DATA TRANSLATION®

UM-17792-D

GLOBAL LAB Image/2 API Manual

(for Windows)

Fourth Edition
March, 2001

Data Translation, Inc.
100 Locke Drive
Marlboro, MA 01752-1192
(508) 481-3700
www.datatranslation.com
Fax: (508) 481-8620
E-mail: info@datx.com

Table of Contents

About this Manual

This manual describes the API for the main application of GLOBAL LAB® Image/2 (GLI/2) as well as the tools that are provided for it.

Intended Audience

This manual is intended for application programmers who want to add custom tools to GLI/2 or create an imaging application using GLI/2. You should be familiar with Windows programming using the Windows® 98, Windows NT 4.0, Windows 2000, or Windows Me (Millennium Edition) operating system, Visual C++, and the Microsoft® Foundation Classes (MFC). In addition, if you intend to modify the Picture tool or DTiX server code, you should be familar with COM programming.

What You Should Learn from this Manual

The main GLI/2 application and all its tools provide an object-oriented set of APIs. These APIs are included with the application and each tool. This manual describes how you can create your own custom tools and imaging application using the API for GLI/2 and its tools.

The manual is organized as follows:

- Chapter 1, "Introducing the GLI/2 API," provides an overview of the API for GLI/2 main application, and provides installation and technical support information.
- Chapter 2, "Using the GLI/2 API," describes the API for the GLI/2 main application.
- Chapter 3, "Using the Arithmetic Tool API," describes the API for the Arithmetic tool.

- Chapter 4, “Using the AVI Player Tool API,” describes the API for the AVI Player tool.
- Chapter 5, “Using the Blob Analysis Tool API,” describes the API for the Blob Analysis tool.
- Chapter 6, “Using the Custom Script Tool API,” describes the API for the Custom Script tool.
- Chapter 7, “Using the Edge Finder Tool API,” describes the API for the Edge Finder tool.
- Chapter 8, “Using the File Manager Tool API,” describes the API for the File Manager tool.
- Chapter 9, “Using the Filter Tool API,” describes the API for the Filter tool.
- Chapter 10, “Using the Histogram Tool API,” describes the API for the Histogram tool.
- Chapter 11, “Using the Image Modifier Tool API,” describes the API for the Image Modifier tool.
- Chapter 12, “Using the Line Profile Tool API,” describes the API for the Line Profile tool.
- Chapter 13, “Using the Measurement Tool API,” describes the API for the Measurement tool.
- Chapter 14, “Using the Morphology Tool API,” describes the API for the Morphology tool.
- Chapter 15, “Using the Picture Tool API,” describes the API for the Picture tool.
- Chapter 16, “Using the Pixel Change Tool API,” describes the API for the Pixel Change tool.
- Chapter 17, “Using the Serial I/O Tool API,” describes the API for the Serial I/O tool.
- Chapter 18, “Using the Sound Tool API,” describes the API for the Sound tool.

- Chapter 19, "Using the Text Tool API," describes the API for the Text tool.
- Chapter 20, "Using the Threshold Tool API," describes the API for the Threshold tool.
- Chapter 21, "Creating GLI/2 Tools," describes how to create custom tools using GLI/2.
- An index completes this manual.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information or information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.
- Method names and items that you select or type are shown in **bold**.
- Parameter names are shown in *italic*.

Related Information

Refer to the following documents for more information on using GLI/2:

- *GLOBAL LAB Image/2 User's Manual*, which is shipped with the software.
- GLOBAL LAB Image/2 online help, which is part of the GLI/2 software.

Where To Get Help

Should you run into problems installing or using GLI/2, the Data Translation Technical Support Department is available to provide technical assistance. Refer to page 6 for more information. If you are outside the U.S. or Canada, call your local distributor, whose number is listed in your Data Translation product handbook.

Introducing the GLI/2 API

What is the GLI/2 API?

GLI/2 is an application and an API dedicated to image processing. It provides an object-oriented approach to all of the needed operations for images, region of interests (ROIs), and other commonly needed operations found in most imaging processing applications. All tools, all tool APIs, and the GLI/2 main application use this API.

The following subsections describe what the API is and what is not in more detail.

What the API Is

The GLI/2 API is a small, robust, and easy-to-use core imaging API that can be used in all areas of imaging. At the center of the GLI/2 API are two types of objects: Image objects and ROI objects.

The API currently supports the following types of Image objects:

- Binary,
- 8-bit grayscale,
- 16-bit grayscale,
- 32-bit grayscale,
- Floating-point grayscale,
- 24-bit RGB true-color, and
- 24-bit HSL color image objects.

All Image objects are derived from a virtual Base Class object and operate the same way.

The API currently supports the following ROI objects:

- Point,
- Rectangular,

- Elliptical,
- Line,
- Freehand line,
- Poly line,
- Freehand, and
- Poly freehand.

All ROI objects are derived from a virtual base class ROI object and operate the same way.

Along with these two central imaging object types, other imaging objects, including the following, are often used to create imaging applications:

- Graph and Curve objects are often used for graphing two-dimensional data that is derived from an image.
- List objects keep a list of any type of GLI/2 base class that is derived from an object.
- Calibration objects convert pixel measurements to real-world measurements.

Although the set of objects included in this API is small, each object tries to supply all needed functionality for its type of object. Image objects, for example, have methods for displaying the image in multiple ways, accessing its data in multiple ways, printing the image, clipboard access, file I/O, and more.

Using this small set of objects, it very easy to implement the GLI/2 main application and all of its tools.

What the API Is Not

Although GLI/2 supplies many common imaging operations, such as arithmetic, blob analysis, display, filtering, histograms, line profiles, morphological processing, thresholding, and more, this specialized functionality is not part of the core API. All these types of processes are located in separate tool APIs. For more information on a specific tool API, see the appropriate chapter of this document.

Note: All tool APIs and the core API work together. If you wish, you can create a custom tool, a custom imaging application, or a custom algorithm that uses every type of functionality at the same time. Refer to Chapter 21 starting on page 661 for more information on creating custom tools.

Installation

The GLI/2 API is installed during the GLI/2 product installation. Please refer to the installation instructions in the *GLOBAL LAB Image/2 User's Manual*.

Service and Support

The goal of this manual is to help you use the APIs for the GLI/2 main application and its tools. If you have difficulty using these APIs, Data Translation's Technical Support Department is available to provide technical assistance. Support upgrades, technical information, and software are also available.

All customers can always obtain the support needed. The first 90 days are complimentary, as part of the product's original warranty, to help you get your system running. Customers who call outside of this time frame can either purchase a support contract or pay a nominal fee (charged on a per-incident basis).

For "priority support," purchase a support contract. Support contracts guarantee prompt response and are very affordable; contact your local sales office for details.

Refer to the Data Translation Support Policy located at the end of this manual for a list of services included and excluded in our standard support offering.

Telephone Technical Support

Telephone support is normally reserved for original warranty and support-contract customers. Support requests from non-contract or out-of-warranty customers are processed after requests from original warranty and support-contract customers.

For the most efficient service, please complete the form on page 8 and be at your computer when you call for technical support. This information helps to identify specific system and configuration-related problems and to replicate the problem in house, if necessary.

You can reach the Technical Support Department by calling (508) 481-3700 x1401.

If you are located outside the USA, call your local distributor. The name and telephone number of you nearest distributor are provided in your Data Translation catalog.

If you are leaving a message to request a support call, please include the following information:

- Your name (please include proper spelling),
- Your company or organization (please include proper spelling),
- A phone number,
- An email address where you can be reached,
- The hardware / software product you need help on,
- A summary of the issue or question you have,
- Your contract number, if applicable, and
- Your product serial number or purchase date.

Omitting any of the above information may delay our ability to resolve your issue.

Information Required for Technical Support

Name:_____Phone_____
Contract Number: _____
Address: _____
_____

Data Translation hardware product(s): _____
serial number: _____
configuration: _____
Data Translation device driver - SPO number: _____ _____
_____ version: _____
Data Translation software - SPO number:_____ _____
serial number: _____ version:_____
PC make/model: _____
operating system: _____ version:_____
Windows version: _____
processor: _____ speed:_____
RAM:_____ hard disk space:_____
network/number of users: _____ disk cache:_____
graphics adapter: _____ data bus:_____

I have the following boards and applications installed in my system:_____
_____
_____
I am encountering the following problem(s): _____
_____
_____
_____
and have received the following error messages/codes: _____
_____
_____
I have run the board diagnostics with the following results: _____
_____
You can reproduce the problem by performing these steps:
1. _____
_____
2. _____
_____
3. _____
_____

E-Mail and Fax Support

You can also get technical support by e-mailing or faxing the Technical Support Department:

- **E-mail**: You can reach Technical Support at the following address: tsupport@datx.com

  Ensure that you provide the following minimum information:

  – Your name,

  – Your company or organization,

  – A phone number,

  – An email address where you can be reached,

  – The hardware/software product you need help on,

  – A summary of the issue you are experiencing,

  – Your contract number, if applicable, and

  – Your product serial number or purchase date.

  Omitting any of the above information may delay our ability to resolve your issue.

- **Fax**: Please photocopy and complete the form on page 8, then fax Technical Support at the following number: (508) 481-8620.

Support requests from non-contract and out-of-warranty customers are processed with the same priority as telephone support requests.

World-Wide Web

For the latest tips, software fixes, and other product information, you can always access our World-Wide Web site free of charge at the following address: http://www.datatranslation.com

2

Using the GLI/2 API

Overview of the GLI/2 API

The API for the GLI/2 main application consists of a set of object-oriented classes that are derived from a base GLI/2 object. The hierarchy of the classes is as follows:

- ROI base objects, which include the following types:
  - point,
  - line,
  - poly line,
  - freehand line,
  - rectangular,
  - elliptical,
  - freehand, and
  - poly freehand.
- Image base objects, which include the following types:
  - binary,
  - 24-bit RGB color,
  - 24-bit HSL color,
  - 8-bit grayscale,
  - 32-bit grayscale,
  - 16-bit grayscale, and
  - floating-point grayscale.
- Curve,
- Graph,
- List, and
- Calibration.

Each class is documented separately in this chapter. After describing the class in general, each method is described in detail. Methods are documented in groups according to the operation they perform instead of alphabetically.

The image and ROI classes are multi-level, virtually-derived objects; therefore, they are implemented in separate classes. However, their methods, being truly virtual, operate the same way. For this reason, the methods are documented only once. For example, the **Show( )** method operates exactly the same way for all image classes. If any of the methods for a specific class operates differently, the difference is noted when the specific method is described.

The GLI/2 Base Class Object

All objects in the GLI/2 API are derived from a base class named CcHLObject; therefore, all GLI/2 objects contain the following items:

- **Name** – All objects in GLI/2 can have a name assigned to them. Using names, it is possible to keep track of objects. For example, you could request an ROI object from a List object by asking for the “lower-right” ROI in the list. This is a convenient method of tracking objects instead of keeping track of them in the usual ways. This also makes it easy to assign names to images. The length of the name for all objects is set to the Windows constant _MAX_PATH.
- **Type** – In object-oriented programming, it is useful to know what type of object you are pointing to for error checking reasons. All objects in GLI/2 have a type assigned to them.

Note: The Base Class object CcHLObject is not meant to be used directly.

The methods for the Base Class object, grouped by method, are listed in Table 1.

Table 1: Base Class Object Methods

Method Type	Method Name	Description
Name Methods	GetName( )	Returns the name of an object.
	SetName( )	Sets the name of an object.
Type Methods	GetType( )	Returns the object's type.

Name Methods

These methods set and retrieve the name for all object types. This section describes the name methods in detail.

GetName

Syntax `char* GetName(void);`

Description Returns the name of the object.

Return Values

NULL	Unsuccessful.
The name of the object.	Successful.

SetName

Syntax `int SetName(char* cNewName);`

Description Sets the name of the object.

Parameters

Name:	cNewName
Description:	New name for the object. The length of the string is limited to _MAX_PATH.

Return Values

–1	Unsuccessful.
0	Successful.

Type Methods

This method retrieves the object type for all objects. This section describes the type method in detail.

GetType

Syntax `int GetType(void);`

Description Returns the object's type.

Parameters

Name: cNewName

Description: New name for the object. The length of the string is limited to _MAX_PATH.

Return Values

HLOBJECT_UNDEFINED	Object has not yet been defined.
HLOBJECT_TYPE_IMAGE	Image object. To determine the type of Image object, see **GetImageType( )** on page 62.
HLOBJECT_TYPE_ROI	ROI object. To determine the type of ROI object, see **GetROIType( )** on page 102.
HLOBJECT_TYPE_CURVE	Curve object.
HLOBJECT_TYPE_GRAPH	Graph object.
HLOBJECT_TYPE_LIST	List object.
HLOBJECT_TYPE_CALIBRATION	Calibration object.
HLOBJECT_TYPE_NUMBER	Number object (used for point and click scripting).
HLOBJECT_TYPE_STRING	String object (used for point and click scripting).

Image Objects

Image objects are the core objects of the GLI/2 main application and API. An Image object is a class that supports all of the needed functionality for all images in an imaging application.

In the field of imaging, different types of images can be used depending on the requirements of the application. GLI/2 supports binary, 8-bit, 16-bit, 32-bit, and floating-point grayscale images, as well as a 24-bit true-color RGB and 24-bit color HSL images.

All methods that are specific to each type of image (if the API were written in C) are virtual C++ methods, making them operate the same way. When writing an application, you can use the base class pointer with most all methods. For example, when showing an image in a window, regardless of the image's type, you can always use the following code for the operation:

```
CImage->Show();
```

The Image objects provided in the GLI/2 API are listed in Table 2.

Table 2: Image Objects

Image Objects	Description
Binary Image Object	Contains pixels in the range from 0 to 1 only. A value of 0 is considered a background value and has a color of white. A value of 1 is considered a foreground value and has a color of black.
8-Bit Grayscale Image Object	The standard type of image used in almost all of today's imaging applications. A pixel in this type of image can have a value from 0 to 255.
16-Bit Grayscale Image Object	A true 16-bit Image object where each pixel in the image can contain any value that a 16-bit unsigned integer value can contain in the operating system. This object linearly scales the 16-bit image data automatically when displaying it in a window.

Table 2: Image Objects

Image Objects	Description
32-Bit Grayscale Image Object	A true 32-bit Image object where each pixel in the image can contain any value that a 32-bit integer value can contain in the operating system. These include both negative and positive values. This object linearly scales the 32-bit image data automatically when displaying it in a window.
Floating-Point Grayscale Image Object	A true floating-point Image object where each pixel in the image can contain any value that a floating-point value can contain in the operating system. These include both negative and positive values. This object linearly scales the floating-point image data automatically when displaying it in a window.
24-Bit True-Color RGB Image Object	A 24-bit RGB true-color Image object. Each pixel in the image contains an 8-bit red, 8-bit green, and 8-bit blue color plane. This image can be accessed using its red, green, or blue color planes. It can also be accessed using its luminance (brightness) value.
24-Bit HSL True-Color RGB Image Object	A 24-bit HSL true-color Image object. Each pixel in the image contains an 8-bit hue, 8-bit saturation, and 8-bit luminance color plane. This image can be accessed using its hue, saturation, or luminance color planes. Note that the range for hue, saturation, and luminance is 0 to 240.

The hierarchy of the Image object classes is shown in Table 3.

Table 3: Image Object Classes Hierarchy

Class Name	Description	Include File
CcHLObject	GLI/2 Base Class Object	
CcImage	Virtual Base Class Image Object	C_IMAGE.H
CcBinaryImage	Binary Image Object	C_BINARY.H
Cc24BitRGBImage	24-bit RGB Color Image Object	C_24BIT.H
Cc24BitHSLImage	24-bit HSL Color Image Object	C_24BITHSL.H

Table 3: Image Object Classes Hierarchy (cont.)

Class Name	Description	Include File
CcGrayImage256	8-bit Grayscale Image Object	C_GRYIMG.H
CcGrayImageInt16	16-bit Grayscale Image Object	C_GINT16.H
CcGrayImageInt32	32-bit Grayscale Image Object	C_GINT32.H
CcGrayImageFloat	Floating-Point Grayscale Image Object	C_GFLOAT.H

The methods for the Image objects, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Overlay methods** – These methods access the overlay of the image. All images can have an 8-bit overlay of the exact same size as the image itself.
- **Thresholding methods** – These methods show a given threshold range for a grayscale image in a given color. They provide visual feedback for thresholding type operations. These methods do not produce a binary image. To create a binary image from a grayscale image, see the Threshold tool's API, described in Chapter 20 starting on page 649.
- **Image allocation methods** – These methods allocate, restore, and save image data. Note that you must first allocate image data memory before you can use it.
- **Image display methods** – These methods provide display, printing, and clipboard access.

- **EZ image data access methods** – One of the most important aspects of image processing is accessing the image data. EZ access is accomplished by virtually overriding the operators ( ) and =. Using these operators, accessing the image data is easy and is independent of the type of image you are using, including color. You can access both the image data and the image overlay data using these methods.
- **Fast image data access methods** – EZ image data access is an easy way to access image data but, for large operations, it is not as fast as accessing the data directly using pointers. You can use fast image data access methods to access the image data and image overlay data directly.
- **Output look-up table methods** – Grayscale images always use an output look-up table when they are displayed. This includes 8-bit, 32-bit, and floating-point grayscale images. These methods have no effect on color images.
- **Instance methods** – It is sometimes helpful to differentiate images with similar features (such as having the same name) by using instance numbers. These methods set and get the instance numbers.
- **Point conversion methods** – When performing operations with the mouse in a window, it is sometimes necessary to obtain the location of the mouse pointer with respect to the image or to real-world coordinates. These methods convert mouse coordinates into image coordinates and image coordinates into real-world coordinates.
- **List method** – If you are writing your own application, you can use a list method to hold a list of any type of GLI/2 object. If you are writing a tool to use with GLI/2, do not use this list. It is already in use by the application.
- **Calibration methods** – Calibration methods convert pixel coordinates to real-world coordinates.

