as DVA file
        CmdStr = "Image Save ";
        CmdStr += '"';
        CmdStr += "E:\\TMPZOOM.DVA";
        CmdStr += '"';
        CmdStr += " DVA NTSC FRAME";
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        // Display the zoomed image
        CmdStr = "FStore $VWin LOAD 3 ";
        CmdStr += '"';
        CmdStr += "E:\\TMPZOOM.DVA";  // Have to use double \ because \ is an escape char
```

```
        CmdStr += '"';
        CmdStr.MakeUpper(); // File names have to be upper case
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

}


////////////////////////////////////////////////////////////////////
// CImProcDlg message handlers

void CImProcDlg::OnUpLeft(UINT, int, CWnd*, LPVOID)
{
// In order to zoom, the image must be saved with a smaller Iregion and
// a full size Osize

        WORD            result;
        char            *str;
    CString             CmdStr;
        int                     fileIOErr = 0;

        // Set the cursor to an hourglass
        BeginWaitCursor();

        pBridgeView->freezeVideo(TRUE);  // Freeze display

        SaveAndLoad();

        // Size the input image to be upper left
        CmdStr = "Image Iregion 0 0 320 240";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        SaveAndDisplay();

        // Restore old cursor
        EndWaitCursor();

//              AfxMessageBox("File Save Error", MB_OKIMB_ICONINFORMATION,0);

}


void CImProcDlg::OnLowerLeft(UINT, int, CWnd*, LPVOID)
{
        WORD            result;
        char            *str;
    CString             CmdStr;
        int                     fileIOErr = 0;

        // Set the cursor to an hourglass
        BeginWaitCursor();

        pBridgeView->freezeVideo(TRUE);  // Freeze display
```

```
        SaveAndLoad();

        // Size the input image to be lower left
        CmdStr = "Image Iregion 0 240 320 479";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        SaveAndDisplay();

        // Restore old cursor
        EndWaitCursor();

}

void CImProcDlg::OnLowerRight(UINT, int, CWnd*, LPVOID)
{
        WORD            result;
        char            *str;
    CString             CmdStr;
        int                     fileIOErr = 0;

        // Set the cursor to an hourglass
        BeginWaitCursor();

        pBridgeView->freezeVideo(TRUE);  // Freeze display

        SaveAndLoad();

        // Size the input image to be lower right
        CmdStr = "Image Iregion 320 240 639 479";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        SaveAndDisplay();

        // Restore old cursor
        EndWaitCursor();

}

void CImProcDlg::OnUpRight(UINT, int, CWnd*, LPVOID)
{
        WORD            result;
        char            *str;
    CString             CmdStr;
        int                     fileIOErr = 0;

        // Set the cursor to an hourglass
        BeginWaitCursor();

        pBridgeView->freezeVideo(TRUE);  // Freeze display

        SaveAndLoad();
```

```
        // Size the input image to be upper right
        CmdStr = "Image Iregion 320 0 639 240";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        SaveAndDisplay();

        // Restore old cursor
        EndWaitCursor();

}

void CImProcDlg::OnCenter(UINT, int, CWnd*, LPVOID)
{
        WORD            result;
        char            *str;
    CString             CmdStr;
        int                     fileIOErr = 0;

        // Set the cursor to an hourglass
        BeginWaitCursor();

        pBridgeView->freezeVideo(TRUE);  // Freeze display

        SaveAndLoad();

        // Size the input image to be center
        CmdStr = "Image Iregion 160 120 480 360";
        CmdStr.MakeUpper();
        str = CmdStr.GetBuffer(256);
        result = pBridgeView->micWrite((LPSTR)str );

        SaveAndDisplay();

        // Restore old cursor
        EndWaitCursor();

}
```

```
// improcdl.h : header file
//

class    CBridgeView; // Forward Declaration

/////////////////////////////////////////////////////////////////////////////
// CImProcDlg dialog

class CImProcDlg : public CDialog
{
// Construction
public:
        CImProcDlg(CWnd* pParent = NULL);   // standard constructor

        // MIC SUPPORT
        void            SetViewPtr(CBridgeView* p);
        CBridgeView*    pBridgeView;
        void                    SaveAndDisplay();
        void                    SaveAndLoad();

// Dialog Data
        //{{AFX_DATA(CImProcDlg)
        enum { IDD = IDD_ImageProc };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CImProcDlg)
        afx_msg void OnUpLeft(UINT, int, CWnd*, LPVOID);
        afx_msg void OnLowerLeft(UINT, int, CWnd*, LPVOID);
        afx_msg void OnLowerRight(UINT, int, CWnd*, LPVOID);
        afx_msg void OnUpRight(UINT, int, CWnd*, LPVOID);
        afx_msg void OnCenter(UINT, int, CWnd*, LPVOID);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "bridge.h"

#include "signal.h"
#include "dos.h"
#include "mainfrm.h"

#include "knifevbx.h"


#include "bridgdoc.h"
#include "bridgvw.h"

#include "about.h"


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_WM_MOVE()
        ON_COMMAND(ID_EXIT, OnExit)
        ON_WM_DESTROY()
        ON_WM_TIMER()
        ON_WM_SYSCOMMAND()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_VideoSelect,
        ID_SNAP_SHOT,
                ID_SEPARATOR,
        ID_IDS,
        ID_VISCAActive,
        ID_CAMERAActive,
        ID_IMAGE_PROC,
                ID_SEPARATOR,
        ID_USER_OPTIONS,
```

```
        /*ID_USER_LEVEL, */
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_HELP_INDEX,
        //ID_USER_HELP,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
                ID_SEPARATOR,
        ID_EXIT,
        /* ID_CONTEXT_HELP, */
};

static UINT BASED_CODE indicators[] =
{
        ID_SEPARATOR,                   // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        SIZE    butsiz, imsiz;
        CMenu*          sysMenu;

        butsiz.cx = 38;
        butsiz.cy = 54;
        imsiz.cx = 32;
        imsiz.cy = 48;

        // Append about to system menu
        sysMenu = GetSystemMenu(FALSE);
        sysMenu->AppendMenu(MF_SEPARATOR,0,(LPSTR)NULL);
        sysMenu->AppendMenu(MF_STRING,ID_ABOUT,"About...");


  // Create a serial port control to interface with pan & tilt

        CRect           portRect(5,5,50,50);
        BOOL            result;
        int                     pO;
```

```
        result = platformPort.Create("MSCOMM.VBX;MSCOMM;Port",
                NULL,portRect, this,101,NULL, FALSE);

        if(result){  // Init the port

        platformPort.SetNumProperty("CommPort", 2); // Comm2
        // A lower baud rate results in smoother motion.
        platformPort.SetStrProperty("Settings", "1200, N, 8, 1");
        // Open the  port
        platformPort.SetNumProperty("PortOpen", TRUE);

        pO = (int)platformPort.GetNumProperty("PortOpen");

        // Send Monitor and Camera select code
        // NOTE: THE MONITOR SELECT SWITCHES MUST BE ALL OFF ON THE 1641
CONTROLLER
        // Send the string
        platformPort.SetStrProperty("Output", "1Ma1#a");
        // Clear the return code from the input buffer
        platformPort.SetNumProperty("InBufferCount", 0);
        // Set a timer to read the joystick position and move the platform.