- **24-Bit RGB true color image specialized methods** – These methods are specific to the 24-bit RGB true-color image. To access these methods, the pointer must be cast to an RGB color image.
- **24-Bit HSL color image specialized methods** – These methods are specific to the 24-bit HSL color image. To access these methods, the pointer must be cast to an HSL color image.

Table 4 briefly summarizes the methods for the Image object.

Table 4: Image Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcImage( )	Constructor.
	~CcImage( )	Destructor.
Overlay Methods	CreateOverlay( )	Allocates the memory for the overlay.
	ClearOverlay( )	Sets all pixels to 0 in the overlay. A pixel of value 0 is not displayed when the overlay is displayed on the image.
	GetOverlay( )	Returns a pointer to the overlay data so that you can access it directly.You can also use the EZ image data access operators ( ) and = to access the overlay data.
	ShowOverlay( )	Displays the overlay on the image in a window.
	FreeOverlay( )	Frees the memory used for the overlay.

Table 4: Image Object Methods (cont.)

Method Type	Method Name	Method Description
Thresholding Methods	BeginThresholding( )	Begins a thresholding operation.
	ThresholdImage( )	Displays the grayscale image in the specified color for all pixels between the given threshold range.
	ThresholdImageMulti( )	Thresholds an image using multiple thresholding regions.
	EndThresholding( )	Ends a thresholding operation.
Thresholding Methods (cont.)	GetMinPixelValue( )	Returns the minimum pixel value in the entire image.
	GetMaxPixelValue( )	Returns the maximum pixel value in the entire image.
Image Allocation Methods	MakeBlankBMP( )	Allocates and initializes all pixel values to the given value. Sets the size of the image.
	OpenBMPFile( )	Opens a standard Windows BMP file from disk using the given full path name, allocates memory for the image data, and sets the size of the image.
	SaveBMPFile( )	Saves the current image data as a standard Windows BMP file to disk using the given full path name.
Image Display Methods	Show( )	Displays the image in the given window. Can show the window with different color tables and in different modes. Color images are always displayed in true 24-bit color.
	Print( )	Prints an image to the printer.
	CopyToClipboard( )	Copies the image with respect to a rectangular region to the clipboard.

Table 4: Image Object Methods (cont.)

Method Type	Method Name	Method Description
EZ Image Data Access	SetOperatorOverloadAccess( )	Determines whether to change the image data or to change the image overlay data when the operators ( ) and = are used.
	operator(x,y) and operator=	Overloaded methods.
Fast Image Data Access	SizeOf( )	Determines the size in bytes of a pixel element. Used with memmove type operations so you can have a single line of code support all image types.
	GetHeightWidth( )	Returns the height and width of the image so you can correctly calculate offsets into the image data.
	GetImageType( )	Returns the image type so you know how to handle accessing the image data directly.
	GetBitMapImageData( )	Returns a void pointer to the image data; you must cast the correct type depending on type of image you are accessing.
	ReScaleImageOnShow( )	You need to call this method if you have changed any image data so that the image is displayed correctly when it is displayed.
Output Look-up Table	GetAutoUpdateDisplay( )	Returns the mode of operation: Auto (TRUE) or manual (FALSE).
	SetAutoUpdateDisplay( )	Sets mode of operation: auto (TRUE) or manual (FALSE).
	SetDisplayLUT( )	Sets the output LUT for the image.
	GetDisplayLUT( )	Gets the output LUT for the image.

Table 4: Image Object Methods (cont.)

Method Type	Method Name	Method Description
Instance Methods	SetInstance( )	Sets the instance number for the image.
	GetInstance( )	Gets the instance number for the image.
Point Conversion	ConvertPointToImageCoords( )	Converts mouse coordinates to image coordinates.
	ConvertImagePointToWorld Coords( )	Converts Image coordinates to real-world coordinates
List Method	GetListROI( )	Returns a pointer to a List object contained in the image class.
Calibration Methods	SetCalibrationObject( )	Sets a Calibration object for use by the image.
	GetCalibrationObject( )	Returns the Calibration object being used by the image.
	ClearCalibrationObject( )	Clears the Calibration object being used by the image.
24-Bit RGB True Color Image Specialized Methods	SetAccess( )	Sets the access method of the RGB color image.
	GetAccess( )	Gets the access method of the RGB color image.
	ThresholdImageRGB( )	Thresholds the image as a true color RGB image.

Table 4: Image Object Methods (cont.)

Method Type	Method Name	Method Description
24-Bit HSL True Color Image Specialized Methods	SetAccess( )	Sets the access method of the HSL color image.
	GetAccess( )	Gets the access method of the HSL color image.
	ThresholdImageHSL( )	Thresholds the image as a true color HSL image.
	GetBitmapImageDataHSL( )	Returns a pointer to the HSL image data.
24-Bit HSL True Color Image Specialized Methods (cont.)	DoConvert( )	Converts an RGB image into HSL format inside an HSL image object.
	UpdateRGB( )	Updates the RGB display data based on the HSL data.
	SetClipping( )	Enables or disables clipping HSL values.

Constructor and Destructor Methods

This section describes the constructor and destructor methods for the Image objects.

CcImage and ~CcImage

Syntax

```
CcImage* CImage=
   new CcBinaryImage( );
//Binary
   CcImage* CImage=
   new Cc24BitRGBImage( );
//24-bit RGB color
   CcImage* CImage=
   new Cc24BitHSLImage( );
//24-bit HSL color
   CcImage* CImage=
   new CcGrayImage256( );
//8-bit grayscale
   CcImage* CImage=
   new CcGrayImageInt16( );
//16-bit grayscale
   CcImage* CImage=
   new CcGrayImageInt32( );
//32-bit grayscale
   CcImage* CImage=
   new CcGrayImageFloat( );
//floating-point grayscale
   Delete CImage;
```

Include Files

C_Binary.h, if using binary images.

C_24Bit.h, if using 24-bit RGB color images.

C_24BitHSL.h, if using 24-bit HSL color images.

C_GryImg.h, if using 8-bit grayscales.

C_Gint16.h, if using 16-bit grayscales.

C_GInt32.h, if using 32-bit grayscales.

C_GFloat.h, if using floating-point grayscales.

Description These are the standard constructor and destructor for the Image objects.

Notes The constructor and destructor for all Image objects are standard. All memory allocated by all Image objects is released when you delete the object using its base class pointer.

Overlay Methods

Often in the field of imaging it is useful to have an image overlay accompany an image to highlight some aspect of the image. Using an overlay is a nondestructive method of drawing over the image instead of the destructive method of drawing in the image.

Each Image object has its own image overlay associated with it. To use the image overlay, you must first allocate memory for the overlay. The following is the standard sequence for using an image overlay:

1. Allocate memory for the overlay by calling **CreateOverlay( )**.
2. Draw in the overlay by using the EZ image data access **operators =** and **( )** or by calling **GetOverlay( )** directly.
3. Display the overlay on the image by calling **ShowOverlay( )**.
4. Continue to update the overlay and redisplay it as needed. You can easily clear the overlay by calling **ClearOverlay( )**. When you create the overlay using **CreateOverlay( )**, the overlay is initially cleared.
5. When finished, free the memory for the overlay by calling **FreeOverlay( )**. All memory for the overlay is released when you delete the Image object; therefore, you do not have to call this method. However, if you are finished with the overlay, and you are still using the Image object, you should release the unused memory for the overlay, so as not to keep valuable system memory.

Note: The colors are shown as transparent colors. This allows you to see both the highlighted area and what is in the area at the same time.

When showing the overlay on the image, the values in the overlay determine the colors used to draw the overlay. The supported predefined constant values with corresponding colors are listed in Table 5.

Table 5: Predefined Constant Values

Pre-Defined Constant	Displayed Color
OVERLAY_CLEAR	Clear
OVERLAY_RED	Red
OVERLAY_GREEN	Green
OVERLAY_YELLOW	Yellow
OVERLAY_BLUE	Blue
OVERLAY_VIOLET	Violet
OVERLAY_CYAN	Cyan
OVERLAY_WHITE	White

The overlay methods are described in detail in the remainder of this section.

CreateOverlay

Syntax `int CreateOverlay(void);`

Include File C_Image.h

Description Allocates memory for the image overlay.

Notes This method allocates and clears (sets to 0) the memory for the image overlay. The image overlay is an 8-bit overlay capable of holding pixel values from 0 to 255. The overlay is the exact same size (height x width) as the image itself. You must have valid image data before calling this method.

Return Values

–1 Unsuccessful.

0 Successful.

ClearOverlay

Syntax `int ClearOverlay(void);`

Include File C_Image.h

Description Sets all pixel values in the overlay of the Image object to 0.

Notes You must first create the overlay for the image using the method **CreateOverlay( )** before using any overlay methods.

Return Values

–1 Unsuccessful.

0 Successful.

GetOverlay

Syntax `BYTE* GetOverlay(void);`

Include File C_Image.h

Description Returns a direct pointer to the image overlay data.

Notes You must first create the overlay for the image using **CreateOverlay( )** before using any overlay methods.

You can access image data and image overlay data within the Image objects using either EZ image data access methods (described starting on page 55) or fast image data access methods (described starting on page 59). For more examples of how to use these methods, refer to Chapter 21 starting on page 661.

The returned pointer points to the start of the image's overlay data. This is position 0,0. To calculate an offset to any other position (x,y), use the following equation:

```
Offset = Width*Y + X;
```

where, *Width* is the width of the image, and *X* and *Y* represent the desired position within the overlay (x,y). This is called direct-access or fast image data access.

To obtain the height and width of the image (and thus the height and width of the image overlay), use the method **GetHeightWidth( )**.

Example The following sample code shows how to use the pointer returned to access the location 5,5 of the image overlay. The overlay pixel is set at this position to the value 10:

```
//Get pointer to overlay data
OverlayData=CImage->GetOverlay( );
//Get height and width
CImage->GetHeightWidth
   (&Height,&Width);
//Calculate offset to the
// desired location
Offset = Width*5 + 5;
//Set pixel at the desired
//location to blue
OverlayData[Offset]=OVERLAY_BLUE;
```

ShowOverlay

Syntax

```
int ShowOverlay(
   HWND hChildWindow,
   WORD wDisplay,
   int iHorzScrolPosition,
   int iVertScrolPosition,
   int iZoom = 1);
```

or

```
int ShowOverlay(
   HDC hMemoryDC,
   HWND hChildWindow,
   WORD wDisplay,
   int iHorzScrolPosition,
   int iVertScrolPosition,
   int iZoom = 1);
```

Include File C_Image.h

Description Draws the image overlay in the given window or in the given memory device context.

Parameters

Name: hMemoryDC

Description: Handle to a memory device context.

Name: hChildWindow

Description: The handle of the window in which to display the image and its overlay.

Name: wDisplay

Description: The display mode for the image and its overlay. This can be one of the following:

- SIZE_IMAGE_TO_WINDOW – Displays the image and its overlay by stretching the image and overlay to fit inside the window without resizing the window. The aspect ratio is lost.
- SIZE_IMAGE_AS_ACTUAL – Displays the image and its overlay in their actual sizes. The image and overlay are offset by the values given in the *iHorzScrolPosition* and *iVertScrolPosition* parameters. The aspect ratio is retained.
- iHorzScrolPosition – If you use the scrollbars to help view the image in the viewport, enter the value of the position of the horizontal scrollbar. If you are not using a horizontal scrollbar, enter 0 for this parameter.

Description (cont):

- iVertScrolPosition – If you use the scrollbars to help view the image in the viewport, enter the value of the position of the vertical scrollbar. If you are not using a vertical scrollbar, enter 0 for this parameter.

Name: iZoom

Description: The zoom factor with which you are displaying the image. The default is no zooming.

Notes

Two versions of this method are available. The first version draws the overlay directly to the given window. The second version uses the extra parameter (*hMemoryDC*) to draw the overlay into the given memory device context. Before calling either version, call **Show( )** to draw the image and then call one of these methods to draw the overlay on the image. **Show( )** also has the same parameters and should be used with the correct version of **ShowOverlay( )**. The memory device context version is given for faster drawing of the image and its overlay.

The overlay is 8 bits and can hold a value between 0 and 255. If you are using the overlay to display graphics on the screen, use only the values associated with a predefined constant. Using other values may produce strange effects when the overlay is displayed.

The colors are shown as transparent. This allows you to see both the highlighted area and what is in the area at the same time.

Notes (cont.) When showing the overlay on the image, the values in the overlay determine the colors used to draw the overlay. The supported predefined constant values with corresponding colors are as follows:

- OVERLAY_CLEAR – Clear.
- OVERLAY_RED – Red.
- OVERLAY_GREEN – Green.
- OVERLAY_YELLOW – Yellow.
- OVERLAY_BLUE – Blue.
- OVERLAY_VIOLET – Violet.
- OVERLAY_CYAN – Cyan.
- OVERLAY_WHITE – White.

Return Values

–1	Unsuccessful.
0	Successful.

FreeOverlay

Syntax `int FreeOverlay(void);`

Include File C_Image.h

Description Releases the memory being used by the image overlay.

Notes The memory for the image overlay is released automatically when the Image object is deleted. You do not need to call this method if you delete the Image object. Call this method if you no longer need the image overlay but are still using the Image object.

Return Values

–1	Unsuccessful.
0	Successful.

Thresholding Methods

This group of methods provides visual feedback. For a given threshold range or multiple threshold ranges (a range between some given low and high value), you can assign a color for all pixels in the range to be displayed. You can use this data as visual feedback while determining proper low and high threshold values before creating a binary image in a thresholding operation. You can also use this data in other processes that need to show how pixels are grouped within an image.

These methods apply to all images. The color Image objects have extra methods for true RGB and HSL thresholding.

To threshold an image into a binary image, you can use the Threshold API provided with the Threshold tool. The Threshold tool uses these methods to allow you to visually select proper thresholding values before creating a binary image. The Threshold tool then uses its own API to actually create a binary image. Refer to Chapter 20 starting on page 649 for more information on the Threshold tool.

Note: The thresholding methods use the linear RGB color table (CTABLE_TO_LINR_RGB) while thresholding an image. This also applies to HSL images, which use the RGB data internally for display purposes.

The color tables are described in Table 6.

Table 6: Color Tables Used By An Image Object

Color Table	Description	Usage
CTABLE_TO_ORIG_RGB	The bitmap's original RGB 256-color table.	Used to view a bitmap opened from disk (not created with GLI/2) in its original state.
CTABLE_TO_LINR_RGB	A linear 256-color RGB color table with all entries in the color table set to grayscale values.	Used to view an RGB or HSL color image, or a grayscale image using false coloring. USED FOR THRESHOLDING
CTABLE_TO_INDEXED256	A linear indexed 256-color grayscale color table. Provides faster screen painting than the RGB equivalent.	Used to view a grayscale image in its highest resolution. Cannot produce false coloring. Not that using this color table may not actually give you a visible enhancement over using the default 64 grayscale color table due to the human visual system and your video board.
CTABLE_TO_INDEXED128	A linear indexed 128-color grayscale color table.	Used to view a grayscale image in its second highest resolution. Cannot produce false coloring. Note that using this color table may not actually give you a visible enhancement over using the default 64 grayscale color table due to the human visual system and your video board.

Table 6: Color Tables Used By An Image Object (cont.)

Color Table	Description	Usage
CTABLE_TO_INDEXED064	A linear indexed 64 color grayscale color table (the default).	Used to view a grayscale image in its lowest resolution. This is the default grayscale color table used in the GLI/2 main application. Using only 64 shades of gray to display the grayscale image leaves more colors to display other images more accurately. Using a color table with more than 64 colors usually does not enhance the image's appearance due to the human visual system and your hardware's limitations.

The standard sequence of events for thresholding an image is as follows:

1. Begin a thresholding operation by calling **BeginThresholding( )**.
2. Make sure you are displaying the image using the CTABLE_TO_LINR_RGB color table (for both RGB and HSL images) or you will not see the result of the thresholding.
3. Threshold the image, using different low and high values, by calling **ThresholdImage( )**, **ThresholdImageMulti( )**, **ThresholdImageRGB( )**, or **ThresholdImageHSL( )** multiple times.

Note: **ThresholdImageRGB( )** is specific to 24-bit RGB color images; **ThresholdImageHSL( )** is specific to 24-bit HSL color images.

---

4. After determining the best low and high values, stop thresholding the image by calling **EndThresholding( )**.

5. If needed, create a binary image (an image containing only background or foreground pixels) by using the Threshold tool's API.

Note: For more information on color tables and palette animation, see the Windows NT Win32 Programmer's Reference Volumes 1 and 2.

To view an image using all of the color tables, you can use the menu item **Display | Grayscale Color Mode** in the GLI/2 main application. For more information, refer to the *GLOBAL LAB Image/2 User's Manual*.

The thresholding methods in the main application are described in detail in the remainder of this section.

BeginThresholding

Syntax

```
int BeginThresholding(
  HWND hChildWindow,
  WORD wPalette);
```

Include File C_Image.h

Description Begins a thresholding procedure for the image.

Parameters

Name: hChildWindow

Description: Handle of the window in which you want the image to be displayed while it is being thresholded.

Name: wPalette

Description: Palette/color table to use while thresholding the image. It must be set to CTABLE_TO_LINR_RGB.

Notes This method starts the operation of thresholding an image only. No visual feedback is given at this point. You must call **ThresholdImage( )** and **Show( )** before any visual feedback occurs.

Make sure you are displaying the image using the CTABLE_TO_LINR_RGB color table or you will not see the result of the thresholding.

The CTABLE_TO_LINR_RGB color table applies to both RGB and HSL color images.

Return Values

–1 Unsuccessful.

0 Successful.

ThresholdImage

Syntax

```
int ThresholdImage(
  HWND hChildWindow,
  float fLOThresholdValue,
  float fHIThresholdValue,
  int iRed,
  int iGreen,
  int iBlue);
```

Include File C_Image.h

Description Sets all pixels between or equal to the given low and high threshold values to the specified color.