        SetTimer(CHECK_JOY, 250, NULL);
        }else
                AfxMessageBox((LPCSTR) "Could not create comm port.",
                        MB_OK|MB_ICONINFORMATION,0);

        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        // Size the toolbar buttons
        m_wndToolBar.SetSizes(butsiz, imsiz);

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE("Failed to create toolbar\n");
                return -1;                      // fail to create
        }

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE("Failed to create status bar\n");
                return -1;                      // fail to create
        }

        // First maximize the size of the window so we can get the client size
        SetWindowPos(&wndTop, 0,0,640,480,0);
        CRect  client;
        GetClientRect(&client);
        float adj = (5.0/4.0)*((float)client.bottom/(float)client.right);
        // Now set the size to keep a 5/4 aspect ratio for the client area
```

```
        SetWindowPos(&wndTop,0,0,(int) 640.0*adj, 480,0);

        // Hide the menu since its functions are performed by control bar
        SetMenu(NULL);

        return 0;
}


/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// The length of the command strings is proportional to how often you
// send the string and the baud rate you send it at.
// For smooth operation you want to keep the output continuous.
void CMainFrame::TiltDown()
{
        // Send the string
        platformPort.SetStrProperty("Output",
        "DaDaDaDaDaDaDaDaDaDaDaDaDaDaDa");
        // Clear the return code from the input buffer
        platformPort.SetNumProperty("InBufferCount", 0);

}
void CMainFrame::TiltUp()
{
        // Send the string
        platformPort.SetStrProperty("Output",
        "UaUaUaUaUaUaUaUaUaUaUaUaUaUaUa");
        // Clear the return code from the input buffer
        platformPort.SetNumProperty("InBufferCount", 0);

}

void CMainFrame::PanRight()
{
        // Send the string
        platformPort.SetStrProperty("Output",
        "RaRaRaRaRaRaRaRaRaRaRaRaRaRaRa");
        // Clear the return code from the input buffer
        platformPort.SetNumProperty("InBufferCount", 0);

}
```

```
void CMainFrame::PanLeft()
{
        // Send the string
        platformPort.SetStrProperty("Output",
        "LaLaLaLaLaLaLaLaLaLaLaLaLaLaLa");
        // Clear the return code from the input buffer
        platformPort.SetNumProperty("InBufferCount", 0);

}


unsigned int    CMainFrame::ReadTilt()
{

        union           _REGS inregs, outregs;

        // Try to read the joystick A(x) and A(y) values
        // using BIOS interrupt 0x15

        inregs.h.ah = 0x84;  // read joystick command
        inregs.x.dx = 0x01;  // read position

        // Execute the interrupt
        _int86(0x15, &inregs, &outregs);

        return(outregs.x.bx);


}


unsigned int CMainFrame::ReadPan()
{

        union           _REGS inregs, outregs;

        // Try to read the joystick A(x) and A(y) values
        // using BIOS interrupt 0x15

        inregs.h.ah = 0x84;  // read joystick command
        inregs.x.dx = 0x01;  // read position

        // Execute the interrupt
        _int86(0x15, &inregs, &outregs);

        return(outregs.x.ax);


}

BOOL  CMainFrame::ReadButton1()
{
        union           _REGS inregs, outregs;

// Try to read the joystick button values
```

```
        inregs.h.ah = 0x84;  // read joystick command
        inregs.x.dx = 0x00;  // read buttons

        // Execute the interrupt
        _int86(0x15, &inregs, &outregs);



        // Return results
        if(outregs.h.al & 0x10)
                return(FALSE);  // Button is UP
        else
                return(TRUE);   // Button is pressed

#ifdef BUTTON2
        if(outregs.h.al & 0x20)
                stickVal += " Button 2 Up ";
        else
                stickVal += " Button 2 Down ";
#endif
}

void CMainFrame::KillJoy()
{
   KillTimer(CHECK_JOY);
}

void CMainFrame::StartJoy()
{
  SetTimer(CHECK_JOY, 250, NULL);
}



/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnTimer(UINT nIDEvent)
{
        CWnd*           pCV;

        if(nIDEvent == CHECK_JOY){
                if(ReadPan() > 120) PanLeft();
                if(ReadPan() < 60) PanRight();
                if(ReadTilt() > 120) TiltDown();
                if(ReadTilt() < 60) TiltUp();

                // Send snapshot message if Button 1 down
                if(ReadButton1()){
                        pCV = FindWindow((LPCSTR)"CBridgeView", NULL); //
WM_COMMAND
                        this->PostMessage((UINT) WM_COMMAND, ID_SNAP_SHOT , 0);
                }
        }

        CFrameWnd::OnTimer(nIDEvent);
}
```

```
void CMainFrame::OnDestroy()
{
        CFrameWnd::OnDestroy();

        platformPort.SetNumProperty("PortOpen", FALSE);
        KillTimer(CHECK_JOY);

        imkEnd();       // Remove the KNIFE.VBX library

}
////////////////////////////
// OnMove
// Any time the main frame is moved, then send invalidate the
// view so that the view will be repainted and the video window
// will be moved.
void CMainFrame::OnMove(int x, int y)
{
        CView* pV;

        CFrameWnd::OnMove(x, y);

        if(pV = GetActiveView())
                pV->Invalidate(TRUE);

}

void CMainFrame::OnExit()
{
        if(MessageBox((LPCSTR)"Are you sure you want to exit?",
                                        (LPCSTR)"Are you sure?",
                                        MB_ICONQUESTION | MB_OKCANCEL) == IDOK)
                DestroyWindow();

}

////////////////////////////////////////////////////////


void CMainFrame::OnSysCommand(UINT nID, LPARAM lParam)
{
        if(nID == ID_ABOUT){
                CAbout AboutDlg;

                AboutDlg.DoModal();
        }else

                CFrameWnd::OnSysCommand(nID, lParam);
}
```

```
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
        CVBControl      platformPort;

// Operations
public:
        void    TiltUp();
        void    TiltDown();
        void    PanRight();
        void    PanLeft();
        unsigned int            ReadTilt();
        unsigned int            ReadPan();
        BOOL    ReadButton1();
        void    KillJoy();
        void    StartJoy();

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual  void AssertValid() const;
        virtual  void Dump(CDumpContext& dc) const;
#endif


protected:      // control bar embedded members
        CStatusBar      m_wndStatusBar;
        CToolBar        m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnMove(int x, int y);
        afx_msg void OnExit();
        afx_msg void OnDestroy();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
/** mscomm.h ***************************************
*       This is the header file for the MSComm VBX control
*
*
*****************************************************/
#include "constant.h"  // VBX constants file

#ifdef __cplusplus
extern "C" {
#endif  /* __cplusplus */
int FAR PASCAL ComOutput(HWND hWnd, LPSTR lpData, int cbData);
int FAR PASCAL ComInput(HWND hWnd, LPSTR lpData, int cbData);
#ifdef __cplusplus
}
#endif
```

```
// usroptdg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "usroptdg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CUsrOptDg dialog

CUsrOptDg::CUsrOptDg(CWnd* pParent /*=NULL*/)
        : CDialog(CUsrOptDg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CUsrOptDg)
        m_OptionRD = NULL;
        m_OptionHD = NULL;
        m_OptionFD = NULL;
        m_seqSpin = NULL;
        m_ImName = NULL;
        m_ImNamePrompt = NULL;
        m_Im_Format_Btn = NULL;
        m_autoNameBtn = NULL;
        //}}AFX_DATA_INIT
}


void CUsrOptDg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CUsrOptDg)
        DDX_VBControl(pDX, IDC_OPTIONRD, m_OptionRD);
        DDX_VBControl(pDX, IDC_OPTIONHD, m_OptionHD);
        DDX_VBControl(pDX, IDC_OPTIONFD, m_OptionFD);
        DDX_Control(pDX, IDC_ImSequence, m_sequenceEdit);
        DDX_VBControl(pDX, IDC_SequenceSpin, m_seqSpin);
        DDX_Control(pDX, IDC_ImBaseName, m_ImBaseName);
        DDX_VBControl(pDX, IDC_ImName, m_ImName);
        DDX_VBControl(pDX, IDC_NamePrompt, m_ImNamePrompt);
        DDX_VBControl(pDX, IDC_IMAGE_FORMAT, m_Im_Format_Btn);
        DDX_VBControl(pDX, IDC_AUTO_NAME, m_autoNameBtn);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CUsrOptDg, CDialog)
        //{{AFX_MSG_MAP(CUsrOptDg)
        ON_VBXEVENT(VBN_CLICK, IDC_AUTO_NAME, OnAutoName)
        ON_VBXEVENT(VBN_CLICK, IDC_IMAGE_FORMAT, OnImageFormat)
        ON_EN_UPDATE(IDC_ImBaseName, OnUpdateImBaseName)
        ON_EN_UPDATE(IDC_ImSequence. OnUpdateImSequence)
        ON_VBXEVENT(VBN_SPINDOWN, IDC_SequenceSpin, OnSequenceSpinDown)
        ON_VBXEVENT(VBN_SPINUP, IDC_SequenceSpin, OnSequenceSpinUp)
```