Parameters

Name: hChildWindow

Description: Handle of the window in which you want the image to be displayed in while it is being thresholded. This is the same handle that you used in **BeginThresholding( )**.

Name: fLOThresholdValue

Description: Low value in threshold range.

Name: fHIThresholdValue

Description: High value in threshold range.

Name: iRed

Description: Red portion of RGB color in which to display the threshold.

Name: iGreen

Description: Green portion of RGB color in which to display the threshold.

Name: iBlue

Description: Blue portion of RGB color in which to display the threshold.

Notes When thresholding an 8-bit grayscale image, you have exact visual feedback. If the values in the image are not a 1-to-1 match with the color table, the image is linearly interpolated to best show the thresholding. This is the case for thresholding over an 8-bit image.

Notes (cont.) When thresholding a 16-bit, 32-bit, or floating-point image, the data is always linearly interpolated to best show the thresholding. For more information, see the Threshold tool, described in Chapter 20 starting on page 649.

The low and high values in the range are inclusive (low <= range <= high).

Return Values

–1	Unsuccessful.
0	Successful.

ThresholdImageMulti

Syntax

```
int ThresholdImageMulti(
  HWND hChildWindow,
  STTHRESHOLD* stThreshold,
  int iNumberOfRegions);
```

Include File C_Image.h

Description Sets all pixels between or equal to the given low and high threshold values to the specified color.

Parameters

Name:	hChildWindow
Description:	Handle of the window in which you want the image to be displayed while it is being thresholded. This is the same handle that you used in **BeginThresholding( )**.
Name:	stThreshold
Description:	Pointer to an array of thresholding structures.

Name: iNumberOfRegions

Description: Number of thresholding structures in the *stThreshold* array.

Notes

Use this method to threshold an image with multiple thresholding regions. If you have only one region, use the simpler **ThresholdImage( )**. You can have as many regions as you like. Each region can have a different color associated with it. Each region works the same way as a single region used with **ThresholdImage( )**.

When thresholding an 8-bit grayscale image, you have exact visual feedback. If the values in the image are not a 1-to-1 match with the color table, the image is linearly interpolated to best show the thresholding. This is the case for thresholding over an 8-bit image.

When thresholding a 16-bit, 32-bit, or floating-point image, the data is always linearly interpolated to best show the thresholding. For more information, see the Threshold tool, described in Chapter 20 starting on page 649.

The low and high values in the range are inclusive (low <= range <= high).

The thresholding structure is described as follows:

```
struct STTHRESHOLD {
//Low Limit of thresholding region
float fLOThresholdValue;
//High Limit of thresholding
//region
float fHIThresholdValue;
```

Notes (cont.)

```
//Color of this region
int iRed;
int iGreen;
int iBlue;
}
```


Return Values

–1 Unsuccessful.

0 Successful.

EndThresholding

Syntax `int EndThresholding(void);`

Include File C_Image.h

Description Ends a thresholding process.

Notes When you end a thresholding process, the color table is NOT reset to that of a grayscale image so that you can view the image data using this threshold color information later. If you wish to reset the color table, you must reset it yourself while in the thresholding stage.

Return Values

–1 Unsuccessful.

0 Successful.

GetMinPixelValue

Syntax `float GetMinPixelValue(void);`

Include File C_Image.h

Description Returns the minimum pixel value contained in the entire image.

Notes This method is useful for setting initial thresholding limits. It searches the entire image for the minimum pixel value in the image.

Return Values

–1	Unsuccessful.
Minimum pixel value.	Successful.

GetMaxPixelValue

Syntax `float GetMaxPixelValue(void);`

Include File C_Image.h

Description Returns the maximum pixel value contained in the entire image.

Notes This method is useful for setting initial thresholding limits. It searches the entire image for the maximum pixel value in the image.

Return Values

–1	Unsuccessful.
Maximum pixel value.	Successful.

Image Allocation Methods

When you create a new Image object with the new operator, the memory for the image data itself is not allocated because the Image object does not know where to obtain the image data or its dimensions. Another reason for not allocating the image data at this time is that image data can come from a wide variety of places: file I/O, imaging boards, serial I/O, parallel I/O, and so on.

Two methods allocate memory for the image data: **OpenBMPFile( )** and **MakeBlankBMP( )**. You can call these methods multiple times and you can intermix them while using the same instance of an Image object. **OpenBMPFile( )** is a dedicated method for opening a standard noncompressed Windows bitmap file from disk. **MakeBlankBMP( )** is a generic method that allocates memory for the image data. You can then retrieve the data for the image from any source and copy it into the image data using direct pointer access.

The memory for the image data is handled completely by the Image object. If you called **OpenBMPFile( )** for an image of dimension 512x512 and then called **MakeBlankBMP( )** for an image of dimension 640x480, the Image object would free and reallocate all necessary memory for you.

SaveBMPFile( ) is described here because it best fits into this group.

MakeBlankBMP

Syntax

```
int MakeBlankBMP(
  int iNewHeight,
  int iNewWidth,
  int iNewColor,
  char* cNewName;
```

Include File C_Image.h

Description Allocates memory for the image data and sets all pixels in the image to the given value.

Parameters

Name:	iNewHeight
Description:	Desired height for the new image in pixels.
Name:	iNewWidth
Description:	Desired width for the new image in pixels.
Name:	iNewColor
Description:	Initializing value given to all pixels in the new image.
Name:	cNewName
Description:	Name given to the Image object. If you do not need to name your object, enter “” for its name.

Notes

This method is normally used to allocate blank memory before storing incoming image data from some source into the allocated memory. The fastest way to transfer the incoming image data into this memory is by using a direct pointer. You can obtain a direct pointer to the memory using the method **GetBitMapImageData( )**.

The memory for the image data is released when the Image object is deleted.

Return Values

–1	Unsuccessful.
Maximum pixel value.	Successful.

OpenBMPFile

Syntax `int OpenBMPFile(char *cFileName);`

Include File C_Image.h

Description Opens a standard Windows bitmap file from disk.

Parameters

Name: cFileName

Description: The full path name of the image file to open.

Notes This method first allocates all needed image memory before it opens the file from disk. The file must be a standard 256 color noncompressed Windows bitmap file for grayscale images. For 24-bit color images, the file must be a standard 24-bit true color Windows bitmap.

Return Values

–1 Unsuccessful.

Maximum pixel value. Successful.

SaveBMPFile

Syntax `int SaveBMPFile(char *cFileName);`

Include File C_Image.h

Description Saves the image as a standard Windows bitmap file.

Parameters

Name: cFileName

Description: The full path name for the file.

Notes The image data is saved as a standard 256 color noncompressed Windows bitmap file for grayscale images. For 24-bit color images, the image data is saved as a standard 24-bit true-color Windows bitmap file.

Return Values

–1	Unsuccessful.
Maximum pixel value.	Successful.

Image Display Methods

Image display methods deal with displaying the image to the screen, printing the image to the printer, or copying the image to the Windows clipboard.

When showing the image in a window, you can use any of the image's color tables. You can also show the image in its actual size or stretch the image to fit within the viewport. You can also display the same image in multiple windows, each using a different color table and a different display mode.

Show

Syntax

```
int Show(
  HWND hChildWindow,
  WORD wPalette,
  WORD wDisplay,
  int iHorzScrolPosition,
  int iVertScrolPosition,
  int iZoom = 1);
```

```
int Show(
  HDC hMemoryDC,
  HWND hChildWindow,
  WORD wPalette,
  WORD wDisplay,
  int iHorzScrolPosition,
  int iVertScrolPosition,
  int iZoom = 1);
```

Include File C_Image.h

Description Displays the image in the given window.

Parameters

Name: hMemoryDC

Description: Handle to a memory device context.

Name: hChildWindow

Description: The handle of the window in which you want to show the image.

Name: wPalette

Description: Color table / palette to use when showing the image. It can be one of the following:

- CTABLE_TO_ORIG_RGB – Original 256-color RGB color table.
- CTABLE_TO_LINR_RGB – 256-color linear RGBL color table, which applies to both RGB and HSL color images.
- CTABLE_TO_INDEXED256 – Linear indexed 256-color grayscale color table.
- CTABLE_TO_INDEXED128 – Linear indexed 128-color grayscale color table.
- CTABLE_TO_INDEXED064 – Linear indexed 64-color grayscale color table.

Name: wDisplay

Description: The display mode for the image. It can be one of the following:

- SIZE_IMAGE_TO_WINDOW – Displays the image by stretching it to fit in the current size of the window. The aspect ratio of image is lost.
- SIZE_IMAGE_AS_ACTUAL – Displays the image in its actual size. The aspect ratio is kept.

Name: iHorzScrolPosition

Description: If you are using scrollbars to position the image, enter the position of the horizontal scrollbar. If you are not using scrollbars, enter 0.

Name: iVertScrolPosition

Description: If you are using scrollbars to position the image, enter the position of the vertical scrollbar. If you are not using scrollbars, enter 0.

Name: iZoom

Description: The zoom factor with which you are displaying the image. The default is no zooming.

Notes

This method displays the image in a window. This window can be *any* window including owner draw buttons. It is sometimes useful to show a thumbnail of an image. The Memory Images tool provides this functionality by showing the selected image in a 32x32-owner draw button. You can display an image in any window that makes sense for your application.

The same image can be shown in multiple windows at the same time using different display modes (actual size vs. stretching) and using different color tables. This is an easy way to view the same image in different ways.

The memory device context version is given for faster drawing of the image and its overlay.

Return Values

–1 Unsuccessful.

0 Successful.

Print

Syntax

```
int Print(
  HWND hChildWindow=NULL,
  WORD wPalette=CTABLE_TO_LINR_
   RGB,
  WORD wDisplay=0,
  int iHorzScrolPosition=0,
  int iVertScrolPosition=0,
  int iZoom=1);
```

Include File C_Image.h

Description Prints the image to the printer.

Parameters

Name: hChildWindow

Description: The handle of the window in which you want to show the image.

Name: wPalette

Description: The color table / palette to use when showing the image. It can be one of the following:

- CTABLE_TO_ORIG_RGB – Original 256 color RGB color table.
- CTABLE_TO_LINR_RGB – 256 color linear RGB color table, which applies to both RGB and HSL color images.
- CTABLE_TO_INDEXED256 – Linear indexed 256 color grayscale color table.
- CTABLE_TO_INDEXED128 – Linear indexed 128 color grayscale color table.
- CTABLE_TO_INDEXED064 – Linear indexed 64 color grayscale color table.

Name: wDisplay

Description: The display mode for the image. It can be one of the following:

- SIZE_IMAGE_TO_WINDOW – Display the image by stretching it to fit in the current size of the window. The aspect ratio of image is lost.
- SIZE_IMAGE_AS_ACTUAL – Display the image in its actual size. The aspect ratio is kept.

Name: iHorzScrolPosition

Description: If you are using scrollbars to position the image, enter the position of the horizontal scrollbar. If you are not using scrollbars, enter 0.

Name: iVertScrolPosition

Description: If you are using scrollbars to position the image, enter the position of the vertical scrollbar. If you are not using scrollbars, enter 0.

Name: iZoom

Description: The zoom factor with which you are displaying the image. The default is no zooming.

Notes The image is printed as large as possible while keeping its aspect ratio. If all parameters are given, the image prints exactly as shown in the given window. If the image uses an overlay, the overlay is also printed with the image.

Return Values

–1 Unsuccessful.

0 Successful.

CopyToClipboard

Syntax

```
int CopyToClipboard(
  HWND hChildWindow,
  WORD wPalette,
  RECT* stRoi=NULL);
```

Include File C_Image.h

Description Copies the image to the Windows clipboard.

Parameters

Name: hChildWindow

Description: The handle to the window in which the image is displayed.

Name: wPalette

Description: Color palette with which the image is displayed.

Name: stRoi

Description: Rectangular region of image to copy to the clipboard. If left blank or NULL, the entire image is copied.

Notes Clipboard access is limited to copy. This functionality is provided so that you can copy the image into reports and such. Pasting into the Image object is not supported because floating-point and 32-bit images are supported.

Return Values

–1	Unsuccessful.
0	Successful.

EZ Image Data Access Methods

One of the most important aspects of image processing is accessing the image data. This class supports two forms of access: EZ and fast. EZ access is accomplished by virtually overriding the operators ( ) and =. Using these operators, accessing the image data is easy and is independent of the type of image you are using, including color. You can access both the image data and the image overlay data using these methods.

To set the pixel at location 25, 25 to the sum of three other images at the same location, you could use the following code (even if each of the images is of a different type):

```
Image1(25,25);
Image1 = Image2(25,25) + Image3(25,25) +
   Image4(25,25);
```

EZ access is by default set to access the image data. If you are using an overlay, you can also access the overlay data using the same code. All you need to do is tell the class which data you want to access. To access the image overlay data, you could use the following code (even if each of the images is a different type):

```
Image1.SetOperatorOverloadAccess
   (SET_ACCESS_TO_OVERLAY_DATA);
Image1(25,25);
Image1 = Image2(25,25) + Image3(25,25) +
   Image4(25,25);
```

Note: You can use subpixel accuracy or pixel accuracy. If you supply a floating-point number, subpixel accuracy is used. If you supply an integer, pixel accuracy is used.

Example:

Subpixel: `Image2(25.5,30.3)`
Pixel: `Image2(25,25)`

SetOperatorOverloadAccess

Syntax

```
int SetOperatorOverloadAccess(
  int iAccess);
```

Include File C_Image.h

Description Sets the mode of operation for the overloaded operators ( ) and =.

Parameters

Name: iAccess

Description: The desired mode of access, which can be one of the following:

- SET_ACCESS_TO_IMAGE_DATA – Accesses the image data when using the ( ) and = operators.
- SET_ACCESS_TO_OVERLAY_DATA – Accesses the image overlay data when using the ( ) and = operators.

Notes By default, the ( ) and = operators access the image data. If you want to access the image overlay data, call this method using the *SET_ACCESS_TO_OVERLAY_DATA* parameter. Then, when you use the ( ) and = operators, you access the image overlay data. To then access the image data, call this method again using the S*ET_ACCESS_TO_IMAGE_DATA* parameter.

Return Values

–1 Unsuccessful.

0 Successful.

operator(x,y) and operator=

Syntax

```
CcImage& Image = *CImage;
  Image(x,y);
  Image = 5;
```

Include File C_Image.h

Description Allows easy access to both image data and image overlay data with built-in error checking.

Parameters

Name: x

Description: The x-position in the image you want to access.

Name: y

Description: The y-position in the image you want to access.

Notes

By using the (x,y) and = operators, it is possible to easily access image data or image overlay data at the desired x,y location. Using these parameters is the same for all types of image data, including 24-bit color image data. Thus, you can mix and match all images using the same code.

To set the location in the image before assigning it a new value, you must first use the ( ) operator followed by an assignment operator =.

When accessing image data for a 24-bit RGB or HSL color image in this manner, you can access each plane of the image. You can also access the color image using its intensity. For further information, see **SetAccess( ),** described on page 85 for RGB images and on page 89 for HSL images, and **GetAccess( )**, described on page 86 for RGB images and on page 90 for HSL images.

You can use subpixel accuracy or pixel accuracy. If you supply a floating-point number, subpixel accuracy is used. If you supply an integer, pixel accuracy is used.

Return Values

A reference to the specified pixel location.

Successful.

Example

```
Subpixel:
Image2(25.5,30.3)

Pixel:
Image2(25,25)
```

Fast Image Data Access Methods

The EZ image data access method is an easy way to access image data, but, for large operations, it is not as fast as accessing the data directly using pointers. You can use fast image data access methods to access the image data and image overlay data directly. Although these methods behave the same way for all image types, accessing the data is done differently. You must be careful not to overrun the array boundaries and must point to the image data with the correct type of pointer.

For a detailed example of how to access image data directly, see the documentation on creating your own custom tools, described in Chapter 21. Also, GLI/2 includes an example change tool that provides all the code necessary to rebuild the entire tool both directly and using the EZ method of access. This example is located in the GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL subdirectory, by default.

GetBitmapImageData

Syntax `VOID* GetBitMapImageData(void);`

Include File C_Image.h

Description Returns a pointer to the image data.

Notes To obtain the height and width of the image, use the method **GetHeightWidth( )**.

This returns a pointer to the image data contained in the Image object. The pointer returned is a VOID pointer that you must cast to the correct type of pointer before accessing the image data.

Notes

The following describes what to cast the pointer to depending on the type of image you are accessing:

- Binary – BYTE*
- 8-bit grayscale image – BYTE*
- 16-bit grayscale image – unsigned short*
- 32-bit grayscale image –int*
- Floating point grayscale image – float*
- 24-bit RGB color image – RGBTRIPLE*
- 24-bit HSL color image – RGBTRIPLE*; pointer to the RGB part of the HSL image object (see also **GetBitmapImageDataHSL**())

You can determine the type of image by calling **GetImageType( )**.

The pointer returned points to the start of the image data. This is position 0,0. To calculate the offset to any other position (x,y), use the following equation:

```
Offset = Width*Y + X;
```

where, *Width* is the width of the image, and *X* and *Y* represent the desired position within the image (X,Y).

Return Values

NULL	Unsuccessful.
A pointer to the image data.	Successful.

Example This is a small pseudo-code example that shows how to use the pointer returned to access the location 5,5 of the image data. The pixel at this position is set to 10:

```
//Get pointer to image data
ImageData=CImage->
   GetBitMapImageData( );
//Get height and width
CImage->GetHeightWidth(
   &Height,&Width);
//Calculate offset to desired
//location
Offset = Width*5 + 5;
//Set pixel at desired location
//to 10
ImageData[Offset]=10;
```

GetHeightWidth

Syntax

```
int GetHeightWidth(
  int* iHt,
  int* iWd);
```

Include File C_Image.h

Description Retrieves the height and width of the image.

Parameters

Name: iHt

Description: A pointer to an integer value that accepts the height of the image.

Name: iWd

Description: A pointer to an integer value that accepts the width of the image.

Notes Call this method to obtain the height and width of the image. Since the image overlay is always the same size as the image itself, you can also use these values for the image overlay.

Return Values

–1	Unsuccessful.
0	Successful.

GetImageType

Syntax `int GetImageType(void);`

Include File C_Image.h

Description Retrieves the image's type.

Notes If you have a method that takes base class pointers so that it can be used with any type of image and you want to access the image data directly, you need to know the type of image you are dealing with. To get the image's type, call this method. For a complete example, see the code supplied with the example change tool, located in the GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL subdirectory, by default.

Return Values

–1	Unsuccessful.
IMAGE_TYPE_BINARY	Binary image.
IMAGE_TYPE_08BIT_GS	8-bit grayscale image.
IMAGE_TYPE_16BIT_GS	16-bit grayscale image.

Return Values (cont.)

IMAGE_TYPE_32BIT_GS	32-bit grayscale image.
IMAGE_TYPE_FLOAT_GS	Floating-point grayscale image.
IMAGE_TYPE_24BIT_RGB	24-bit RGB color image.
IMAGE_TYPE_24BIT_HSL	24-bit HSL color image.

ReScaleImageOnShow

Syntax `int ReScaleImageOnShow(void);`

Include File C_Image.h

Description Instructs the Image object to rescale the image data, if necessary, before showing it.

Notes When you modify the image data using the EZ data access methods, the class knows about it and automatically determines the best method of showing the image when **Show( )** is called. When you access the image data directly, the class needs to know whether anything changed so that it can show the image correctly.

If you change any image data directly, you must call this method before you call **Show( )** or the image is not displayed correctly.

Return Values

–1	Unsuccessful.
0	Successful.

SizeOf

Syntax `int SizeOf(void);`

Include File C_Image.h

Description Returns the size in bytes of a single pixel element.

Notes When writing methods that handle any image types that access the image data directly, it is sometimes necessary to move large amounts of the image data using the C function **memmove**. When using these types of functions, it is necessary to supply the amount of data you want moved, in bytes. You could accomplish this by first getting the image type and then calling one of four separate **memmove** functions that would handle the situation properly. Instead, you can use this method to supply the needed information.

This method ensures that existing methods work in the future with all new image types.

Return Values

–1	Unsuccessful.
Returns the size of a pixel element in bytes.	Successful.

Example Suppose you want to copy Image1's data into Image2, assuming they are the same type and size of image.

You could write the following code:

```
int FastCopy(CcImage* CImage1,
   CcImage* CImage2)
{
int Height1,Height2,Width1,Width2;
```

Example (cont.)

```
//Are they the same type of image
if(CImage1->GetImageType( ) !=
   CImage2->GetImageType( ) )
   return(-1);
//Get size of images
CImage1->GetHeightWidth(
   &Height1,&Width1);
CImage2->GetHeightWidth(
   &Height2,&Width2);
if(Height1 != Height2) return(-1);
if(Width1 != Width2) return(-1);
//Copy data from Image 1 into
//Image 2
memmove(CImage2->
   GetBitMapImageData( ),
CImage1->GetBitMapImageData( ),
   Height1*Width1* CImage1->
   SizeOf( ));
return(0);
}
```


Output Look-Up Table Methods

Output look-up table (LUT) methods are provided for grayscale images only. If called for a color image, these methods return –1.

Grayscale images always use an output LUT when being displayed**.** The output LUT is simply the color table you are using to display the image. This includes 32-bit and floating-point grayscale images.

You can also view an output LUT using a transfer curve between the actual value of the pixels and the color they are displayed as. The transfer curve can have a different number of points (256, 128, or 64) depending on the color table you are using. The y-value for all points is located between 0 and 255. This corresponds to the pixels' displayed color. The x-value for all points is evenly distributed along

the input axis for all grayscale images. The value of the x-value along the axis corresponds to the actual value of the pixel that is displayed. For example, assume that there is a point on the curve at position 5,10. This means that all pixels with a true value of 5 are displayed with a value of 10. Before they are modified, the color tables are all linear grayscale. This means that all x- and y-values are the same for all points. For example, the first five points in the 256 linear grayscale color table have the following values: 0,0 ; 1,1; 2,2; 3,3; 4,4; and so on, up to 255,255.

The output of the LUT (the y-values) is always fixed between 0 and 255 for all grayscale image types. If you use a color table with only 128 colors, the range of the points for the output is still fixed between 0 and 255 (not between 0 and 128). The number of colors in the color table (256, 128, 64) is the number of points along the transfer curve that maps the image data pixel values to output colors.

The x-positions for 8-bit grayscale images cannot be changed. For 32-bit and floating-point grayscale images, the input to the transfer curve (the x-values) can be positioned anywhere along the input axis. This repositioning can happen in one of two ways: automatically or manually. The default mode of operation is to automatically scale the input to the transfer curve to best show all pixel values when displaying the image. Finding the minimum and maximum pixel values in the entire image does this. The minimum pixel value is the minimum value on the input transfer curve; the maximum value is the maximum value on the input transfer curve. All the points between the minimum and maximum values are linearly redistributed to best show the image. When redistributing, the y-values remain the same for all points.

You can set the mode of operation to manual using the method **SetAutoUpdateDisplay( )**. You can then set the actual points for the transfer curve using **GetDisplayLUT( )** and **SetDisplayLUT( )**.

Note: The output LUT, the color table, and the transfer curve are all the same thing. In image processing terms, they are referred to as the output LUT. In Windows programming, they are referred to as the color table. In scientific terms, they are referred to as the transfer curve. Regardless of what you call it, inside the computer in Windows programming, the color table displays the image data.

GetAutoUpdateDisplay

Syntax `int GetAutoUpdateDisplay(void);`

Include File C_Image.h

Description Returns the mode of operation for setting the output LUT.

Notes Call this method only for 32-bit and floating-point grayscale images. The mode of operation for 8-bit grayscale images is always automatic since there is no need to change the input scaling for an 8-bit image.

Return Values

–1	Unsuccessful.
1	Automatic mode of operation.
0	Manual mode of operation.

SetAutoUpdateDisplay

Syntax

```
int SetAutoUpdateDisplay(
   BOOL bFlag);
```

Include File C_Image.h

Description Sets the mode of operation for setting the output LUT.

Parameters

Name: bFlag

Description: Flag for setting the mode of operation, which can be one of the following:

- TRUE – Automatic mode of operation.
- FALSE – Manual mode of operation.

Notes Call this method only for 32-bit and floating-point grayscale images. The mode of operation for 8-bit grayscale images is always automatic since there is no need to change the input scaling for an 8-bit image.

Return Values

–1 Unsuccessful

0 Successful.

GetDisplayLUT

Syntax

```
int GetDisplayLUT(
  int iColorTableTypeFlag,
  int iColorFlag,
  STPOINTS* stDisplayLUT);
```

Include File C_Image.h

Description Returns the requested output LUT.

Parameters

Name: ColorTableTypeFlag

Description: The requested color table. This value can be one of the following:

- CTABLE_TO_LINR_RGB – Linear 256 color RGB color table, which applies to both RGB and HSL color images.
- CTABLE_TO_INDEXED256 – Linear 256 color grayscale color table.
- CTABLE_TO_INDEXED128 – Linear 128 color grayscale color table.
- CTABLE_TO_INDEXED064 – Linear 64 color grayscale color table.

Name: iColorFlag

Description: The specific color transfer curve within the color table. This parameter is dependent on *iColorTableTypeFlag*. If *iColorTableTypeFlag* is set to any of the grayscale color tables, you must enter the value HL_COLOR_TABLE_ GRAYSCALE. For a *iColorTableTypeFlag* of CTABLE_TO_LINR_RGB, the flag indicates which RGB color you are requesting. It can be one of the following:

- HL_COLOR_TABLE_RED – Returns a 256 color transfer curve representing the red plane of the RGB color table.
- HL_COLOR_TABLE_GREEN – Returns a 256 color transfer curve representing the green plane of the RGB color table.

Description:
- HL_COLOR_TABLE_BLUE – Returns a 256 color transfer curve representing the blue plane of the RGB color table.

Name: stDisplayLUT

Description: Pointer to a user-allocated array of STPOINTS capable of holding the requested color table. An array of 256 STPOINTS can handle any of the color tables (such as. STPOINTS stColor[256]).

Notes

A binary image supports only the CTABLE_TO_LINR_RGB color table. This color table applies to both RGB and HSL color images.

Once you have the returned color table, you can alter the values in the color table. Once altered, you can use the altered color table with the Image object by calling **SetDisplayLUT( )**. To see the effect of the altered color table, show the image in a window by calling **Show( )**.

You cannot alter the x-locations for an 8-bit grayscale Image object because these values are always fixed between 0 and 255.You need to alter the end points only for 32-bit and floating-point images because points between the end points are always linearly interpolated to best show the image data (this is with respect to the x-axis only).

Notes (cont.) The main purpose of this method is to change the y- (the output color) value of the color table. You can set the y-values to any value between 0 and 255 for any grayscale image. If you set the values outside the range of 0 to 255, the value are clipped between 0 and 255.

Return Values

–1	Unsuccessful
0	Successful.

Example In this example, the 256 color RGB color table is changed to show all pixel values in the range of 53 to 153 with a red highlight.

```
void MakeRed(CcImage* CImage)
{
int x;
STPOINTS stColorTable[256];
//Get red portion of 256 color RGB
//color table from Image object so
//we can alter it
CImage->GetDisplayLUT(
   CTABLE_TO_LINR_RGB,
//Get the 256 RGB color table
HL_COLOR_TABLE_RED,
//Get the red plane of the RGB
// color table
   &stColorTable);
//Place the desired info in this
//array
```

Example (cont.)

```
//Alter the color table between
//input values of 53 and 153 to
//have an output color of 255.
//Remember, we are only altering
//the red plane of the overall
//RGB color table. Note how we do
//not alter the x positions of the
//color table
for(x=53; x<=153; x++)
stColortable[x].y = 255;
//Place the altered color table
//back into the Image object
CImage->SetDisplayLUT(
   CTABLE_TO_LINR_RGB,
//Get the 256 RGB color table
   HL_COLOR_TABLE_RED,
//Get the red plane of the RGB
//color table
   &stColorTable);
//New red plane for the RGB
//color table
//Redisplay image using the
//altered color table
CImage->Show(hWnd,
   CTABLE_TO_LINR_RGB,
//Show using altered color table
   SIZE_IMAGE_AS_ACTUAL,0,0);
}
```

SetDisplayLUT

Syntax

```
int SetDisplayLUT(
  int iColorTableTypeFlag,
  int iColorFlag,
  STPOINTS* stDisplayLUT);
```

Include File C_Image.h

Description Sets the requested output LUT.

Parameters

Name: iColorTableTypeFlag

Description: The color table to be set / altered. This value can be one of the following:

- CTABLE_TO_LINR_RGB – Linear 256-color RGB color table, which applies to both RGB and HSL color images.
- CTABLE_TO_INDEXED256 – Linear 256-color grayscale color table.
- CTABLE_TO_INDEXED128 – Linear 128-color grayscale color table.
- CTABLE_TO_INDEXED064 – Linear 64-color grayscale color table.

Name: iColorFlag

Description: Specific color transfer curve within the color table. This parameter is dependent on *iColorTableTypeFlag*.

If *iColorTableTypeFlag* is set to any of the grayscale color tables, you must enter HL_COLOR_TABLE_GRAYSCALE.

If *iColorTableTypeFlag* is set to CTABLE_TO_LINR_RGB, you can set one of the following values for *iColorFlag*:

- HL_COLOR_TABLE_RED – Sets the 256-color transfer curve representing the red plane of the RGB color table.

Description (cont.):

- HL_COLOR_TABLE_GREEN – Sets the 256-color transfer curve representing the green plane of the RGB color table.
- HL_COLOR_TABLE_BLUE – Sets the 256 color transfer curve representing the blue plane of the RGB color table.

Name: stDisplayLUT

Description: Pointer to a user-allocated array of STPOINTS that is holding the new color table. An array of 256 STPOINTS can handle any of the color tables (such as, STPOINTS stColor[256]).

Notes For further information and an example program, see **GetDisplayLUT( )** on page 68.

Return Values

–1	Unsuccessful
0	Successful.

Instance Methods

It is sometimes helpful to differentiate images of similar features (such as having the same name) by using instance numbers.

When two images with the same name are present in the system and in the GLI/2 main application, GLI/2 assigns unique instance values to the images. This is to help you keep track of images. If you are creating a tool to use with GLI/2, and this tool creates its own image and adds this new image to the main application's image list, you must make sure that the image you add has a unique instance in the system. You can do this by obtaining the list of images from the main application. Then, determine whether any other image in the system has the same name as your image. If you find one or more images with the same name, assign a unique instance number to your image using **SetInstance( )**. To check the instances of other images, use **GetInstance( )**.

The Image object itself makes no use of the instance number.

This section describes the instance methods in detail.

SetInstance

Syntax `int SetInstance(int iNewInstance);`

Include File C_Image.h

Description Sets the instance number for the object.

Parameters

Name: iNewInstance

Description: New instance number for the Image object.

Notes When you create an Image object using the new operator, the object has an instance value of 0. The Image object makes no use of the instance value. This is provided to help keep track of sequential images or unique images within your own application and within GLI/2.

Return Values

–1 Unsuccessful

0 Successful.

GetInstance

Syntax `int GetInstance(void);`

Include File C_Image.h

Description Returns the instance number for the object.

Parameters

Name: iNewInstance

Description: New instance number for the Image object.

Notes When you create an Image object using the new operator, the object has an instance value of 0. The Image object makes no use of the instance value. This is provided to help keep track of sequential images or unique images within your own application and within GLI/2.

Return Values

–1 Unsuccessful

Instance value. Successful.

Point Conversion Methods

GLI/2 is a GUI application. In a GUI application, you often make use of the mouse or other pointing device. Point conversion methods convert mouse coordinates into image coordinates. Mouse coordinates are sent to your Windows procedure each time you process any type of mouse action.

ConvertPointToImageCoords

Syntax

```
int ConvertPointToImageCoords(
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  POINT* stPointLogical,
  POINT* stPointImage,
  int iZoom = 1);
```

or

```
int ConvertPointToImageCoords(
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  POINT* stPointLogical,
  STPOINTS* stPointImage,
  int iZoom = 1);
```

Include File C_Image.h

Description Takes a point given in mouse coordinates and converts the point into both logical and image coordinates.

Parameters

Name: hChildWindow

Description: Handle to the window to receive the mouse message (the window in which the image is displayed).

Name: iHorzScrolPos

Description: If the image is being displayed in a window with scrollbars, specify the position of the horizontal scrollbar.

Name: iVertScrolPos

Description: If the image is being displayed in a window with scrollbars, specify the position of the vertical scrollbar.

Name: wDisplay

Description: The display mode for the image. It can be one of the following:

- SIZE_IMAGE_TO_WINDOW – Displays the image by stretching it to fit in the current size of the window. The aspect ratio of the image is lost.
- SIZE_IMAGE_AS_ACTUAL – Displays the image in its actual size. The aspect ratio is kept.

Name: stPointLogical

Description: Pointer to a POINT structure that holds the returned logical coordinates.

Name: stPointImage

Description: Pointer to a POINT or STPOINTS structure that holds the returned image coordinates.

Name: iZoom

Description: The zoom factor with which you are displaying the image, in image coordinates.

Notes Two versions of this method are provided. They differ only by the *stPointImage* parameter. If this parameter is a POINT structure, the returned coordinates are pixel-based. If this parameter is an STPOINTS structure, the returned coordinates are subpixel based.

Return Values

–1 Unsuccessful

Instance value. Successful.

ConvertImagePointToWorldCoords

Syntax

```
int ConvertImagePointToWorld
   Coords(
  STPOINTS* stPointImage,
  STPOINTS* stPointWorld);
```

Include File C_Image.h

Description Converts a point, given in image coordinates, into a point in real-world coordinates.

Parameters

Name: stPointImage

Description: Pointer to a STPOINTS structure that holds the image coordinates to be converted.

Name: stPointWorld

Description: Pointer to a STPOINTS structure to receive the real-world coordinates.

Notes This method uses the image's Calibration object to convert the points. If the Image object does not have an associated Calibration object, the points are converted using pixel coordinates. Thus, the real-world points are the same as the given image points.

Return Values

–1	Unsuccessful
Instance value.	Successful.

List Method

Two modes of operation are available for ROIs within the GLI/2 main application: the ROI can be attached to the viewport, or the ROI can be attached to the image itself. The image contains a List object, which can contain a list of ROIs to associate with the image. The image does not use this list internally; it simply contains it. If you are writing your own application, you can use this list to hold a list of any type of GLI/2 object. If you are writing a tool to use with GLI/2, do not use this list. It is already in use by the application.

If you are writing your own application and you need the Image object to keep a list of more than one thing, remember that a list is an object itself. Thus, you can have the Image object's list keep a list of other lists. Then, you can have these lists keep track of anything you wish. The levels of lists you can have is unlimited.

GetListROI

Syntax `CcList* GetListROI(void);`

Include File C_Image.h

C_List.h

Description Returns a pointer to the Image object's internal List object.

Notes This method call was designed specifically for use by the GLI/2 main application. If you are creating a tool for use with the GLI/2 main application, do not use this List object because it is already in use by the main application. If you are creating your own application, you can use this list to hold any GLI/2 derived object(s).

Return Values

NULL	Unsuccessful
A pointer to the List object.	Successful.

Calibration Methods

Calibration objects convert pixel coordinates to real-world coordinates. Image objects do not contain their own Calibration objects. Rather, they are associated with a Calibration object. Since many Image objects in the system use the same calibration, memory is not wasted. The methods described in this section are used to associate, retrieve, and unassociate a Calibration object from Image objects.

Note: Calibration objects are separate objects and are documented separately in this document. Refer to page 193 for more information on Calibration objects.

SetCalibrationObject

Syntax

```
int SetCalibrationObject(
  CcCalibration*
   NewCalibrationObject);
```

Include File C_Image.h

Description Associates the given Calibration object with the Image object.