```
        ON_VBXEVENT(VBN_CLICK, IDC_OPTIONFD, OnOptionfd)
        ON_VBXEVENT(VBN_CLICK, IDC_OPTIONHD, OnOptionhd)
        ON_VBXEVENT(VBN_CLICK, IDC_OPTIONRD, OnOptionrd)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


//////////////////////////////////////////////////////////////////////
// CUsrOptDg message handlers

void CUsrOptDg::OnAutoName(UINT, int, CWnd*, LPVOID lpParams)
{  char *baseStr;
        char    seqStr[4];

        //if(AFX_NUM_EVENTPARAM(short, lpParams)){
        if(m_autoNameBtn->GetNumProperty("Value")){
                m_ImBaseName.ShowWindow(SW_SHOW);
                baseStr = baseName.GetBuffer(7);
                m_ImBaseName.SetWindowText((LPSTR)baseStr);
                m_ImNamePrompt->SetNumProperty("Visible", TRUE);
                itoa(sequenceNum, seqStr, 10);
                m_sequenceEdit.ShowWindow(SW_SHOW);
                m_sequenceEdit.SetWindowText((LPSTR)seqStr);
                m_seqSpin->SetNumProperty("Visible", TRUE);
                autoNameButton = TRUE;
        }else{
                m_ImBaseName.ShowWindow(SW_HIDE);
                m_ImNamePrompt->SetNumProperty("Visible", FALSE);
                m_sequenceEdit.ShowWindow(SW_HIDE);
                m_seqSpin->SetNumProperty("Visible", FALSE);
                autoNameButton = FALSE;
        }

        OnUpdateImBaseName();

}

//////////////////////////////////////////////////////////////////////

void CUsrOptDg::OnImageFormat(UINT, int, CWnd*, LPVOID)
{
        if(m_Im_Format_Btn->GetNumProperty("Value"))
                DIBButton = TRUE;
        else
                DIBButton = FALSE;
        OnUpdateImBaseName();

}


//////////////////////////////////////////////////////////////////////

BOOL CUsrOptDg::OnInitDialog()
{
        CDialog::OnInitDialog();
```

```
        m_ImBaseName.LimitText(5); // Limit base names to 5 char
        m_sequenceEdit.LimitText(3); // Limit sequence num to 3 char

        if(autoNameButton){
                m_autoNameBtn->SetNumProperty("Value", TRUE);
        }else{
                m_autoNameBtn->SetNumProperty("Value", FALSE);
        }

        if(DIBButton)  m_Im_Format_Btn->SetNumProperty("Value", TRUE);
        else   m_Im_Format_Btn->SetNumProperty("Value", FALSE);

        // Turn selected one on (others automatically turn off)
        switch(destination){
                case HardDisk:
                        m_OptionHD->SetNumProperty("Value", TRUE);
                        break;
                case RemovableDisk:
                        m_OptionRD->SetNumProperty("Value", TRUE);
                        break;
                case FloppyDisk:
                        m_OptionFD->SetNumProperty("Value", TRUE);
                        break;
        }

        return TRUE;  // return TRUE  unless you set the focus to a control
}


//////////////////////////////////////////////////////////

void CUsrOptDg::OnUpdateImBaseName()
{
        CString fullName;
        char    *fullString;
        char    baseStr[7];
        char    seqStr[10];

        m_ImBaseName.GetWindowText((LPSTR)baseStr,6);
        switch(destination){
                case HardDisk:
                        fullName = HDDIR;
                        break;
                case RemovableDisk:
                        fullName = RDDIR;
                        break;
                case FloppyDisk:
                        fullName = FDDIR;
                        break;
        }

        if(autoNameButton){
                fullName += baseStr;
                fullName += itoa(sequenceNum,seqStr,10);
        }else{
```

```
            fullName += "*";
        }

        if(DIBButton)
            fullName += ".DIB";
        else
            fullName += ".DVA";
        fullName.MakeUpper();
        fullString = fullName.GetBuffer(50);
        m_ImName->SetStrProperty("Caption", fullString);
        baseName = baseStr;

}

//////////////////////////////////////////////////////////

void CUsrOptDg::OnUpdateImSequence()
{
        char    seqStr[4];

        m_sequenceEdit.GetWindowText((LPSTR)seqStr,4);
        sequenceNum = (unsigned int) atoi(seqStr);
        OnUpdateImBaseName();

}

void CUsrOptDg::OnSequenceSpinDown(UINT, int, CWnd*, LPVOID)
{
        char    seqStr[4];

        if(sequenceNum > 0){
            sequenceNum--;
            itoa(sequenceNum, seqStr, 10);
            m_sequenceEdit.SetWindowText((LPSTR)seqStr);
            OnUpdateImBaseName();
        }
}

void CUsrOptDg::OnSequenceSpinUp(UINT, int, CWnd*, LPVOID)
{
        char    seqStr[4];

        if(sequenceNum < 999){
            sequenceNum++;
            itoa(sequenceNum, seqStr, 10);
            m_sequenceEdit.SetWindowText((LPSTR)seqStr);
            OnUpdateImBaseName();
        }
}

void CUsrOptDg::OnOptionfd(UINT, int, CWnd*, LPVOID)
{
        if(m_OptionFD->GetNumProperty("Value")){
            destination = FloppyDisk;
            OnUpdateImBaseName();
        }
```

```
}

void CUsrOptDg::OnOptionhd(UINT, int, CWnd*, LPVOID)
{
        if(m_OptionHD->GetNumProperty("Value")){
                destination = HardDisk;
                OnUpdateImBaseName();
        }

}

void CUsrOptDg::OnOptionrd(UINT, int, CWnd*, LPVOID)
{
        if(m_OptionRD->GetNumProperty("Value")){
                destination = RemovableDisk;
                OnUpdateImBaseName();
        }

}
```

```
// usroptdg.h : header file
//
#define         HDDIR "C:\\BRIDGE\\"
#define         RDDIR "D:\\"
#define         FDDIR "A:\\"
#define         HardDisk                0
#define         RemovableDisk 1
#define         FloppyDisk              2
/////////////////////////////////////////////////////////////////////////////
// CUsrOptDg dialog

class CUsrOptDg : public CDialog
{
// Construction
public:
        CUsrOptDg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
        //{{AFX_DATA(CUsrOptDg)
        enum { IDD = IDD_USER_OPTIONS };
        CVBControl*     m_OptionRD;
        CVBControl*     m_OptionHD;
        CVBControl*     m_OptionFD;
        CEdit   m_sequenceEdit;
        CVBControl*     m_seqSpin;
        CEdit   m_ImBaseName;
        CVBControl*     m_ImName;
        CVBControl*     m_ImNamePrompt;
        CVBControl*     m_Im_Format_Btn;
        CVBControl*     m_autoNameBtn;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CUsrOptDg)
        afx_msg void OnAutoName(UINT, int, CWnd*, LPVOID);
        afx_msg void OnImageFormat(UINT, int, CWnd*, LPVOID);
        virtual BOOL OnInitDialog();
        afx_msg void OnUpdateImBaseName();
        afx_msg void OnUpdateImSequence();
        afx_msg void OnSequenceSpinDown(UINT, int, CWnd*, LPVOID);
        afx_msg void OnSequenceSpinUp(UINT, int, CWnd*, LPVOID);
        afx_msg void OnOptionfd(UINT, int, CWnd*, LPVOID);
        afx_msg void OnOptionhd(UINT, int, CWnd*, LPVOID);
        afx_msg void OnOptionrd(UINT, int, CWnd*, LPVOID);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

public:
        // Save button states for the next display
        BOOL  DIBButton;
        BOOL  autoNameButton;
        CString baseName;
```

```
    unsigned int    sequenceNum;
    unsigned int    destination;

};
```

```
// vidsdlg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "vidsdlg.h"

#include "bridgdoc.h"

#include "bridgvw.h"

#include "vscadlg.h"

#include "viscadev.h"

#include "usroptdg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

static UINT NEAR WM_VISCAINQ = RegisterWindowMessage((LPCSTR)"VISCA_INQ");
static UINT NEAR WM_VISCAERR = RegisterWindowMessage((LPCSTR)"VISCA_ERR");

/////////////////////////////////////////////////////////////////////////////
// CVidSDlg dialog

CVidSDlg::CVidSDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CVidSDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CVidSDlg)
        m_SelIDSButton = NULL;
        m_SelCamButton = NULL;
        m_FreezeButton = NULL;
        //}}AFX_DATA_INIT
}

void CVidSDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CVidSDlg)
        DDX_VBControl(pDX, IDC_SelectIDS, m_SelIDSButton);
        DDX_VBControl(pDX, IDC_SelectCamera, m_SelCamButton);
        DDX_VBControl(pDX, IDC_CHECKFreeze, m_FreezeButton);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CVidSDlg, CDialog)
        //{{AFX_MSG_MAP(CVidSDlg)
        ON_VBXEVENT(VBN_CLICK, IDC_SelectCamera, OnClickSelectCamera)
        ON_VBXEVENT(VBN_CLICK, IDC_SelectIDS, OnClickSelectIDS)
        ON_VBXEVENT(VBN_CLICK, IDC_CHECKFreeze, OnClickFreeze)
        ON_VBXEVENT(VBN_CLICK, IDC_DisplaySaved, OnDisplaySaved)
        ON_REGISTERED_MESSAGE(WM_VISCAINQ, OnViscaInq)
        ON_REGISTERED_MESSAGE(WM_VISCAERR, OnViscaErr)
```

```
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////
// Helper functions

void CVidSDlg::SetViewPtr(CBridgeView* p)
{

        pBridgeView = p;

}

void CVidSDlg::SetVSCADlgPtr(CVSCAdlg* p)
{
        pVSCAdlg = p;
}


////////////////////////////
// OnInitDialog()
// When the dialog is initialized, inquire about the current state
// of the DVA and the tape player and set the switch states accordingly.
BOOL CVidSDlg::OnInitDialog()
{
#define BUFSIZE         256
        char    szMICRspBuf[BUFSIZE], *token;
        WORD enabled;           // 1=Enabled, 0=Disabled

        // Call Base class OnInitDialog
        CDialog::OnInitDialog();

        // Get/Set freezebutton state ;

        pBridgeView->micInq((LPSTR)"? FStore $VWin Input", szMICRspBuf, BUFSIZE);
        token=strtok(szMICRspBuf, " \n\r");  // Skip the first word
        token=strtok(NULL, " \n\r");  // Skip the second word
        enabled = (WORD)atoi(strtok(NULL," \n\r"));
        if(enabled)
                m_FreezeButton->SetNumProperty("Value", (int)FALSE);
        else
                m_FreezeButton->SetNumProperty("Value", (int)TRUE);

        // Get/Set the video input selection by sending a query message
        // to the VCR
        GetInputSel();

        return(TRUE);

}


////////////////////////////////////////////////////////////////////////////
// CVidSDlg message handlers
```

```
void CVidSDlg::OnClickSelectCamera(UINT, int, CWnd*, LPVOID)
{
	pBridgeView->m_SelectInput('1'); // Camera is Video 1
}

void CVidSDlg::OnClickSelectIDS(UINT, int, CWnd*, LPVOID)
{
	pBridgeView->m_SelectInput('2'); // IDS is Video 2
}



void CVidSDlg::OnClickFreeze(UINT, int, CWnd*, LPVOID lpParams)
{
	if(AFX_NUM_EVENTPARAM(short, lpParams))
		pBridgeView->freezeVideo(TRUE);
	else
		pBridgeView->freezeVideo(FALSE);

}

//////////////////////////
// OnDisplaySaved()
// Recall and display a previously saved image
// Action:
// Show file dialog to get name to display, if valid:
//  Freeze video - show freeze selected and disable camera and IDS buttons
//  Load image
//  Show image
//
void CVidSDlg::OnDisplaySaved(UINT, int, CWnd*, LPVOID)
{
#define		FILEDIAL	201

	WORD		result;
	char		*str;
	HCURSOR			hCursorOld;

	CVBControl	fileDialog;
	int			fileIOErr = 0;


	// Create the VBX file dialog
	CRect  dRect(0,0,100,100);
	result = (WORD)fileDialog.Create((LPCSTR)"CMDIALOG.VBX;CommonDialog;Save
Image",
		// The name CommonDialog MUST be used or it won't create
		WS_CHILD, dRect, this, FILEDIAL);
		// The dRect is a dummy since the CommonDialog VBX determines the rect.
	if (result){
		fileDialog.SetStrProperty("DefaultExt", "DVA");
		fileDialog.SetStrProperty("DialogTitle", "Display Image");
		fileDialog.SetStrProperty("Filter",
			"VideoLogic DVA (*.DVA)|*.DVA|Windows DIB (*.DIB)|*.DIB|All Files
(*.*)|*.*");
		fileDialog.SetNumProperty("FilterIndex", 1);
```

```
                fileDialog.SetNumProperty("Flags", OFN_OVERWRITEPROMPT);
                switch(pBridgeView->m_saveDestination){
                        case HardDisk:
                                fileDialog.SetStrProperty("InitDir","C:\\Bridge"); // Have
to use double \ because \ is an escape char
                                break;
                        case RemovableDisk:
                                fileDialog.SetStrProperty("InitDir","D:\\");
                                break;
                        case FloppyDisk:
                                fileDialog.SetStrProperty("InitDir","A:\\");
                                break;
                }
                fileDialog.SetNumProperty("CancelError",TRUE);

                // At last lets open the dialog
                fileDialog.SetNumProperty("Action", 1);

                if(fileDialog.m_nError == 0){
                        // Set the cursor to an hourglass
                        hCursorOld = SetCursor(LoadCursor(NULL, IDC_WAIT));

                        // Get fileName and extension
                        CString fileName(fileDialog.GetStrProperty("FileName"));
                        CString ext(fileName.Right(3));

                        // Freeze the video
                        m_FreezeButton->SetNumProperty("Value", (int)TRUE);

                    // Switch based on fileName extension
                    if(_strnicmp(ext.GetBuffer(4), "dva", 3) == 0){
                                // DVA type. Use the MIC FStore Load
                                CString CmdStr("FStore $VWin Load 3 ");
                                CmdStr += '"';
                                CmdStr += fileName;
                                CmdStr += '"';
                                CmdStr.MakeUpper(); // File names have to be upper case
                                str = CmdStr.GetBuffer(256);
                                result = pBridgeView->micWrite((LPSTR)str );
                        }else if(_strnicmp(ext.GetBuffer(4), "dib", 3) == 0){
                                // Since we can only directly display DVA images,
                                // First load the DIB image and write it to RAM disk as DVA
                                // then use FStore to display it.
                                CString ImLdStr("Image Load ");
                                ImLdStr += '"';
                                ImLdStr += fileName;
                                ImLdStr += '"';
                                ImLdStr += " DIB";
                                ImLdStr.MakeUpper(); // File names have to be upper case
                                str = ImLdStr.GetBuffer(256);
                                result = pBridgeView->micWrite((LPSTR)str );
                                // Save to RAM disk
                                CString ImSvStr("Image Save ");
                                ImSvStr += '"';
                                ImSvStr += "E:\\tmpib2va.dva";
                                ImSvStr += '"';
```

```
                                ImSvStr += " DVA NTSC FRAME";
                                ImSvStr.MakeUpper(); // File names have to be upper case
                                str = ImSvStr.GetBuffer(256);
                                result = pBridgeView->micWrite((LPSTR)str );
                                // Finally load from RAM disk
                                CString FSStr("FStore $VWin Load 3 ");
                                FSStr += "";
                                FSStr += "E:\\tmpib2va.dva";
                                FSStr += "";
                                FSStr.MakeUpper(); // File names have to be upper case
                                str = FSStr.GetBuffer(256);
                                result = pBridgeView->micWrite((LPSTR)str );