Parameters

Name: NewCalibrationObject

Description: Pointer to the Calibration object to associate with this Image object.

Notes Calibration objects are created using information from a given image. They store the height and width of this image. If you try to associate a Calibration object with an image that is a different size (height and width), the method fails.

Return Values

–1	Unsuccessful
0	Successful.

GetCalibrationObject

Syntax

```
CcCalibration *
  GetCalibrationObject(void);
```

Include File C_Image.h

Description Retrieves the associated Calibration object for this image if it has one.

Notes If an Image object has no Calibration object associated with it, this method returns NULL.

Return Values

NULL	Unsuccessful
A pointer to the image's Calibration object.	Successful.

ClearCalibrationObject

Syntax

```
int ClearCalibrationObject(void);
```

Include File C_Image.h

Description Disassociates any Calibration object that is associated with this image.

Return Values

–1	Unsuccessful
0	Successful.

24-Bit RGB Specialized Methods

Out of all the methods for the Image object, only **ThresholdImageRGB( )** is specific to 24-bit color RGB Image objects. In addition, the **SetAccess( )** and **GetAccess( )** methods, which are also used for HSL Image objects, are used for 24-bit RGB Image objects. To access these methods for RGB images, the pointer to the color Image object must be of the type 24-bit RGB color.

For example, if you are sent a base class image pointer, you must cast this pointer before you can access these methods. The following examples show legal and illegal method access:

```
void SomeFunction(CcImage* CImage)
{
```

Legal:

```
Cc24BitRGBImage* C24BitColor =
(Cc24BitRGBImage*)CImage;
C24BitColor->SetAccess(RGB_ACCESS_RED);
```

Legal:

```
((Cc24BitRGBImage*)CImage)->SetAccess(
RGB_ACCESS_RED);
```

Illegal:

```
CImage->SetAccess(RGB_ACCESS_RED);
}
```

When you access the image data for a color image using the EZ data access operators ( ) and =, you can access the red, green, and blue planes of the RGB color image. You can also access the color image using its luminance (brightness) value, which is calculated as:
`luminance = 0.299*R + 0.587*G + 0.114*B;`

This section describes the RGB specialized methods in detail.

SetAccess

Syntax `int SetAccess(int iType);`

Include File C_Image.h

Description Sets the image data access mode for all RGB color images.

Parameters

Name: iType

Description: Specify the type of access into the color image's data. It can be one of the following:

- RGB_ACCESS_LUM – Access the image data by calculating its luminance value (the default).
- RGB_ACCESS_RED – Access the image data by accessing its red plane.
- RGB_ACCESS_GRN – Access the image data by accessing its green plane.
- RGB_ACCESS_BLU – Access the image data by accessing its blue plane.

Notes This method sets a static flag in the color image. When you call this method to set its access, you are setting the access for all color images in the entire application.

Return Values

–1	Unsuccessful
0	Successful.

GetAccess

Syntax

```
int GetAccess(void);
```

Include File C_Image.h

Description Gets the image data access mode for all RGB color images.

Return Values

–1	Unsuccessful
RGB_ACCESS_LUM	Accesses the image data by calculating its luminance value.
RGB_ACCESS_RED	Accesses the image data by accessing its red plane.
RGB_ACCESS_GRN	Accesses the image data by accessing its green plane.
RGB_ACCESS_BLU	Accesses the image data by accessing its blue plane.

ThresholdImageRGB

Syntax

```
int ThresholdImageRGB(
  HWND hChildWindow,
  int iRedMin,
  int iRedMax,
  int iGreenMin,
  int iGreenMax,
  int iBlueMin,
  int iBlueMax,
  int iRed,
  int iGreen,
  int iBlue);
```

Include File C_Image.h

Description Sets all pixels between or equal to the given low and high threshold values for the specified color.

Parameters

Name: hChildWindow

Description: The handle of the window in which you want to display the image while it is being thresholded. This is the same handle that you used in **BeginThresholding( )**.

Name: iRedMin

Description: Low value in the red threshold range.

Name: iRedMax

Description: High value in the red threshold range.

Name: iGreenMin

Description: Low value in the green threshold range.

Name: iGreenMax

Description: High value in the green threshold range.

Name: iBlueMin

Description: Low value in the blue threshold range.

Name: iBlueMax

Description: High value in the blue threshold range.

Name: iRed

Description: Red portion of the RGB color in which to display the threshold.

Name: iGreen

Description: Green portion of the RGB color in which to display the threshold.

Name: iBlue

Description: Blue portion of the RGB color in which to display the threshold.

Notes When thresholding a 24-bit color image, you may want to threshold all three color planes at once. This method takes into account all three color planes of the RGB image at once. If all three limits for the given pixel are between the associated thresholding limits then the pixel is shown in the given *iRed*, *iGreen*, and *iBlue* colors.

For more information, see the Threshold tool, in Chapter 20 starting on page 649.

The low and high values in the range are inclusive (low <= range <= high).

Return Values

–1 Unsuccessful.

0 Successful.

24-Bit HSL Specialized Methods

Out of all the methods for the Image object, **ThresholdImageHSL( )**, **GetBitmapImageDataHSL( )**, **DoConvert( )**, **UpdateRGB( )**, and **SetClipping( )** are specific to 24-bit HSL Image objects. In addition, the **SetAccess( )** and **GetAccess( )** methods, which are also used for RGB Image objects, are used for 24-bit HSL Image objects. To access these methods for HSL images, the pointer to the color Image object must be of the type 24-bit HSL color.

For example, if you are sent a base class image pointer, you must cast this pointer before you can access these methods. The following examples show legal and illegal method access:

```
void SomeFunction(CcImage* CImage)
{
```

Legal:

```
Cc24BitHSLImage* C24BitColorHSL =
(Cc24BitHSLImage*)CImage;
C24BitColor->SetAccess(HSL_ACCESS_HUE);
```

Legal:

```
((Cc24BitHSLImage*)CImage)->SetAccess(
HSL_ACCESS_HUE);
```

Illegal:

```
CImage->SetAccess(HSL_ACCESS_HUE);
}
```

When you access the image data for a color image using the EZ data access operators ( ) and =, you can access the hue, saturation, and luminance planes of the HSL color image.

This section describes the HSL specialized methods in detail.

SetAccess

Syntax `int SetAccess(int iType);`

Include File C_Image.h, C_24BitHSL.h

Description Sets the image data access mode for HSL color images.

Parameters

Name: iType

Description: Specify the type of access into the color image's data. It can be one of the following:

- HSL_ACCESS_HUE – Access the HSL image data by calculating its hue plane.
- RGB_ACCESS_SAT – Access the HSL image data by accessing its saturation plane.
- RGB_ACCESS_GRN – Access the HSL image data by accessing its luminance plane.

Return Values

–1 Unsuccessful

0 Successful.

GetAccess

Syntax `int GetAccess(void);`

Include File C_Image.h, C_24BitHSL.h

Description Gets the image data access mode for HSL color images.

Return Values

–1 Unsuccessful

HSL_ACCESS_HUE Accesses the HSL image data by accessing its hue plane.

HSL_ACCESS_SAT Accesses the HSL image data by accessing its saturation plane.

Return Values (cont.)

HSL_ACCESS_LUM	Accesses the HSL image data by accessing its luminance plane.

ThresholdImageHSL

Syntax

```
int ThresholdImageHSL(
  HWND hChildWindow,
  int iHueMin,
  int iHueMax,
  int iSatMin,
  int iSatMax,
  int iLumMin,
  int iLumMax,
  int iRed,
  int iGreen,
  int iBlue);
```

Include File C_Image.h

Description Sets all pixels between or equal to the given low and high threshold values for the specified color.

Parameters

Name: hChildWindow

Description: The handle of the window in which you want to display the image while it is being thresholded. This is the same handle that you used in **BeginThresholding( )**.

Name: iHueMin

Description: Low value in the hue threshold range.

Name: iHueMax

Description: High value in the hue threshold range.

Name: iSatMin

Description: Low value in the saturation threshold range.

Name: iSatMax

Description: High value in the saturation threshold range.

Name: iLumMin

Description: Low value in the luminance threshold range.

Name: iLumMax

Description: High value in the luminance threshold range.

Name: iRed

Description: Red portion of the RGB color in which to display the threshold.

Name: iGreen

Description: Green portion of the RGB color in which to display the threshold.

Name: iBlue

Description: Blue portion of the RGB color in which to display the threshold.

Notes

When thresholding a 24-bit color image, you may want to threshold all three color planes at once. This method takes into account all three color planes of the HSL image at once. If all three limits for the given pixel are between the associated thresholding limits then the pixel is shown in the given *iRed*, *iGreen*, and *iBlue* colors.

For more information, see the Threshold tool, in Chapter 20 starting on page 649.

Notes (cont.) The low and high values in the range are inclusive (low <= range <= high).

Return Values

–1	Unsuccessful.
0	Successful.

GetBitmapImageDataHSL

Syntax `VOID* GetBitMapImageDataHSL(void);`

Include File C_Image.h

Description Returns a pointer to the image data.

Notes To obtain the height and width of the image, use the method **GetHeightWidth( )**.

This returns a pointer to the HSL image data contained in the Image object. The pointer returned is a VOID pointer that you must cast to the RGBTRIPLE* type of pointer (where R corresponds to the H data, G corresponds to the S data, and B corresponds to the L data) before accessing the image data.

The pointer returned points to the start of the image data. This is position 0,0. To calculate the offset to any other position (x,y), use the following equation:

Offset = Width*Y + X;

where *Width* is the width of the image, and *X* and *Y* represent the desired portion within the image (X,Y).

Return Values

NULL	Unsuccessful.
A pointer to the image data.	Successful.

DoConvert

Syntax `int DoConvert(void);`

Include File C_Image.h, C_24BitHSL.h

Description Converts RGB data into HSL format inside the HSL Image object.

Notes This method is invoked automatically when a BMP file is loaded into an HSL object. If you modify the RGB portion of the object, invoke this method to update the HSL data.

Return Values

–1	Unsuccessful.
0	Conversion was successful.

UpdateRGB

Syntax `int UpdateRGB(void);`

Include File C_Image.h, C_24BitHSL.h

Description Updates the RGB display data based on the HSL data.

Notes If you manipulate the HSL planes, invoke this method to update the RGB data, which is used when displaying the data.

Notes (cont.) HSL values are limited to a range of 0 to 240. If you modify the HSL data and enter any values outside of this range, an error message is issued when the **UpdateRGB( )** method is invoked. If this behavior is undesired, use the **SetClipping( )** method.

Return Values

–1	Unsuccessful.
0	Successful.

SetClipping

Syntax `void SetClipping(bool bEnable);`

Include File C_Image.h, C_24BitHSL.h

Description Enables HSL data to be clipped automatically and converted into RGB values.

Parameters

Name:	bEnable
Description:	Set to TRUE to enable HSL value clipping; set to FALSE to disable HSL value clipping.

Notes HSL values are limited to a range of 0 to 240. If you modify the HSL data and enter any values outside of this range, an error message is issued when the **UpdateRGB( )** method is invoked. If this behavior is undesired, use the **SetClipping( )** method.

Return Values None

ROI Objects

An ROI object is a class that supports all the needed functionality for all ROIs in an imaging application.

In the field of imaging, different types of ROIs can be used depending on the requirements of your application. GLI/2 supplies the following ROIs:

- Point,
- Rectangular,
- Line,
- Freehand line,
- Poly line,
- Elliptical,
- Poly freehand, and
- Freehand ROIs.

All methods are virtual C methods, making them operate the same way. Thus, when writing an application, you can use the base class pointer with almost all methods. For example, when showing an ROI in a window, regardless of what type of ROI it is, you can always use the following code for the operation:

```
CROI->ShowROI( );
```

Because all ROI objects are derived from a base class ROI object, and all methods specific to a given type of ROI object are virtual, the methods are documented only once. This is because the methods behave identically for all types of ROI objects. If a method does not behave identically for all ROI object types, the method is documented with the object.

Note: The term poly refers to a many-sided (straight sides) line or freehand ROI.

The hierarchy of the ROI object classes is shown in Table 7.

Table 7: Hierarchy of the ROI Object Classes

Class Name	Description	Include File
CcHLObject	GLI/2 Base Class Object	
CcRoiBase	Virtual Base Class ROI Object	C_RBASE.H
CcRoiPoint	Point ROI	C_POINT.H
CcRoiLine	Line ROI	C_LINE.H
CcRoiPolyLine	Poly Line ROI	C_PLINE.H
CcRoiFreeHandLine	Freehand ROI	C_FLINE.H
CcRoiRect	Rectangular ROI	C_RECT.H
CcRoiEllipse	Elliptical ROI	C_ELIPSE.H
CcRoiFreeHand	Freehand ROI	C_FREE.H
CcRoiPolyFreeHand	Poly Freehand ROI	C_PFREE.H

The methods for the ROI objects, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Type method** – This method is used to determine the ROI's type.
- **Selection methods** – These methods keep track of ROI selection and selection colors.
- **Position methods** – These methods position the ROI with respect to image coordinates.

- **Mouse methods** – These methods interface the ROI to the mouse.
- **ROI display methods** – This method shows the ROI in a window.
- **ROI image access methods** – These methods return the pixel locations of the image inside or on the ROI perimeter.
- **Save and restore methods** – These methods save and restore an ROI to or from disk.
- **Graphic ROI methods** – Some ROIs are graphic ROIs. These ROIs are not part of the GLI/2 API and are not documented here. There are ROIs that also contain graphics, such as the Text ROI object used by the Text tool. The Text ROI works like an ROI but also shows text on an image and places text on an image or its overlay.

Table 8 briefly describes the methods for the ROI object.

Table 8: ROI Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcRoiBase( )	Constructor.
	CcRoiBase( )	Destructor.
Type Methods	GetROIType( )	Returns the ROI's type: rectangular, line, elliptical, or freehand.
	SetSelected( )	Selects or unselects the ROI.
	IsROISelected( )	Returns 1 if the ROI is selected or 0 if the ROI is not selected.
	SetSelectedColor( )	Sets the color used to display a selected ROI.

Table 8: ROI Object Methods (cont.)

Method Type	Method Name	Method Description
Type Methods (cont.)	SetUnSelectedColor( )	Sets the color used to display an unselected ROI.
	GetSelectedColor( )	Gets the color used to display a selected ROI.
	GetUnSelectedColor( )	Gets the color used to display an unselected ROI.
Position Methods	SetRoiImageCord( )	Returns a void pointer to a structure describing the ROI's position.
	GetRoiImageCord( )	Takes a void pointer to a structure describing the ROI's position.
Mouse Methods	StartMouseDrag( )	Starts positioning the ROI using the given mouse coordinates. This is usually called in conjunction with pressing down the left mouse button.
	DoMouseDrag( )	Redraws the position of the ROI using the new mouse coordinates. This is usually called in conjunction with dragging the mouse while holding down the left mouse button.
	StopMouseDrag ( )	Stops positioning the ROI at the given mouse coordinates. This is usually called in conjunction with releasing the left mouse button.
	GetCurrentBoundingRect( )	Returns a pointer to a RECT structure describing the bounding rectangle of the ROI. A bounding rectangle is the smallest rectangle that encompasses the entire ROI.
	MouseHitTest( )	Returns whether the given mouse coordinates are inside or on the ROI.

Table 8: ROI Object Methods (cont.)

Method Type	Method Name	Method Description
ROI Display Methods	ShowROI( )	Displays the ROI in the given window.
ROI Image Access Methods	GetBoundingRect( )	Returns the range of pixels that lie inside the ROI for the given image. You use these values as a reference for going through the entire ROI.
	GetYBoundary( )	Given a y-value, returns an array containing all the x-pixel locations inside the ROI. (Use this if you can because it is a faster method to process).
	GetXBoundary( )	Given an x-value, returns an array containing all the y-pixel locations inside the ROI.
Save and Restore Methods	Save( )	Saves an ROI to disk using a given file name.
	Restore( )	Restores an ROI from disk using a given file name.
Graphic ROI Methods	IsRoiAGraphicObject( )	Returns true if a ROI is a graphic ROI. All ROIs documented above are NOT graphic ROIs.
	UpdateImageIfNeeded( )	Updates the image with its graphics if the graphics need to be updated.

Constructor and Destructor Methods

This section describes the constructor and destructor for the ROI objects.

CcRoiBase( ) and ~CcRoiBase( )

Syntax

```
CcRoiBase* CRoi=new CcRoiPoint( );
  //Point ROI
CcRoiBase* CRoi=new CcRoiLine( );
  //Line ROI
CcRoiBase* CRoi=
   new CcRoiFreeHandLine( );
  //Freehand Line ROI
CcRoiBase* CRoi=
   new CcRoiPolyLine( );
  //Poly Line ROI
CcRoiBase * CRoi=new CcRoiRect( );
  //Rect ROI
CcRoiBase * CRoi=
   new CcRoiEllipse( );
  //Elliptical ROI
CcRoiBase * CRoi=
  new CcRoiFreeHand( );
  //Freehand ROI
CcRoiBase * CRoi=
  new CcRoiPolyFreeHand( );
  //Poly Freehand ROI
Delete CRoi;
```

Include File

C_Point.h, if using point ROIs.

C_Line.h, if using line ROIs.

C_Fline.h, if using freehand line ROIs.

C_Pline.h, if using Poly line ROIs.

C_Rect.h, if using rectangular ROIs.

C_Elipse.h, if using elliptical ROIs.

C_Free.h, if using freehand ROIs.

C_Pfree.h, if using Poly freehand ROIs.

Description These are the standard constructor and destructor for the ROI objects.

Notes All memory allocated by all ROI objects is released when the object is deleted using its base class pointer.

Type Method

This method returns the ROI's type.

GetROIType

Syntax `int GetROIType(void);`

Include File C_RBase.h

Description Returns the ROI's type

Return Values

–1	Unsuccessful.
ROI_POINT	Point ROI.
ROI_LINE	Line ROI.
ROI_FLINE	Freehand line ROI.
ROI_PLINE	Poly line ROI.
ROI_RECT	Rectangular ROI.
ROI_ELLIPSE	Elliptical ROI.
ROI_FREEHAND	Freehand ROI.
ROI_PFREEHAND	Poly freehand ROI.