                        }else
                                AfxMessageBox("Wrong Image Type. Should be DVA or DIB.",
MB_OKIMB_ICONINFORMATION,0);

                        // Restore the old cursor
                        SetCursor(hCursorOld);

                } else
                        AfxMessageBox("Canceled display image.",
MB_OKIMB_ICONINFORMATION,0);
        } else
                AfxMessageBox("Image Display Error", MB_OKIMB_ICONINFORMATION,0);



}


//////////////////////////////////////////
// Send an inquiry to get the current input selection
void CVidSDlg::GetInputSel()
{
        char            pS[50];
        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);   // Device 1
        pS[1] = VS_INQUIRY;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)MD_InputSelectInq;  // MD_InputSelectInq
        pS[4] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 5);
}

//////////////////////////////////
void CVidSDlg::SetSourceButtons(BYTE source)
// Set the button values to the appropriate one
// NOTE: Camera is video line 1
//               IDS is video line 2
{
        switch (source) {
                case 0x11: // Video line 1
                        m_SelIDSButton->SetNumProperty("Value", (int) FALSE);
                        m_SelCamButton->SetNumProperty("Value", (int) TRUE);
```

```
                break;
        case 0x12: // Video line 2
                m_SelIDSButton->SetNumProperty("Value", (int) TRUE);
                m_SelCamButton->SetNumProperty("Value", (int) FALSE);
                break;
        case 0x21: // S-Video
        default:
                m_SelIDSButton->SetNumProperty("Value", (int) FALSE);
                m_SelCamButton->SetNumProperty("Value", (int) FALSE);
                break;
    }

}


////////////////////////////////////
// OnViscaInq(WPARAM wparam, LPARAM lparam)
// When an inquiry message completes, BridgeView sends a message
// with the result of the inquiry if it completed.
// It is up to this function to parse the message and perform
// desired action.
// This will only respond to source selection messages
LRESULT CVidSDlg::OnViscaInq(WPARAM wparam, LPARAM lparam)
{
    BYTE        *byteArray;

    byteArray = (BYTE*)lparam;
    // Messages of this type are destined for the Video select
    // dialog box.
    SetSourceButtons((unsigned char) byteArray[2]);

    return((long)0);
}

////////////////////////////////////
// OnViscaErr(WPARAM wparam, LPARAM lparam)
// When an error message is returned, BridgeView sends a message
// with the result code.
// It is up to this function to parse the message and perform
// desired action.
LRESULT CVidSDlg::OnViscaErr(WPARAM wparam, LPARAM lparam)
{
    CString     oS("");
    int             nChar = (int) wparam, i;
    char        outString[20], oBuf[30];
    BYTE        *byteArray;
    WORD        errorCode;

    byteArray = (BYTE*)lparam;

    errorCode = (WORD)byteArray[2];
    oS.LoadString(VS_STRINGTABLE | errorCode);
    oS += "  ";
    for(i = 0;i<nChar;i++){
        oS += _itoa((int)byteArray[i], oBuf, 16);
        oS += " : ";
```

```
        }
        AfxMessageBox(oS, MB_OKIMB_ICONINFORMATION,0);

    return((long)0);
}
```

```
// vidsdlg.h : header file
//

class CBridgeView;
class CVSCAdlg;

/////////////////////////////////////////////////////////////////////////////
// CVidSDlg dialog

class CVidSDlg : public CDialog
{
// Construction
public:
        CVidSDlg(CWnd* pParent = NULL);      // standard constructor

// Called by View
        CBridgeView*            pBridgeView;
        void SetViewPtr(CBridgeView* p);
        void SetVSCADlgPtr(CVSCAdlg* p);
        CVSCAdlg*       pVSCAdlg;


// Dialog Data
        //{{AFX_DATA(CVidSDlg)
        enum { IDD = IDD_VideoSelIDlg };
        CVBControl*     m_SelIDSButton;
        CVBControl*     m_SelCamButton;
        CVBControl*     m_FreezeButton;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CVidSDlg)
        afx_msg void OnClickSelectCamera(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickSelectIDS(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickFreeze(UINT, int, CWnd*, LPVOID);
        afx_msg void OnDisplaySaved(UINT, int, CWnd*, LPVOID);
        afx_msg LRESULT OnViscaInq(WPARAM wparam, LPARAM lparam);
        afx_msg LRESULT OnViscaErr(WPARAM wparam, LPARAM lparam);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

        // Override the base class virtual function
        // to set button states
        BOOL OnInitDialog();

        // Send Input selection inq
        void GetInputSel();
        // After a query sent to the VCR
        void SetSourceButtons(BYTE source);
```

};

```
// viscadev.h : VISCA header file
//
// The control computer is the VISCA 0 device.

// Define Constants
// Define Max size
#define     VS_MAXDEVICE     7
#define     VS_MAXMSG        14
#define     VS_MAXPACKET     16

// Define packet terminator byte
#define     VS_TERM                 (BYTE)0xff

// Define message header format type
#define     VS_RAWMODE              (BYTE)0x00
#define     VS_COMMAND1             (BYTE)0x01
#define     VS_COMMAND2             (BYTE)0x02
#define     VS_COMMAND3             (BYTE)0x03
#define     VS_COMMAND4             (BYTE)0x04
#define     VS_INQUIRY          (BYTE)0x09
#define     VS_CLEAR            (BYTE)0x00
#define     VS_CANCEL           (BYTE)0x20
#define     VS_ADDRESS          (BYTE)0x30
#define     VS_ACK                  (BYTE)0x40
#define     VS_COMP                 (BYTE)0x50
#define     VS_ERROR            (BYTE)0x60
#define     VS_NETCHANGE        (BYTE)0x38


// Define category code
#define     VS_INTERFACE        (BYTE)0x00
#define     VS_CONTROL_S        (BYTE)0x01
#define     VS_MEDIA_DEVICE     (BYTE)0x02
#define     VS_SWITCHER             (BYTE)0x03
#define     VS_VENDOR_XL        (BYTE)0x7e
#define     VS_VENDOR_ADJ       (BYTE)0x7f

// Define counter types
#define     VS_TOP_MID_END      (BYTE)0x01
#define     VS_4_DIGIT          (BYTE)0x11
#define     VS_HMS                  (BYTE)0x12
#define     VS_TIME_CODE        (BYTE)0x21

// Define error codes
#define     VS_ER_NETCHANGE  1      /* VISCA network changed */
#define     VS_ER_RAWIO              2      /* I/O Error */
#define     VS_ER_MEMORY     3      /* Can't alloc memory */
#define     VS_ER_INVALID    4      /* Invalid argument */
#define     VS_ER_REPLY      5      /* Illegal reply message */
#define     VS_ER_INITVISCA  6      /* Failed to init VISCA */

// Define address header of packet
#define     VS_BROADCAST        (BYTE)0x88
#define     VS_NORMAL           (BYTE)0x80

// Define timeout for packet
```

```
#define         VS_TIMEOUT          2       /* sec */
#define         VS_TIMEOUT_ID       1000

// Define the VS error message stringtable offset
#define         VS_STRINGTABLE      62000

// Define the Device Unavailable error returned from the VBX MSComm
#define         VBX_DEVUNAVAIL      68

#define         INPUT_Q                     1
#define         OUTPUT_Q            0

// Define some of the Media device inquiry codes
#define         MD_ModeInq                  0x01
#define         MD_PositionInq      0x03
#define         MD_InputSelectInq   0x13
#define         MD_OSDInq                   0x15

// Define the MD_ModeInq return codes
#define         MD_STOP                     0x00
#define         MD_STOP_TOP                 0x02
#define         MD_STOP_END                 0x04
#define         MD_STOP_EMER        0x06
#define         MD_FF               0x08
#define         MD_REW                      0x10
#define         MD_EJECT            0x18
#define         MD_STILL            0x20
#define         MD_SLOW2            0x24
#define         MD_SLOW1            0x26
#define         MD_PLAY                     0x28
#define         MD_FAST1            0x2a
#define         MD_SCAN                     0x2e
#define         MD_REV_SLOW2        0x34
#define         MD_REV_SLOW1        0x36
#define         MD_REV_PLAY                 0x38
#define         MD_REV_FAST1        0x3a
#define         MD_REV_SCAN                 0x3e
#define         MD_REC_PAUSE        0x40
#define         MD_RECORD           0x48

// Define a mode and tape position update period in milliseconds
// Change VS_VCR_UPDATE from 500 to 2000
#define         VS_VCR_UPDATE       2000
#define         VS_VCR_UPDATE_ID 1001
```

```
// vscadlg.cpp : implementation file
//

#include "stdafx.h"
#include "bridge.h"
#include "vscadlg.h"

#include "bridgdoc.h"

#include "bridgvw.h"

#include "viscadev.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

static UINT NEAR WM_VISCAINQ = RegisterWindowMessage((LPCSTR)"VISCA_INQ");
static UINT NEAR WM_VISCAERR = RegisterWindowMessage((LPCSTR)"VISCA_ERR");

/////////////////////////////////////////////////////////////////////////////
// CVSCAdlg dialog

CVSCAdlg::CVSCAdlg(CWnd* pParent /*=NULL*/)
        : CDialog(CVSCAdlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CVSCAdlg)
        m_Panel_Stop = NULL;
        m_Panel_REW = NULL;
        m_Panel_REC = NULL;
        m_Panel_Pause = NULL;
        m_Panel_FF = NULL;
        m_Panel_Play = NULL;
        m_TimePanel = NULL;
        m_TFSButton = NULL;
        //}}AFX_DATA_INIT
}

void CVSCAdlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CVSCAdlg)
        DDX_VBControl(pDX, IDC_PANEL_Stop, m_Panel_Stop);
        DDX_VBControl(pDX, IDC_PANEL_REW, m_Panel_REW);
        DDX_VBControl(pDX, IDC_PANEL_REC, m_Panel_REC);
        DDX_VBControl(pDX, IDC_PANEL_Pause, m_Panel_Pause);
        DDX_VBControl(pDX, IDC_PANEL_FF, m_Panel_FF);
        DDX_VBControl(pDX, IDC_PANEL_Play, m_Panel_Play);
        DDX_VBControl(pDX, IDC_TimePanel, m_TimePanel);
        DDX_VBControl(pDX, IDC_TimeFormatSelect, m_TFSButton);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CVSCAdlg, CDialog)
```

```
        //{{AFX_MSG_MAP(CVSCAdlg)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCASTOP, OnClickViscastop)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCAFF, OnClickViscaff)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCAPAUSE, OnClickViscapause)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCAPLAY, OnClickViscaplay)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCAREC, OnClickViscarec)
        ON_REGISTERED_MESSAGE(WM_VISCAINQ, OnViscaInq)
        ON_REGISTERED_MESSAGE(WM_VISCAERR, OnViscaErr)
        ON_VBXEVENT(VBN_CLICK, IDC_TimeFormatSelect, OnTimeFormatSelect)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCAREW, OnClickViscarew)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCA_FRAME_BACK, OnViscaFrameBack)
        ON_VBXEVENT(VBN_CLICK, IDC_VISCA_FRAME_FWD, OnViscaFrameFwd)
        ON_WM_TIMER()
        ON_WM_HSCROLL()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


#define VOL_LO 0
#define VOL_HI 255
////////////////////////////////////////////////////////////////
// Override the default OnInitDialog and the OnOK to start
// and kill the VS_VCR_UPDATE timer
BOOL CVSCAdlg::OnInitDialog()
{
        CScrollBar*     VolScroll;

        // First call the default
        CDialog::OnInitDialog();

        // Get the current mode at invocation.
        GetMode();
        GetOSDMode();

        SetTimer(VS_VCR_UPDATE_ID, VS_VCR_UPDATE,NULL);

        pBridgeView->micWrite((LPSTR)"Audio $VWin OFF");

        // Use GetDlgItem to get the volume scroll bar handle and init the scroll bar.
        if((VolScroll = (CScrollBar*)GetDlgItem(IDC_VOLUME_SB)) != NULL){
                VolScroll->SetScrollRange(VOL_LO, VOL_HI, FALSE);
                VolScroll->SetScrollPos(Volume, TRUE);
        } else
                AfxMessageBox("Didn't init volume", MB_OK, 0);


        return(TRUE);

}

////////////////////////////////
// This handles the volume scroll bar on the dialog
// Because the scroll bar pointer is passed as an argument,
// I cannot set some of the parameters of the scroll bar.
// Since it is a control scroll its range is 0. I want to set it
// to 0 to 255, but it jumps to 255 when you try to move the
```

```
// control box the first time. Add an if statement to watch for
// big nPos
void CVSCAdlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        int             result;
        char    szMICCmdBuf[256];

        pScrollBar->SetScrollRange(0, 255, TRUE);

        switch(nSBCode){
                case SB_LINELEFT:
                        break;
                case SB_LINERIGHT:
                        break;
                case SB_PAGELEFT:
                        break;
                case SB_PAGERIGHT:
                        break;
                case SB_THUMBPOSITION:
                        if((nPos >=0) && (nPos <= 255)){
                                Volume = (UINT) nPos;
                                pScrollBar->SetScrollPos((int) Volume, TRUE);
                                wsprintf((LPSTR) szMICCmdBuf, (LPSTR) "Audio $VWin Fade
%d 1",
                                        Volume);

                                result = pBridgeView->micWrite((LPSTR)szMICCmdBuf);
                        } else  pScrollBar->SetScrollPos(0, TRUE);
                        break;
                default:
                        break;
        }

        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);



}


void CVSCAdlg::OnOK()
{
        KillTimer(VS_VCR_UPDATE_ID);

  // Call the base class function
        CDialog::OnOK();

        // Set the VBX control pointers to NULL to prevent a memory exception
        // caused by using the same location upon each invocation of the dialog
        m_Panel_Stop = NULL;
        m_Panel_REW = NULL;
        m_Panel_REC = NULL;
        m_Panel_Pause = NULL;
        m_Panel_FF = NULL;
        m_Panel_Play = NULL;
        m_TimePanel = NULL;
```

```
        m_TFSButton = NULL;
}


//////////////////////////////////////////////////////////////////
// OnTimer()
// If the ID is VS_VCR_UPDATE_ID, then uptate time code display
// and the button mode
void CVSCAdlg::OnTimer(UINT nIDEvent)
{
        if(nIDEvent == VS_VCR_UPDATE_ID){
                //GetMode();  // The button labels flash too much
                // You can only get the time code if the player is not stopped
                // or ejected
                if((m_playerMode != MD_STOP) && (m_playerMode != MD_STOP_TOP) &&
                        (m_playerMode != MD_STOP_END) && (m_playerMode !=
MD_STOP_EMER) &&
                        (m_playerMode != MD_EJECT))
                        GetTimeCode();
        }

        CDialog::OnTimer(nIDEvent);
}

//////////////////////////////////////////////////////////////////
// Helper functions

void CVSCAdlg::SetViewPtr(CBridgeView* p)
{

        pBridgeView = p;

}

//////////////////////////////////
// Set the caption of the time format select button to the
// current format.
void CVSCAdlg::SetTFSButtonCaption()
{
        switch(m_OSDPage){
                case 0x01:
                        m_TFSButton->SetStrProperty("Caption","Time Code HH:MM:SS:FF");
                        break;
                case 0x03:
                        m_TFSButton->SetStrProperty("Caption","Rec Date MM/DD/YY");
                        break;
                case 0x04:
                        m_TFSButton->SetStrProperty("Caption","Rec Time HH:MM:SS");
                        break;
                default:
                        m_TFSButton->SetStrProperty("Caption"," ");
                        break;
        }
}