Selection Methods

In the GLI/2 main application, each viewport can have only one active (selected) ROI at the same time. If you are writing a tool to use with GLI/2, do not use these methods; they are already in use by the GLI/2 main application.

In your own application, you can use these methods to select or unselect any number of ROIs at the same time. When the ROI is displayed (using the **Show( )** method), a selected ROI is displayed in the selected color (red, by default) and an unselected ROI is displayed in the unselected color (green, by default). You can override this functionality using **Show( )**.

Note: By default, the ROI is unselected and is shown using the unselected color. Thus, if you do not want to use any of this functionality, simply do nothing and all ROIs are displayed in the same color (the unselected color).

SetSelected

Syntax `int SetSelected(BOOL bSel);`

Include File C_RBase.h

Description Selects or unselects the ROI.

Name: bSel

Description: Set to TRUE to select the ROI; set to FALSE to unselect the ROI.

Return Values

–1 Unsuccessful.

0 Successful.

IsROISelected

Syntax

```
BOOL IsROISelected(void);
```

Include File C_RBase.h

Description Returns whether the ROI is selected.

Return Values

False	Unselected.
True	Selected.

SetSelectedColor

Syntax

```
int SetSelectedColor(
   RGBTRIPLE* stColor);
```

Include File C_RBase.h

Description Sets the color that is used to show a selected ROI.

Name:	RGBTRIPLE
Description:	Structure that contains the red, green, and blue colors for the selected color.

Notes The default color for the selected color is red.

Return Values

–1	Unsuccessful.
0	Successful.

SetUnSelectedColor

Syntax

```
int SetUnSelectedColor(RGBTRIPLE*
   stColor);
```

Include File C_RBase.h

Description Sets the color that is used to show an unselected ROI.

Name: RGBTRIPLE

Description: Structure that contains the red, green, and blue colors for the unselected color.

Notes The default color for the unselected color is green.

Return Values

–1 Unsuccessful.

0 Successful.

GetSelectedColor

Syntax

```
int GetSelectedColor(
   RGBTRIPLE* stColor);
```

Include File C_RBase.h

Description Gets the color that is used to show a selected ROI.

Name: RGBTRIPLE

Description: Structure that contains the red, green, and blue colors for the selected color.

Notes The default color for the selected color is red.

Return Values

–1	Unsuccessful.
0	Successful.

GetUnSelectedColor

Syntax

```
int GetUnSelectedColor(
   RGBTRIPLE* stColor);
```

Include File C_RBase.h

Description Gets the color that is used to show an unselected ROI.

Name: RGBTRIPLE

Description: Structure that contains the red, green, and blue colors for the unselected color.

Notes The default color for the unselected color is green.

Return Values

–1	Unsuccessful.
0	Successful.

Position Methods

An ROI can be positioned using the mouse, in which case its size and position are already set, or it can be positioned by calling the position methods. These methods use a void pointer because the ROIs differ in what type of information they need to set their positions directly. For example, you need a single point to set a point ROI, you need two points to set a RECT ROI, and you need several points to set a freehand ROI. You can always determine an ROI's type by calling the method **GetROIType( )**.

This section describes the position methods in detail.

SetRoiImageCord

Syntax

```
int SetRoiImageCord
   VOID* stROI);
```

Include File C_RBase.h

DT_Str.h

Description Sets the position of the ROI in image coordinates.

Name: stROI

Description: A void pointer to a structure that describes the perimeter of the ROI. It can be one of the following types:

- Point – STPOINTS structure (STPOINTS*) describing the x,y-position of the point; it can be subpixel.
- Rect – Rectangle structure (RECT*) that describes the bounding rectangle for the ROI.
- Line – Rectangle structure (RECT*) that describes the line for the ROI.
- Poly Line – Structure (PIXELGROUPING*) that describes each point on the line of the ROI.
- Freehand Line – Structure (PIXELGROUPING*) that describes each point on the line of the ROI.
- Ellipse – Rectangle structure (RECT*) that describes the ellipse for the ROI.

Description (cont):

- Freehand – Structure (PIXELGROUPING*) that describes each point on the perimeter of the ROI.
- Poly freehand – Structure (PIXELGROUPING*) that describes each point on the perimeter of the ROI.

Notes

The line, rectangular, and elliptical ROIs take a Windows RECT structure to describe their position and size. The freehand ROI takes a GLI/2 defined PIXELGROUPING structure, defined as follows:

```
struct PixelGroupTag {
  int iRed,iGreen,iBlue;
  int iNumOfPoints;
  POINT *stPOINTS;
  HGLOBAL hstPOINTS;
};
typedef struct PixelGroupTag
   PIXELGROUPING;
```

The *iRed, iGreen,* and *iBlue* variables are not used and should be set to 0. Set the total number of points in the perimeter of the freehand ROI in the variable *iNumOfPoints*. The actual points are contained in the array of POINT structures, *stPOINTS*. Allocate the memory with the SDK function **GlobalAlloc( )**. Store the handle to the memory in the *hstPOINTS* variable.

Notes (cont.) The following is an example showing how to allocate the memory:

```
(PIXELGROUPING stP):
stP.hstPOINTS = GlobalAlloc(
   GHND,500*sizeof(POINT));
stp.stPOINTS = (POINT*)
    GlobalLock(stP.hstPOINTS);
```

The freehand and poly Freehand ROIs are enclosed ROIs. The last point in the array should not be the same as the first point in the array. The ROI object draws them connected, by default. The poly line and freehand line ROIs are not enclosed ROIs, but still take the same PIXELGROUPING structure.

Each point in any freehand, poly freehand, freehand line, or poly line ROI must be eight-connected and must not touch any other points.

Return Values

–1	Unsuccessful.
0	Successful.

GetRoiImageCord

Syntax `VOID* GetRoiImageCord(void);`

Include File C_RBase.h

DT_Str.h

Description Gets the position of the ROI in image coordinates.

Notes For more information on the returned structures, refer to **SetRoiImageCord( )** on page 107.

Return Values

Point	STPOINTS structure (STPOINTS*) that describes the x,y-position of the point; it can be subpixel.
Rect	Rectangle structure (RECT*) that describes the bounding rectangle for the ROI.
Line	Rectangle structure (RECT*) that describes the line for the ROI.
Freehand Line	Structure (PIXELGROUPING*) that describes the line for the ROI.
Poly Line	Structure (PIXELGROUPING*) that describes each point on the line of the ROI.
Ellipse	Rectangle structure (RECT*) that describes the ellipse for the ROI.
Freehand	Structure (PIXELGROUPING*) that describes each point on the perimeter of the ROI.
Poly Freehand	Structure (PIXELGROUPING*) that describes each point on the perimeter of the ROI.

Mouse Methods

Almost all interaction for a ROI is provided using the mouse in an imaging application. This includes creating, selecting, deleting, moving, copying, resizing, and testing ROIs.

ROI Creation

ROI creation is supported using the following methods:

- **StartMouseDrag( )**;
- **DoMouseDrag( )**; and
- **EndMouseDrag( )**.

In most applications, an ROI is created using a left-button-down, mouse drag, left-button-up sequence. Accompanying this might be a key sequence before the action is invoked. GLI/2 uses the left-button-down key sequence with a SHIFT + CRTL key sequence before invoking the mouse creation methods. Choose the key sequence that works best for your application.

The step-by-step process is as follows:

1. Create the desired ROI type with the new operator, as follows:

   ```
   CcRoiBase* CRoi = new CcRoiRect( );
   ```

2. Using the returned pointer, *CRoi*, begin the visual feedback by calling the start method with the initial mouse coordinates. The mouse coordinates are sent with each mouse message:

   ```
   CRoi->StartMouseDrag( );
   ```

3. Capture the WM_MOUSEMOVE message sent every time you move (drag) the mouse by calling the **DoMouseDrag** method with the new mouse coordinates:

   ```
   CRoi->DoMouseDrag( );
   ```

4. When you end the drag by lifting the depressed mouse button, end the visual feedback by sending the **StopMouseDrag( )** method:

   ```
   CRoi->StopMouseDrag( );
   ```

At any time during the process, you can call **GetCurrentBoundingRect( )** (not **GetBoundingRect( )**) to retrieve the current bounding rectangle of the ROI in image coordinates.

ROI Selection and Deletion

To select or delete an ROI, you need to know if the correct sequence for the mouse within or on the ROI has been performed. To determine if the mouse is in the ROI, call **MouseHitTest( )** with the current mouse coordinates.

ROI Moving and Copying

This procedure is similar to the creation of the ROI. The only difference is that you send different flags to **StartMouseDrag( )**.

Note: One extra parameter is required for the poly line and poly freehand ROIs when calling **DoMouseDrag( )**.

StartMouseDrag

Syntax

```
int StartMouseDrag(
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  POINT stMousePos,
  int iDrawingMode,
  CcRoiBase* COrigRoi,
  int iZoom = 1);
```

Include File C_RBase.h

Description Starts the visual feedback for an ROI create, move, copy, or resize operation.

Parameters

Name: hChildWindow

Description: Handle to the window in which you are performing the operation.

Name: iHorzScrolPos

Description: If using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI. It can be one of the following:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: Pointer to the CcImage object on which you are drawing the ROI.

Name: stMousePos

Description: Position of the mouse in mouse coordinates; this is sent to you along with the mouse message.

Name: iDrawingMode

Description: The mouse operation you are starting. It can be one of the following:

- ROI_MODE_NEW – Creates a new ROI.
- ROI_MODE_MOVE – Moves an existing ROI.
- ROI_MODE_COPY – Creates a new ROI by copying an existing ROI.
- ROI_MODE_SIZE – Resizes an existing ROI. Only supported for line, rectangle, and ellipse ROIs.

Name: COrigRoi

Description: Enter a pointer to the ROI that you are copying if the *iDrawingMode* parameter is ROI_MODE_COPY; otherwise, enter NULL.

Name: iZoom

Description: The zoom factor with which you are displaying the image.

Notes Because the ROI can be drawn on grayscale and color images, the ROI provides visual feedback by inverting the colors in the image. If you are copying an ROI, make sure to copy the same type of ROI that you are creating.

In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

Notes (cont.) In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1	Unsuccessful.
0	Successful.

DoMouseDrag

Syntax

```
int DoMouseDrag(
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  POINT stMousePos,
  int iFlag);
```

Include File C_RBase.h

Description Provides the visual feedback as you drag the mouse for an ROI create, move, or copy operation.

Parameters

Name:	hChildWindow
Description:	Handle to the window in which you are performing the operation.
Name:	iHorzScrolPos
Description:	If you are using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise, enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI. It can be one of the following:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: A pointer to the CcImage object on which you are drawing the ROI.

Name: stMousePos

Description: The position of the mouse in mouse coordinates; this is sent to you along with the mouse message.

Name: iFlag

Description: Flag for the poly line and poly freehand ROIs. If it is not a poly ROI, enter 0. If it is a poly ROI, enter DO_MOUSE_DRAG_ADD_ BREAK_POINT to start a new line segment. Otherwise, enter 0.

Notes

Because the ROI can be drawn on grayscale and color images, the ROI provides visual feedback by inverting the colors in the image. If the ROI is a poly line or poly freehand ROI, you need to tell the ROI when to start a new line segment. To start a new line segment, enter DO_MOUSE_DRAG_ADD_BREAK_POINT for the *iFlag* parameter. If you are not starting a new line segment, enter 0. GLI/2 starts a new line segment when you release the left mouse button.

In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1 Unsuccessful.

0 Successful.

StopMouseDrag

Syntax

```
int StopMouseDrag (
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  POINT stMousePos);
```

Include File C_RBase.h

Description Ends the visual feedback for an ROI create, move, or copy operation.

Parameters

Name: hChildWindow

Description: Handle to the window in which you are performing the operation.

Name: iHorzScrolPos

Description: If using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise, enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI. It can be one of the following:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: Pointer to the CcImage object on which you are drawing the ROI.

Name: stMousePos

Description: Position of the mouse in mouse coordinates; this is sent to you along with the mouse message.

Notes Because the ROI can be drawn on grayscale and color images, the ROI provides visual feedback by inverting the colors in the image.

In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1 Unsuccessful.

0 Successful.

GetCurrentBoundingRect

Syntax

```
RECT* GetCurrentBoundingRect(
  void);
```

Include File C_RBase.h

Description Returns the bounding rectangle for an ROI while it is being created, moved, or copied.

Notes Because the ROI can be drawn on grayscale and color images, the ROI provides visual feedback by inverting the colors in the image.

Return Values

NULL	Unsuccessful.
A pointer to a RECT structure describing the bounding rectangle of the ROI.	Successful.

MouseHitTest

Syntax

```
int MouseHitTest(
  hWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  POINT stMousePos
  int iZoom = 1);
```

Include File C_RBase.h

Description Tests to see if the given mouse position is inside or on the ROI.

Parameters

Name:	hChildWindow
Description:	Handle to the window in which you are performing the operation.
Name:	iHorzScrolPos
Description:	If you are using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise, enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: A pointer to the CcImage object on which you are drawing the ROI.

Name: stMousePos

Description: The position of the mouse in mouse coordinates; this is sent to you along with the mouse message.

Name: iZoom

Description: The zoom factor with which you are displaying the image.

Notes In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1	Unsuccessful.
ROI_HIT_TEST_INSIDE	Mouse is inside the ROI.
ROI_HIT_TEST_TOP	Mouse is at the top of the ROI.
ROI_HIT_TEST_BOTTOM	Mouse is at the bottom of the ROI.
ROI_HIT_TEST_RIGHT	Mouse is on the right side of the ROI.
ROI_HIT_TEST_LEFT	Mouse is on the left side of the ROI.
ROI_HIT_TEST_UL	Mouse is on the upper-left corner of the ROI.
ROI_HIT_TEST_UR	Mouse is on the upper-right corner of the ROI.
ROI_HIT_TEST_LL	Mouse is on the lower-left corner of the ROI.
ROI_HIT_TEST_LR	Mouse is on the lower-right corner of the ROI.

ROI Display Method

This method shows the ROI in a window. It displays a selected ROI in the selected ROI color and an unselected ROI in the unselected color. You can override this functionality by forcing the color in which to display the ROI.

ShowROI

Syntax

```
int ShowROI(
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  int iZoom= -1,
  int iFlag= -1);
```

or

```
int ShowROI(
  HDC hMemoryDC,
  HWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  int iZoom = 1,
  int iFlag= -1);
```

Include File C_RBase.h

Description Shows the ROI in the given window.

Parameters

Name:	hMemoryDC
Description:	Handle to a memory device context.
Name:	hChildWindow
Description:	Handle to the window in which you want to show the ROI.
Name:	iHorzScrolPos
Description:	If you are using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise, enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: A pointer to the CcImage object on which you are drawing the ROI.

Name: iZoom

Description: The zoom factor with which you are displaying the image. The default is no zooming.

Name: iFlag = –1

Description: You can use this parameter to override the default functionality of drawing a selected ROI in the selected color and drawing an unselected ROI in the unselected color. By overriding this parameter, the class bypasses using its internally selected indicator.

Description (cont.): You can override the color in which to draw the ROI by using one of the following values:

- ROI_SELECTED – Draws the ROI in the selected color.
- ROI_NOT_SELECTED – Draws the ROI in the unselected color.

Notes

Two versions of this method are provided. The first version draws the ROI directly to the given window. The second version uses the extra parameter (*hMemoryDC*) to draw the ROI into the given memory device context. These methods should be used with the corresponding version of the Image object's **Show( )** method, described on page 49.

The memory device context version is given for faster drawing of the image and its overlay.

In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1	Unsuccessful.
0	Successful.

ROI Image Access Methods

The main purpose of an ROI is to determine the location of the desired pixels within the image to process. These methods return the pixel locations of the image that lie inside and on the ROI perimeter.

The ROI image access methods work together to supply a standard way to access the locations of the enclosed pixels within the ROI. The same code works for all types of ROIs, including freehand ROIs.

The first step is to obtain the bounding rectangle for the ROI. The bounding rectangle is the smallest rectangle that contains the ROI. Using the bounding rectangle you can go from the bottom to the top (the preferred method), processing each horizontal row along the way, or you can go from the left to the right, processing each vertical row along the way. Because of the way the memory is organized for the image, it is better to go from bottom to top.

To process all pixels encompassed by a ROI (this includes pixels on the perimeter), you can use the following code:

```
void SomeFunction( CcImage* CImage, CcRoiBase*
CRoi)
{
/*Start of Dec Section*/
 int x,y,z;
 int* piRoiData;
 int iNumOfROIPoints;
 RECT* pstROI;
 CcImage& Image = *CImage;
/*End of Dec Section*/
//Get pointer to bounding rectangle
// This code never needs to change
pstROI =(RECT*)CRoi->GetBoundingRect( );
if(pstROI == NULL) return(-1);
//Change Image Data
// This code never needs to change
for(y=pstROI->bottom; y<pstROI->top; y++)
```

```
   {

piRoiData=CRoi->GetXBoundary(y,&iNumOfROIPoints);
   if(piRoiData != NULL)
   for(z=0; z<iNumOfROIPoints; z++)
      {x=piRoiData[z];

      //Put changes here to process your custom
      //methods
      Image(x,y);
      Image=47;
      }
   }
}
```


Note: This is a code fragment from the code provided with the example change tool. All code necessary to rebuild the example change tool is given in the subdirectory GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

GetBoundingRect

Syntax `RECT* GetBoundingRect(void);`

Include File C_RBase.h

Description Returns the bounding rectangle for the ROI.

Notes The bounding rectangle is the smallest rectangle that encompasses the entire ROI.

Return Values

NULL	Unsuccessful.
The bounding rectangle.	Successful.

GetYBoundary

Syntax

```
int* GetYBoundary(
  int iXPos,
  int* iNumOfPoints);
```

Include File C_RBase.h

Description Returns all points in the ROI with a horizontal position of *iXPos*. The returned information is a vertical line, in image coordinates.