//////////////////////////////////
```

```
// Send an inquiry to get the current time code off of the tape
// This inquiry does not work in the Stop mode.
// The time code inquiry depends on the OSD mode.
void CVSCAdlg::GetTimeCode()
{
	char		pS[50];


	pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);   // Device 1
	pS[1] = VS_INQUIRY;
	pS[2] = VS_MEDIA_DEVICE;
	pS[3] = m_InqCode = MD_PositionInq;  // MD_PositionInq

	switch(m_OSDPage){
		case 0x01:
			pS[4] = (BYTE)0x20;  // Time = HHMMSSFF FF= frame
			break;
		case 0x03:
			pS[4] = (BYTE)0x41;  // Rec Date MM/DD/YY
			break;
		case 0x04:
			pS[4] = (BYTE)0x42;  // Rec Time HH:MM:SS
			break;
	}

	pS[5] = VS_TERM;

	pBridgeView->m_WriteMessage(pS, 6);


}

//////////////////////////////////////////////
// Send an inquiry to get the current player mode.
void CVSCAdlg::GetMode()
{
	char		pS[50];


	pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);   // Device 1
	pS[1] = VS_INQUIRY;
	pS[2] = VS_MEDIA_DEVICE;
	pS[3] = m_InqCode = (BYTE)MD_ModeInq;  // MD_ModeInq
	pS[4] = VS_TERM;

	pBridgeView->m_WriteMessage(pS, 5);


}



//////////////////////////////////////
// Set all panel labels to black
void CVSCAdlg::ResetPanelColor()
{
```

```
    m_Panel_Stop->SetNumProperty("ForeColor", BLACK);
        m_Panel_REW->SetNumProperty("ForeColor", BLACK);
        m_Panel_REC->SetNumProperty("ForeColor", BLACK);
        m_Panel_Pause->SetNumProperty("ForeColor", BLACK);
        m_Panel_FF->SetNumProperty("ForeColor", BLACK);
        m_Panel_Play->SetNumProperty("ForeColor", BLACK);

}

//////////////////////////////////////////////
// Send an inquiry to get the current OSD mode.
void CVSCAdlg::GetOSDMode()
{
        char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);   // Device 1
        pS[1] = VS_INQUIRY;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = m_InqCode = (BYTE)MD_OSDInq;  // MD_OSDInq
        pS[4] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 5);


}

//////////////////////////////////////
// Set the on-screen-display page to m_OSDPage
void CVSCAdlg::SetOSDPage()
{
        char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);   // Device 1
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)MD_OSDInq;  // MD_OSD and MD_OSDInq are same
        pS[4] = m_OSDPage;
        pS[5] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 6);


}

/////////////////////////////////////////////////////////////////////////////
// CVSCAdlg message handlers


void CVSCAdlg::OnClickViscastop(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
```

```
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x01;  // MD_Mode1
        pS[4] = (BYTE)0x00;  // Stop
        pS[5] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 6);

        // Inquire about player mode
         GetMode();

}

////////////////////////////////////
// OnClickViscaff()
// Perform action when user clicks on the Fast Forward button
// If stopped, then fast forward, else Scan
void CVSCAdlg::OnClickViscaff(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];

        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x01;  // MD_Mode1

        if(m_playerMode == MD_PLAY) pS[4] = (BYTE)0x2E;// Scan
        else pS[4] = (BYTE)0x08;  // Fast forward

        pS[5] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 6);

        // Inquire about player mode
        GetMode();

}

////////////////////////////////////
// OnClickViscarew()
// Perform action when user clicks on the Rewind button
// If stopped, then rewind, else Reverse Scan
void CVSCAdlg::OnClickViscarew(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];

        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x01;  // MD_Mode1

        if(m_playerMode == MD_PLAY) pS[4] = (BYTE)0x3E;// Reverse Scan
        else pS[4] = (BYTE)0x10;  // rewind

        pS[5] = VS_TERM;
```

```
        pBridgeView->m_WriteMessage(pS, 6);

        // Inquire about player mode
        GetMode();

}

////////////////////////////////////
// OnClickViscapause()
// Perform action when user clicks on the Pause button
void CVSCAdlg::OnClickViscapause(UINT, int, CWnd*, LPVOID)
{
                char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);     // Sent to device 1
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x01;  // MD_Mode1
        pS[4] = (BYTE)0x20;  // Pause
        pS[5] = VS_TERM;

        pBridgeView->m_WriteMessage(pS, 6);

        // Get position
        GetTimeCode();

        // Inquire about player mode
        GetMode();
}

////////////////////////////////////
// OnClickViscaplay()
// Perform action when user clicks on the Play button
void CVSCAdlg::OnClickViscaplay(UINT, int, CWnd*, LPVOID)
{

        char            pS[50];


        pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
        pS[1] = VS_COMMAND1;
        pS[2] = VS_MEDIA_DEVICE;
        pS[3] = (BYTE)0x01;  // MD_Mode1
        pS[4] = (BYTE)0x28;  // Play
        pS[5] = VS_TERM;


        pBridgeView->m_WriteMessage(pS, 6);

        // Inquire about player mode
        GetMode();

}

////////////////////////////////////
```

```
// OnClickViscarec()
// Perform action when user clicks on the Record button
void CVSCAdlg::OnClickViscarec(UINT, int, CWnd*, LPVOID)
{
	char		pS[50];

	if(MessageBox((LPCSTR)"Are you sure you want to record?",
					(LPCSTR)"Are you sure?",
					MB_ICONQUESTION | MB_OKCANCEL) == IDOK) {
		pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
		pS[1] = VS_COMMAND1;
		pS[2] = VS_MEDIA_DEVICE;
		pS[3] = (BYTE)0x01;  // MD_Mode1
		pS[4] = (BYTE)0x48;  // Record
		pS[5] = VS_TERM;


		pBridgeView->m_WriteMessage(pS,  6);

		// Inquire about player mode
		GetMode();

	}
}

//////////////////////////////////////
// OnViscaInq(WPARAM wparam, LPARAM lparam)
// When an inquiry message completes, BridgeView sends a message
// with the result of the inquiry if it completed.
// It is up to this function to parse the message and perform
// desired action.
LRESULT CVSCAdlg::OnViscaInq(WPARAM wparam, LPARAM lparam)
{
	CString		oS("");
	int			nChar = (int) wparam;
	char		outString[20];
	BYTE		*byteArray;
	unsigned long hour, min, sec, frame, month, day, year;

	byteArray = (BYTE*)lparam;
	// Switch based on the  m_InqCode
	switch ( m_InqCode ){
		case MD_ModeInq:  // Mode Inquiry
			switch (byteArray[2]) {
				case	MD_STOP_TOP:;
					m_TimePanel->SetStrProperty("Caption","Beginning of
Tape");
					ResetPanelColor();
					m_Panel_Stop->SetNumProperty("ForeColor",
YELLOW);
					m_playerMode = (UINT)byteArray[2];
					break;
				case	MD_STOP_END:
					m_TimePanel->SetStrProperty("Caption","End of
Tape");
					ResetPanelColor();
```

```
                    m_Panel_Stop->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_STOP:
                case MD_STOP_EMER:
                    ResetPanelColor();
                    m_Panel_Stop->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_FF:
                    ResetPanelColor();
                    m_Panel_FF->SetNumProperty("ForeColor", YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_REW:
                    ResetPanelColor();
                    m_Panel_REW->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_STILL:
                    ResetPanelColor();
                    m_Panel_Pause->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_PLAY:
                    ResetPanelColor();
                    m_Panel_Play->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_SCAN:
                    ResetPanelColor();
                    m_Panel_Play->SetNumProperty("ForeColor",
YELLOW);
                    m_Panel_FF->SetNumProperty("ForeColor", YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_REV_SCAN:
                    ResetPanelColor();
                    m_Panel_Play->SetNumProperty("ForeColor",
YELLOW);
                    m_Panel_REW->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
                case MD_RECORD:
                    ResetPanelColor();
                    m_Panel_REC->SetNumProperty("ForeColor",
YELLOW);
                    m_playerMode = (UINT)byteArray[2];
                    break;
```

```
                                case    MD_EJECT:
                                case    MD_SLOW2:
                                case    MD_SLOW1:
                                case    MD_FAST1:
                                case    MD_REV_SLOW2:
                                case    MD_REV_SLOW1:
                                case    MD_REV_PLAY:
                                case    MD_REV_FAST1:
                                case    MD_REC_PAUSE:
                                        m_playerMode = (UINT)byteArray[2];
                                        break;

                                default:
                                        /***
                                        oS = "Inq Result ";
                                        for(i = 0;i<nChar;i++, lparam++){
                                                oS += _itoa((int)*(LPSTR)lparam, oBuf, 16);
                                                oS += " : ";
                                        }
                                        AfxMessageBox(oS,
MB_OKIMB_ICONINFORMATION,0);
                                        ***/

                                        break;
                        }
                        break;

                case MD_PositionInq:  // Position Inquiry
                        switch (byteArray[2]) {
                                case 0x21: // Time-Code Inquiry
                                        hour = (unsigned long) byteArray[3];
                                        min = (unsigned long) byteArray[4];
                                        sec = (unsigned long) byteArray[5];
                                        frame = (unsigned long) byteArray[6];
                                        wsprintf((LPSTR) outString, (LPSTR)
"%02.2lx:%02.2lx:%02.2lx:%02.2lx",
                                                hour, min, sec, frame );

                                        m_TimePanel->SetStrProperty("Caption",
(LPSTR)outString);

                                        break;

                                case 0x41: // Rec Date Inquiry
                                        year = (unsigned long) ((byteArray[3] & 0x0f) << 4) |
                                                                                (byteArr
(byteArray[4] & 0x0f);
                                        month = (unsigned long) byteArray[5];
                                        day = (unsigned long) byteArray[6];
                                        wsprintf((LPSTR) outString, (LPSTR)
"%2.2lx/%2.2lx/%2.2lx",
                                                month, day, year);

                                        m_TimePanel->SetStrProperty("Caption",
(LPSTR)outString);
```

```
                                        break;

                                case 0x42: // Rec Time Inquiry
                                        hour = (unsigned long) byteArray[3];
                                        min = (unsigned long) byteArray[4];
                                        sec = (unsigned long) byteArray[5];
                                        wsprintf((LPSTR) outString, (LPSTR)
"%02.2lx:%02.2lx:%02.2lx",
                                                hour, min, sec);

                                        m_TimePanel->SetStrProperty("Caption",
(LPSTR)outString);
                                        break;

                                default:
                                        break;
                        }
                        break;

                case MD_OSDInq: // OSD Inquiry
                        m_OSDPage = (BYTE) (byteArray[2] & 0x0f);
                        SetTFSButtonCaption();
                        break;

                default:

                        break;
        }


    return((long)0);
}

//////////////////////////////////////
// OnViscaErr(WPARAM wparam, LPARAM lparam)
// When an error message is returned, BridgeView sends a message
// with the result code.
// It is up to this function to parse the message and perform
// desired action.
LRESULT CVSCAdlg::OnViscaErr(WPARAM wparam, LPARAM lparam)
{
        CString         oS("");
        int                     nChar = (int) wparam, i;
        char            outString[20], oBuf[30];
        BYTE            *byteArray;
        WORD            errorCode;

        byteArray = (BYTE*)lparam;
        // Switch based on the type of response
        switch (byteArray[2]) {
                case 0x46: // Counter Type error
                        wsprintf((LPSTR) outString, (LPSTR) "No Time Code");

                        m_TimePanel->SetStrProperty("Caption", (LPSTR)outString);
```

```
                        break;

                default:
                        errorCode = (WORD)byteArray[2];
                        oS.LoadString(VS_STRINGTABLE | errorCode);
                        oS += "  ";
                        for(i = 0;i<nChar;i++){
                                oS += _itoa((int)byteArray[i], oBuf, 16);
                                oS += " : ";
                        }
                        AfxMessageBox(oS, MB_OKIMB_ICONINFORMATION,0);

                        break;
        }


    return((long)0);
}




//////////////////////////////////////////
// Respond to the time format select button.
// The format select corresponds to MD_OSD pages 1, 3, or 4
// loop through these selections.
void CVSCAdlg::OnTimeFormatSelect(UINT, int, CWnd*, LPVOID)
{
        switch(m_OSDPage){
                case 0x01:
                        m_OSDPage = 0x03;
                        break;
                case 0x03:
                        m_OSDPage = 0x04;
                        break;
                case 0x04:
                        m_OSDPage = 0x01;
                        break;
                default:
                        m_OSDPage = 0x01;
                        break;
        }

        SetTFSButtonCaption();
        SetOSDPage();

}

//////////////////////////
// Step back one frame
void CVSCAdlg::OnViscaFrameBack(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];

        if(m_playerMode == MD_STILL){   // Must be Paused
```

```
            pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
            pS[1] = VS_COMMAND1;
            pS[2] = VS_MEDIA_DEVICE;
            pS[3] = (BYTE)0x02;  // MD_Mode2
            pS[4] = (BYTE)0x03;  // Frame Back
            pS[5] = VS_TERM;


            pBridgeView->m_WriteMessage(pS, 6);

            // Get position
            GetTimeCode();

            // Inquire about player mode
            GetMode();
        }

}


//////////////////////////
// Step forward one frame
void CVSCAdlg::OnViscaFrameFwd(UINT, int, CWnd*, LPVOID)
{
        char            pS[50];

        if(m_playerMode == MD_STILL){   // Must be Paused
            pS[0] = (BYTE)(VS_NORMAL | (BYTE)0x01);
            pS[1] = VS_COMMAND1;
            pS[2] = VS_MEDIA_DEVICE;
            pS[3] = (BYTE)0x02;  // MD_Mode2
            pS[4] = (BYTE)0x02;  // Frame Forward
            pS[5] = VS_TERM;


            pBridgeView->m_WriteMessage(pS, 6);

            // Get position
            GetTimeCode();

            // Inquire about player mode
            GetMode();
        }

}
```

```
// vscadlg.h : header file
//

#define         YELLOW              0x0000ffff
#define         BLACK         0x00000000

class   CBridgeView;

static   UINT   Volume = 128;


/////////////////////////////////////////////////////////////////////////
// CVSCAdlg dialog

class CVSCAdlg : public CDialog
{
// Construction
public:
        CVSCAdlg(CWnd* pParent = NULL);    // standard constructor

// Called by View
        CBridgeView*            pBridgeView;
        void SetViewPtr(CBridgeView* p);

// Dialog Data
        //{{AFX_DATA(CVSCAdlg)
        enum { IDD = IDD_VISCAAct };
        CVBControl*    m_Panel_Stop;
        CVBControl*    m_Panel_REW;
        CVBControl*    m_Panel_REC;
        CVBControl*    m_Panel_Pause;
        CVBControl*    m_Panel_FF;
        CVBControl*    m_Panel_Play;
        CVBControl*    m_TimePanel;
        CVBControl*    m_TFSButton;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(CVSCAdlg)
        afx_msg void OnClickViscastop(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickViscaff(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickViscapause(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickViscaplay(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickViscarec(UINT, int, CWnd*, LPVOID);
        afx_msg LRESULT OnViscaInq(WPARAM wparam, LPARAM lparam);
        afx_msg LRESULT OnViscaErr(WPARAM wparam, LPARAM lparam);
        afx_msg void OnTimeFormatSelect(UINT, int, CWnd*, LPVOID);
        afx_msg void OnClickViscarew(UINT, int, CWnd*, LPVOID);
        afx_msg void OnViscaFrameBack(UINT, int, CWnd*, LPVOID);
        afx_msg void OnViscaFrameFwd(UINT, int, CWnd*, LPVOID);
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
```

```
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

protected:
        // Base class overrides
//      BOOL  OnInitDialog();
        void    OnOK();

public:
        // VISCA Support
        void    GetTimeCode(void);
        void    GetMode(void);
        void    ResetPanelColor(void);
        void    GetOSDMode(void);
        void    SetOSDPage(void);
        void    SetTFSButtonCaption(void);
        UINT    m_playerMode;
        BYTE    m_OSDPage; // page 1, 3, or 4
        // Since the returned inquiry messages are NOT unique we must identify
        // which inquiry was last requested.
        BYTE    m_InqCode;
};
```