Parameters

Name: iXPos

Description: Horizontal position at which to return all vertical points within the ROI.

Name: iNumOfPoints

Description: Pointer to an integer variable that accepts the total number of points returned.

Notes For line, rectangular, and elliptical ROIs, this line is continuous. Freehand ROIs may have separations in the returned vertical line since they can take any shape.

Return Values

NULL Unsuccessful.

Returns an array of integers. Successful.

GetXBoundary

Syntax

```
int* GetXBoundary(
  int iYPos,
  int* iNumOfPoints);
```

Include File C_RBase.h

Description Returns all points in the ROI with a vertical position of *iYPos*. The returned information is a horizontal line in image coordinates.

Parameters

Name: iYPos

Description: Vertical position at which to return all horizontal points within the ROI.

Name: iNumOfPoints

Description: Pointer to an integer variable that accepts the total number of points returned.

Notes For line, rectangular, and elliptical ROIs, this line is continuous. Freehand ROIs may have separations in the returned horizontal line since they can take any shape.

This method is preferred over **GetYBoundary( )** due to the way memory is organized within the Image object for the image data. For continuous lines, you can calculate the beginning pointer and ending pointer into the image data, and then access all pixels by pointer (fast image data access) rather than using the EZ image data access operators ( ) and = . An example of this is given in the code provided with the example change tool. All necessary code to rebuild the example change tool is located in GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

Return Values

NULL Unsuccessful.

Returns an array of integers. Successful.

Save and Restore Methods

These methods save and restore an ROI to and from disk.

Save

Syntax `int Save(char* cFileName);`

Include File C_RBase.h

Description Selects or unselects the ROI.

Parameters

Name: cFileName

Description: Full path name of where to save the ROI.

Return Values

–1 Unsuccessful.

0 Successful.

Restore

Syntax

```
int Restore(
   char* cFileName);
```

Include File C_RBase.h

Description Returns whether the ROI is selected.

Parameters

Name: cFileName

Description: Full path name of the ROI to restore.

Return Values

–1	Unsuccessful.
0	Successful.

Graphic ROI Methods

Some ROIs are graphic ROIs. These ROIs are not part of the GLI/2 API and are not documented here. There are ROIs that also contain graphics, such as the Text ROI object used by the Text tool. It works like an ROI but also shows text on an image and places text on an image or its overlay. Their base class methods are documented in this section.

IsRoiAGraphicObject

Syntax `BOOL IsRoiAGraphicObject(void);`

Include File C_RBase.h

Description Returns whether this ROI object is a graphic ROI.

Return Values

False	ROI is not a graphic ROI.
True	ROI is a graphic ROI.

UpdateImageIfNeeded

Syntax

```
int UpdateImageIfNeeded(
  hWND hChildWindow,
  int iHorzScrolPos,
  int iVertScrolPos,
  WORD wDisplay,
  CcImage* CImage,
  int iFlag= -1);
```

Include File C_RBase.h

Description A graphic ROI updates the given image or overlay (if needed) when this method is called by the GLI/2 main application or by a user-defined application.

Parameters

Name: hChildWindow

Description: Handle to the window in which you want to show the ROI.

Name: iHorzScrolPos

Description: If you are using a horizontal scrollbar, enter the position of the horizontal scrollbar; otherwise, enter 0.

Name: iVertScrolPos

Description: If you are using a vertical scrollbar, enter the position of the vertical scrollbar; otherwise, enter 0.

Name: wDisplay

Description: Mode of display for the image on which you are drawing the ROI:

- SIZE_IMAGE_AS_ACTUAL – Image is shown in its actual size.
- SIZE_IMAGE_TO_WINDOW – Image is stretched to fit in the window.

Name: CImage

Description: A pointer to the CcImage object on which you are drawing the ROI.

Name: iFlag = –1

Description: You can use this variable to override the default functionality of drawing a selected ROI in the selected color and drawing an unselected ROI in the unselected color. By overriding this parameter, the class bypasses using its internally-selected indicator. You can override in which color to draw the ROI by using one of the following values:

- ROI_SELECTED – Draws the ROI in the selected color.
- ROI_NOT_SELECTED – Draws the ROI in the unselected color.

Notes This method is called by the GLI/2 main application just before it draws the image. If needed, the Graphic object updates the image data or image overlay data so that the image appears correctly.

Notes (cont.)

In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Return Values

–1	Unsuccessful.
0	Successful.

Curve Objects

A Curve object is used for accessing an array of points (a curve) so that it can be graphed easily using a Graph object. An object derived from a Base Class object is used for creating an array of points that may or may not be graphed using a Graph object.

For example, if you had an array of points that you wanted graphed using a Graph object, you could create a base class Curve object and associate the array of points with the Curve object. Once associated with the Curve object, the Curve object can be displayed on a graph using a Graph object. When using a curve base class directly, you are responsible for allocating and releasing memory for the points.

On the other hand, you might want to create a class that performs some type of calculation that derives an array of points, such as a histogram operation. In this case you would need to allocate memory for the points and then perform the calculation. To do this, you can derive a new class from the Curve base class object. This is how the GLI/2 histogram and line profile classes were created. Because the histogram class is derived from the curve class, it has all the necessary functionality built in so that it can be graphed by the graph class. The functionality that allocates and calculates the histogram data is what you add in the derived class. When creating these derived types of classes, it is the responsibility of the derived class to allocate the memory; the memory is released automatically by the base class when the object is deleted.

In GLI/2, an array of points comprises a curve. The points are contained in a GLI/2 structure named STPOINTS. STPOINTS is defined as follows:

```
struct tagPoints{
   float fX,fY;
};
typedef struct tagPoints STPOINTS;
```

This structure is just like the Windows POINT structure except that the *x* and *y* variables are floating-point. This is so the Graph object can graph floating-point data that might be produced during a complex imaging calculation.

Within the base class are three very important member variables, all of which are protected and which can be seen by looking at the header file, C_curve.h:

```
protected:
   int iNumOfPoints;
   STPOINTS* stPoints;
   HGLOBAL hstPoints;
```

stPoints is the pointer to the curve data itself (an array of points). *iNumOfPoints* is the number of points in the array. If you are using the base class directly, you can use the method **SetCurveData( )** to make the *stPoints* pointer point to your array of points and to set the correct number of points in the array. Then, you can use the class with a graph class to graph the curve. Since the class did not allocate the memory for the array of points, it does not release the memory.

If you are deriving your own class from a curve base class, such as making your own histogram class, you need to use all of these member variables. The derived class first allocates memory for the array of points, and then performs its calculation, placing the resultant data into the array of points. To do this, you must first allocate the memory and place the handle to the memory in the *hstPoints* member variable. You do this by using the SDK function **GlobalAlloc( )** as follows:

```
iNumOfPoints = 100;
hstPoints = GlobalAlloc(GHND,
   iNumOfPoints*sizeof(STPOINTS));
stPoints = (STPOINTS*) GlobalLock(hstPoints);
```

Note: Do not place the handle into your own variable or the base class does not release the memory when the object is deleted.

Now, you can perform the calculation and place the resulting data into the *stPoints* array. You can then graph the class using a Graph object, or access the resulting data by calling **GetCurveData( )**. When you delete the derived class, the memory for the array of points is released by the base class.

The methods for the base curve class, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Style methods** – All curves are displayed using their own curve style. This includes the color, line width, and line style for the curve.
- **Data access methods** – These methods provide direct access to the curve's array of points.

Table 9 briefly summarizes the methods for the base curve class.

Table 9: Base Curve Class Methods

Method Type	Method Name	Method Description
Constructor & Destructor Method	CcCurve( )	Constructor.
	~CcCurve( )	Destructor.
Style Methods	SetCurveStyle( )	Sets the color, width, and style for the curve.
	GetCurveStyle( )	Gets the color, width, and style for the curve.

Table 9: Base Curve Class Methods (cont.)

Method Type	Method Name	Method Description
Data Access Methods	GetCurveData( )	Gets a pointer to the curve data owned by or being used by the Curve object.
	SetCurveData( )	Sets the location of where the Curve object looks for its curve data to display.
	GetNumberOfPoints( )	Returns the number of curve data points associated with the Curve object.

Constructor and Destructor Methods

This section describes the constructor and destructor for the Curve object.

CcCurve( ) and ~CcCurve( )

Syntax

```
CcCurve* CCurve = new CcCurve( );
   //Base curve class
Delete CCurve;
```

Include File C_Curve.h, if using the base curve class.

Description The standard constructor and destructor for the object.

Notes Memory not allocated by the class is NOT released when the object is deleted by its base class pointer.

Style Methods

All curves are displayed on the Graph object using their own curve style, which includes the color, line width, and line style for the curve. This section describes the style methods in detail.

SetCurveStyle

Syntax

```
int SetCurveStyle(
  int *iCStyle,
  COLORREF *iCColor,
  int *iCWidth);
```

Include File C_Curve.h

Description Sets the curve's line color, line width, and line style.

Parameters

Name: iCStyle

Description: The line style used to draw the curve. This method uses the Windows SDK function **CreatePen( )**. As stated in the Windows SDK documentation, this value can be one of the following:

- PS_SOLID – Pen is solid.
- PS_DASH – Pen is dashed. This style is valid only when the pen width is one or less in device units.
- PS_DOT – Pen is dotted. This style is valid only when the pen width is one or less in device units.

Description (cont.):

- PS_DASHDOT – Pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.
- PS_DASHDOTDOT – Pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.
- PS_NULL – Pen is invisible.
- PS_INSIDEFRAME – Pen is solid. When this pen is used in any graphics device interface (GDI) drawing method that takes a bounding rectangle, the dimensions of the figure are shrunk so it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

Name: iCColor

Description: The color for the curve. Use the Windows **RGB( )** macro to define this color.

Name: iCWidth

Description: The width of the curve; 1 is the default.

Return Values

–1	Unsuccessful.
0	Successful.

GetCurveStyle

Syntax

```
int GetCurveStyle(
  int *iCStyle,
  COLORREF *iCColor,
  int *iCWidth);
```

Include File C_Curve.h

Description Gets the curve's line color, line width, and line style.

Parameters

Name: iCStyle

Description: The line style used to draw the curve. This method uses the Windows SDK function **CreatePen( )**. As stated in the Windows SDK documentation, this value can be one of the following:

- PS_SOLID – Pen is solid.
- PS_DASH – Pen is dashed. This style is valid only when the pen width is one or less in device units.
- PS_DOT – Pen is dotted. This style is valid only when the pen width is one or less in device units.
- PS_DASHDOT – Pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.
- PS_DASHDOTDOT – Pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.

Description (cont.):

- PS_NULL – Pen is invisible.
- PS_INSIDEFRAME – Pen is solid. When this pen is used in any graphics device interface (GDI) drawing method that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

Name: iCColor

Description: The color for the curve.

Name: iCWidth

Description: The width of the curve; 1 is the default.

Return Values

–1 Unsuccessful.

0 Successful.

Data Access Methods

These methods provide direct access to the curve's array of points.

GetCurveData

Syntax `STPOINTS* GetCurveData(void);`

Include File C_Curve.h

DT_Str.h

Description Returns a direct pointer to the curve's array of points.

Notes Before accessing the data, you need to call the method **GetNumberOfPoints( )** to get the number of points in the array.

Return Values

NULL	Unsuccessful.
A direct pointer to the curve data if successful.	Successful.

SetCurveData

Syntax

```
int SetCurveData(
  STPOINTS* stNewPoints,
  int iNumberOfPoints);
```

Include File C_Curve.h

DT_Str.h

Description Associates an array of points with the Curve object.

Parameters

Name: stNewPoints

Description: A pointer to the array of points that you want to associate with the Curve object.

Name: iNumberOfPoints

Description: The number of points in the array.

Notes Do not call this method if you are using a derived Curve object.

Return Values

–1	Unsuccessful.
0	Successful.

GetNumberOfPoints

Syntax `int GetNumberOfPoints(void);`

Include File C_Curve.h

Description Returns the number of points in the array of points associated with the Curve object.

Return Values

–1	Unsuccessful.
Returns the number of points.	Successful.

Graph Objects

In the field of imaging, it is often quite useful to display two-dimensional data that is derived from an image. The GLI/2 API makes this easy by providing the Curve, Graph, and List objects. You display two-dimensional data by drawing a curve(s) on a graph.

The Graph object displays a graph in a window. Many options are provided for displaying the graph. The graph contains a List object that holds a list of curves. Since a List object can contain an unlimited number of objects, a graph can contain an unlimited number of curves. A Curve object consists of a set of points. When you call **CGraph->ShowGraph()**, the Graph object draws the graph, and then draws the curves on the graph.

The Graph object contains internal variables to track a selected curve and a selected point on the selected curve. A curve becomes the selected curve when you call methods that return information about how mouse coordinates are related to points on the graph. Once a curve and point become selected, you can call other methods that return or set information about them. Using this type of functionality, it is easy to program the mouse to graphically interact with the data that is displayed on the curve. Thus, you can provide visual feedback and change curve data using pseudo drag-and-drop.

The methods for the Graph object, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Curve list method** – This method sets the list of curves for the graph to display.
- **Save and restore methods** – These methods save and restore the graph's appearance.
- **Text methods** – These methods set and retrieve the text for the graph's title, x-axis label, and y-axis label.

- **Show/print method** – This method shows the graph in a window or prints the graph to a printer.
- **Axis methods** – These methods set and return the x- and y-axis values.
- **Mouse methods** – These methods allow data interaction between the graph and the mouse.
- **Direct point access methods** – These methods allow direct access to the selected point on the selected curve on the graph.
- **Grid marking methods** – These methods set and retrieve the grid markings for the graph.
- **Dialog box methods** – These methods provide built-in dialog box procedures for changing the graph style.

Table 10 briefly summarizes the methods for the Graph object.

Table 10: Graph Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcGraph( ) –	Constructor.
	~CcGraph( )	Destructor.
Curve List Method	SetCurveList( )	Sets the list of Curve objects to be drawn by the graph.
Save and Restore Methods	SaveAppearance( )	Saves the current appearance of the graph to disk using the given full path name.
	RestoreAppearance( )	Restores a saved appearance from disk using the given full path name.
Text Methods	SetGraphText( )	Sets the graph's text.
	GetGraphText( )	Gets the graph's text.

Table 10: Graph Object Methods (cont.)

Method Type	Method Name	Method Description
Show/Print Method	ShowGraph( )	Displays the graph and all curves in a window (or prints them to the printer).
Axis Methods	SetMinMaxValues( )	Sets new minimum and maximum values for the x- and y-axis.
	GetMinMaxValues( )	Gets the current minimum and maximum values for the x- and y-axis.
Mouse Methods	IsCursorOnBP( )	Returns whether the given mouse location is on a curve point. If the mouse is over a point on a curve, the point's location is stored in the class as the selected point and the curve on which the point was found becomes the selected curve.
	IsCursorOnSelectedBP( )	Returns whether or not the given mouse location is on a selected curve point. If the mouse is over a point on the selected curve, the point's location is stored in the class as the selected point.
	GetPositionViaMouse( )	Returns the given mouse coordinates in graph coordinates. This is useful for showing the location of the mouse in graph coordinates as the mouse is dragged around the graph.
	SetSelBPViaMouse( )	Sets the selected point on the selected curve on the graph to the position given in mouse coordinates.
Direct Point Access Methods	SetSelBPDirect( )	Sets the location (position) of the selected point on the selected curve to the given location.
	GetSelBPDirect( )	Gets the location of the selected point on the selected curve on the graph.

Table 10: Graph Object Methods (cont.)

Method Type	Method Name	Method Description
Grid Marking Methods	SetGridMarkings( )	Sets new grid markings for the graph.
	GetGridMarkings( )	Gets the current grid markings for the graph.
Dialog Box Methods	ShowDLBLineStyle( )	Prompts for the desired color and style for the selected curve.
	ShowDLBSetGridMarkings( )	Prompts for the desired minor and major grid markings.
	ShowDLBSetMM( )	Prompts for the desired minimum and maximum axis values.
	ShowDLBTitle( )	Prompts for the desired graph title, x-axis label, and y-axis label.

Constructor and Destructor Methods

This section describes the constructor and destructor for the Graph object.

CcGraph( ) and ~CcGraph( )

Syntax

```
CcGraph* CGraph = new CcGraph( );
Delete CGraph;
```

Include File C_Graph.h, if using the graph class.

Description The standard constructor and destructor for the object.

Notes Memory not allocated by the class is NOT released when the object is deleted using its class pointer. This includes the list of curves graphed by the graph class. You are the owner of the list(s) and you need to free the memory for them.

Curve List Method

A Graph object first displays a graph in a window. It then draws each curve in its associated list of curves on the graph. The Graph object does not own the list of curves; you do. You simply need to tell the graph which list of curves to draw on the graph. This makes it possible to have multiple lists of curves, and then select which list is displayed on the graph using only one Graph object. If you give the Graph object a NULL value for a List object or an empty List object, no curves are drawn.

SetCurveList

Syntax `int SetCurveList(CcList* CList);`

Include File C_Graph.h

C_List.h

Description Associates a list of curves to be drawn on the graph.

Parameters

Name: CList

Description: Pointer to a List object that contains a list of Curve objects.

Return Values

–1	Unsuccessful.
0	Successful.

Save and Restore Methods

The Graph object lets you specify how the graph is drawn, including the following settings:

- Major and minor tick marks;
- Title, x-axis, and y-axis text; and
- Minimum and maximum axis scales.

This determines the overall appearance of the graph. It does not save the curve's appearance. You can save and restore all this information using the methods described in this section.

SaveApperance

Syntax

```
int SaveAppearance(
   char* cFileName);
```

Include File C_Graph.h

Description Saves the current appearance settings of the graph to disk.

Parameters

Name:	cFileName
Description:	Full path name of the file in which to save the settings.

Return Values

–1	Unsuccessful.
0	Successful.

RestoreApperance

Syntax

```
int RestoreAppearance(
   char* cFileName);
```

Include File C_Graph.h

Description Restores saved appearance settings of the graph from disk.

Parameters

Name:	cFileName
Description:	Full path name of the file that contains the settings.

Return Values

–1	Unsuccessful.
0	Successful.

Text Methods

Text methods set and retrieve the text for the graph's title, x-axis label, and y-axis label. This section describes the text methods in detail.

SetGraphText

Syntax

```
int SetGraphText(
  char* cTitle,
  char* cXAxis,
  char* cYAxis);
```

Include File C_Graph.h

Description Sets the graph's text for its title, x-axis label, and y-axis label.

Parameters

Name: cTitle

Description: Pointer to a string that contains the graph's title.

Name: cXAxis

Description: Pointer to a string that contains the graph's x-axis label.

Name: cYAxis

Description: Pointer to a string that contains the graph's y-axis label.

Return Values

–1 Unsuccessful.

0 Successful.

GetGraphText

Syntax

```
int GetGraphText(
  char* cTitle,
  char* cXAxis,
  char* cYAxis);
```

Include File C_Graph.h

Description Retrieves the graph's text for its title, x-axis label, and y-axis label.

Parameters

Name: cTitle

Description: Pointer to a string that contains the graph's title.

Name: cXAxis

Description: Pointer to a string that contains the graph's x-axis label.

Name: cYAxis

Description: Pointer to a string that contains the graph's y-axis label.

Return Values

–1 Unsuccessful.

0 Successful.

Show/Print Method

Once you have created the graph, associated a list of curves with the graph (optional), and set all graph details (optional), you can show the graph in a window or print the graph to a printer. This section describes the show / print method in detail.

ShowGraph

Syntax

```
int ShowGraph(
  hWND hWnd,
  HDC hdc,
  int iPrintFlag);
```

Include File C_Graph.h

Description Displays the graph in the given window or prints it to the printer.

Parameters

Name: hWnd

Description: A handle to the window in which to show the graph.

Name: hdc

Description: A handle to the device context for showing or printing the graph.

Name: iPrintFlag

Description: A flag that determines whether to show or print the graph. It can be one of the following values:

- 0 – Shows the graph in a window.
- 1 – Prints the graph to a printer.

Return Values

–1 Unsuccessful.

0 Successful.

Example This example shows the graph in a window as a result of getting the WM_PAINT message. This code is taken from the Histogram tool and is shown here with error checking and variable declaration removed:

```
void CcTool::OnPaint( )
{
//This is so the background color
//of the text is the correct color
```

Example (cont.)

```
 ::InvalidateRect(m_hWnd,
   NULL,TRUE);
//Call Begin & End Paint and
//get the HDC
CPaintDC dc(this);

//Show the graph
 CGraph->ShowGraph(m_hWnd,
   dc.m_hDC,0);
}
```

This example prints the graph to a printer. This code is taken from the Histogram tool and is shown here with error checking and variable declaration removed:

```
void CcTool::OnPrint ( )
{
//Get the handle to the printer's
// hDC via the common DLB
PrintDlg(&stPrintSetup);
hdcPrint=stPrintSetup.hDC;

//Set up document size
DocInfo.cbSize = sizeof(DOCINFO);
DocInfo.lpszDocName = "Histogram";
DocInfo.lpszOutput = (LPSTR)NULL;

//Start Document & Page
::StartDoc(hdcPrint,&DocInfo);
::StartPage(hdcPrint);

//Print Graph
 CGraph->ShowGraph(m_hWnd,
   hdcPrint,1);
```

Example (cont.)

```
//End Document & Page
::EndPage(hdcPrint);
::EndDoc(hdcPrint);

 //Free memory
 ::DeleteDC(hdcPrint);
}
```

Axis Methods

These methods set and return the minimum and maximum values for the x- and y-axis. This section describes the axis methods in detail.

SetMinMaxValues

Syntax

```
int SetMinMaxValues(
  float fXMin,
  float fXMax,
  float fYMin,
  float fYMax,
  int iXExp,
  int iXPre,
  int iYExp,
  int iYPre);
```

Include File C_Graph.h

Description Sets the minimum and maximum values for the x- and y-axis.

Parameters

Name: fXMin

Description: The minimum value for the x-axis.

Name: fXMax

Description: The maximum value for the x-axis.

Name: fYMin

Description: The minimum value for the y-axis.

Name: fYMax

Description: The maximum value for the y-axis.

Name: iXExp

Description: The exponent to use to display the x-axis values.

Name: iXPre

Description: The precision behind the decimal point to use for the values along the x-axis.

Name: iYExp

Description: The exponent to use to display the y-axis values.

Name: iYPre

Description: The precision behind the decimal point to use for the values along the y-axis.

Return Values

–1	Unsuccessful.
0	Successful.

GetMinMaxValues

Syntax

```
int GetMinMaxValues(
  float fXMin,
  float fXMax,
  float fYMin,
  float fYMax,
  int iXExp,
  int iXPre,
  int iYExp,
  int iYPre);
```

Include File C_Graph.h

Description Retrieves the minimum and maximum values for the x- and y-axis.

Parameters

Name: fXMin

Description: The minimum value for the x-axis.

Name: fXMax

Description: The maximum value for the x-axis.

Name: fYMin

Description: The minimum value for the y-axis.

Name: fYMax

Description: The maximum value for the y-axis.

Name: iXExp

Description: The exponent to use to display the x-axis values.

Name: iXPre

Description: The precision behind the decimal point to use for the values along the x-axis.

Name: iYExp

Description: The exponent to use to display the y-axis values.

Name: iYPre

Description: The precision behind the decimal point to use for the values along the y-axis.

Return Values

–1 Unsuccessful.

0 Successful.

Mouse Methods

It is sometimes useful to know if an operator is clicking on a point on a curve. You can use this information to show the exact location of the point, to move the point using the mouse, and, thus, change its value graphically, or perform other operations using the mouse.

The mouse methods inform you of the mouse's location with respect to the curve's point locations. This section describes the mouse methods in detail.

IsCursorOnBP

Syntax

```
int IsCursorOnBP(
   HDC hdc,
   POINT* stMousePoint);
```

Include File C_Graph.h

Description Determines if the given mouse position is near a point on the graph.

Parameters

Name: hdc

Description: Handle to the device context that is used to display the graph.

Name: stMousePoint

Description: The mouse position that is sent to you with the mouse message you are processing.

Notes

This method checks all the curves on the graph for a point that is near the given mouse point. The Graph object converts the mouse point into graph coordinates. If it finds a point on a curve, the curve containing the point becomes the selected curve and the point itself becomes the selected point. The actual value returned is the index into the selected curve's array of points to the selected point.

Remember that the Graph object is graphing a list of Curve objects that you associated with the Graph object using the method **SetCurveList( )**. Therefore, you own this List object containing the curves that the Graph object is graphing. The Graph object selects this curve by making it the selected curve within the List object using the method **SelectObjectAtIndex( )**.

Knowing this, it is possible to obtain the selected point and to change the selected point directly. You can also set the selected curve directly by calling **SelectObjectAtIndex( )**. If you no longer want any curves selected, you can call the method **SelectObjectAtIndex(-1).**

Notes (cont.) To easily access the selected point on the selected curve, you can use the methods **SetSelBPDirect( )** and **GetSelBPDirect( )**.

Obtain the *hdc* parameter as follows:

```
hdc = ::GetDC(m_hWnd);
CGraph-> IsCursorOnBP(
   hdc,stMousePoint);
::ReleaseDC(m_hWnd,hdc);
```

Return Values

–1	The mouse position is not near the point.
0	The mouse position is near the point.

IsCursorOnSelectedBP

Syntax

```
int IsCursorOnSelectedBP(
  hDC hdc,
  POINT* stMousePoint);
```

Include File C_Graph.h

Description Determines if the given mouse position is near a point on the selected curve on the graph.

Parameters

Name:	hdc
Description:	Handle to the device context that is used to display the graph.
Name:	stMousePoint
Description:	The mouse position that is sent to you with the mouse message you are processing.

Notes This method checks only the selected curve on the graph for a point that is near the given mouse point. The Graph object converts the mouse point into graph coordinates. If it finds a point on the selected curve, the selected curve containing the point remains the selected curve and the point itself becomes the selected point. The actual value returned is the index into the selected curve's array of points to the selected point.

Remember that the Graph object is graphing a list of Curve objects that you associated with the Graph object by calling **SetCurveList( )**. Therefore, you own the List object that contains the curves that the Graph object is graphing. The Graph object selects this curve by making it the selected curve within the List object by calling **SelectObjectAtIndex( )**.

Knowing this, it is possible to obtain the selected point and to change the selected point directly. You can also set the selected curve directly by calling **SelectObjectAtIndex( )**. If you no longer want any curves selected, you can call **SelectObjectAtIndex(-1)**.

To easily access the selected point on the selected curve, you can use **SetSelBPDirect( )** and **GetSelBPDirect( )**.

The *hdc* parameter can be obtained as follows:

```
hdc = ::GetDC(m_hWnd);
CGraph-> IsCursorOnSelectedBP(
   hdc,stMousePoint);
::ReleaseDC(m_hWnd,hdc);
```

Return Values

–1	The mouse position is not near the point.
0	The mouse position is near the point.

GetPositionViaMouse

Syntax

```
int GetPositionViaMouse(
  HDC hdc,
  POINT* stMousePoint,
  POINT* stLogical,
  STPOINTS* stGraph);
```

Include File C_Graph.h

Description Returns the position of the mouse in graph coordinates.

Parameters

Name:	hdc
Description:	Handle to the device context that is used to display the graph.
Name:	stMousePoint
Description:	The mouse position that is sent to you with the mouse message you are processing.
Name:	stLogical
Description:	Returned position of the mouse in logical coordinates.
Name:	stGraph
Description:	Returned position of the mouse in graph coordinates.

Notes You can use this method to provide visual feedback for the position of the mouse, in graph coordinates, as the mouse is moved around in the graph area.

Obtain the *hdc* parameter as follows:

```
hdc = ::GetDC(m_hWnd);
CGraph-> GetPositionViaMouse(
   hdc,stMousePoint,
   stLogical,stGraph);
::ReleaseDC(m_hWnd,hdc);
```

Return Values

–1 Unsuccessful.

0 Successful.

SetSelBPViaMouse

Syntax

```
int SetSelBPViaMouse(
  HDC hdc,
  POINT* stMousePoint);
```

Include File C_Graph.h

Description Sets the position of the selected point on the selected curve to that of the given mouse point.

Parameters

Name: hdc

Description: Handle to the device context that is used to display the graph.

Name: stMousePoint

Description: The mouse position that is sent to you with the mouse message you are processing.

Notes You can use this method to provide a pseudo drag-and-drop movement for the selected point on the selected curve. The visual feedback for this can be easily provided by changing the cursor while the mouse is being dragged.

Obtain the *hdc* parameter as follows:

```
hdc = ::GetDC(m_hWnd);
 CGraph-> SetSelBPViaMouse(
   hdc,stMousePoint);
 ::ReleaseDC(m_hWnd,hdc);
```

Return Values

–1 Unsuccessful.

0 Successful.

Direct Point Access Methods

If you wish to set or get the location of the selected point on the selected curve on the graph directly, you can use the direct point access methods. These methods are useful for exact placement of points on the graph. This section describes the direct point access methods in detail.

SetSelBPDirect

Syntax

```
int SetSelBPDirect(
  float fX,
  float fY);
```

Include File C_Graph.h

Description Sets the position of the selected point on the selected curve to the given values.

Parameters

Name: fX

Description: Horizontal position of point on graph, given in graph coordinates.

Name: fY

Description: Vertical position of point on graph, given in graph coordinates.

Return Values

–1 Unsuccessful.

0 Successful.

GetSelBPDirect

Syntax

```
int GetSelBPDirect(
  float* fX,
  float* fY);
```

Include File C_Graph.h

Description Gets the position of the selected point on the selected curve.

Parameters

Name: fX

Description: Horizontal position of point on graph, given in graph coordinates.

Name: fY

Description: Vertical position of point on graph, given in graph coordinates.

Return Values

–1	Unsuccessful.
0	Successful.

Grid Marking Methods

The graph has minor and major grid (or tick) markings for both the x- and y-axis. Axis coordinates are displayed at major grid markings. A grid marking can span the entire graph or appear at the very edge of the graph. Minor and major grid markings can be set separately. For an example of this, see any tool that uses a graph (such as the Histogram tool), and experiment with its grid settings. This section describes the grid marking methods in detail.

SetGridMarkings

Syntax

```
int SetGridMarkings(
  int iMajorX,
  int iMajorY,
  int iMinorX,
  int iMinorY,
  int iMajorXFlag,
  int iMajorYFlag,
  int iMinorXFlag,
  int iMinorYFlag);
```

Include File C_Graph.h

Description Sets the values for the current grid markings on the graph.

Parameters

Name: iMajorX

Description: Number of major grid markings for the x-axis. The minimum value is 2.

Name: iMajorY

Description: Number of major grid markings for the y-axis. The minimum value is 2.

Name: iMinorX

Description: Number of minor grid markings for the x-axis. The minimum value is 0.

Name: iMinorY

Description: Number of minor grid markings for the y-axis. The minimum value is 0.

Name: iMajorXFlag

Description: Flag for drawing major x-grid markings. Enter 0 for edge ticks, or 1 for full line.

Name: iMajorYFlag

Description: Flag for drawing major y-grid markings. Enter 0 for edge ticks, or 1 for full line.

Name: iMinorXFlag

Description: Flag for drawing minor x-grid markings. Enter 0 for edge ticks, or 1 for full line.

Name: iMinorYFlag

Description: Flag for drawing minor y-grid markings. Enter 0 for edge ticks, or 1 for full line.

Return Values

–1	Unsuccessful.
0	Successful.

GetGridMarkings

Syntax

```
int GetGridMarkings(
  int* iMajorX,
  int* iMajorY,
  int* iMinorX,
  int* iMinorY,
  int* iMajorXFlag,
  int* iMajorYFlag,
  int* iMinorXFlag,
  int* iMinorYFlag);
```

Include File C_Graph.h

Description Gets the values for the current grid markings on the graph.

Parameters

Name: iMajorX

Description: Number of major grid markings for the x-axis. The minimum value is 2.

Name: iMajorY

Description: Number of major grid markings for the y-axis. The minimum value is 2.

Name: iMinorX

Description: Number of minor grid markings for the x-axis. The minimum value is 0.

Name:	iMinorY
Description:	Number of minor grid markings for the y-axis. The minimum value is 0.
Name:	iMajorXFlag
Description:	Flag for drawing major x-grid markings. Enter 0 for edge ticks, or 1 for full line.
Name:	iMajorYFlag
Description:	Flag for drawing major y-grid markings. Enter 0 for edge ticks, or 1 for full line.
Name:	iMinorXFlag
Description:	Flag for drawing minor x-grid markings. Enter 0 for edge ticks, or1 for full line.
Name:	iMinorYFlag
Description:	Flag for drawing minor y-grid markings. Enter 0 for edge ticks, or 1 for full line.

Return Values

–1	Unsuccessful.
0	Successful.

Dialog Box Methods

You can change all the settings for how the graph is displayed and how the curves are displayed on the graph directly. The dialog box methods simplify this process by providing a simple user interface to query for the necessary information so that you do not have to write this code every time you use a Graph object. This section describes the dialog box methods in detail.

Note: The graph class uses a resource DLL named DT_GRes.DLL. It must be in path used by the SDK function **LoadLibrary( )**.

ShowDLBLineStyle

Syntax `int ShowDLBLineStyle(void);`

Include File C_Graph.h

Description Displays a dialog box that allows you to change the selected curve's style.

Notes The curve's style includes its color and style, but not its width. To change its width, you must do this directly. This dialog box procedure calls the Curve object's methods.

Return Values

–1 Unsuccessful.

0 Successful.

ShowDLBSetGridMarkings

Syntax `int ShowDLBSetGridMarkings(void);`

Include File C_Graph.h

Description Displays a dialog box that allows you to change the grid markings for the graph.

Return Values

–1 Unsuccessful.

0 Successful.

ShowDLBSetMM

Syntax `int ShowDLBSetMM(void);`

Include File C_Graph.h

Description Displays a dialog box that allows you to change the minimum and maximum values for the x- and y-axis.

Notes This procedure also sets the exponent and precision for the x- and y-axis.

Return Values

–1	Unsuccessful.
0	Successful.

ShowDLBTitle

Syntax `int ShowDLBTitle(void);`

Include File C_Graph.h

Description Displays a dialog box that allows you to change the graph's text.

Notes This includes the graph's title, x-axis label, and y-axis label.

Return Values

–1	Unsuccessful.
0	Successful.

List Objects

Keeping a list of needed items in any application is tedious and sometimes error prone. For this reason, a List object is provided to help you keep track of any GLI/2 derived object. This list can track all types of Image objects, all types of ROI objects, Curve objects, Graph objects, and other List objects.

In programming, two common elements are provided to create a list of items: the array and the linked list. Each has its pros and cons. Arrays are nice because you can access them by index, they are fast, and they do not fragment your memory; however, they are limited in size and sometimes waste memory. Linked lists are nice because they have no set amount of items that they can hold, but you cannot access them by index and they fragment your memory. The List object is fast, had unlimited storage, does not fragment memory, does not waste memory, and can be accessed by index or as a linked list. In addition, since all GLI/2 objects have a name, you can access objects in the list by name.

One last detail about the List object is that it holds other objects, not just items, and objects need to be deleted. The List object, if requested, deletes the objects in its list when you delete the List object. If the list contains other lists of other objects, you can free all memory for all objects by deleting the top List object. For example, the Blob Analysis tool uses this feature to easily delete all created blobs and all of their descendants.

If you wish to keep track of user-created objects, derive these objects from an GLI/2 object and store them in a GLI/2 List object. All objects contained by the list can be retrieved, inserted, and deleted using either an array index, name, or a linked list. Objects are stored in the list sequentially. The first object stored in the list has an index of 0, the second object stored in the list has an index of 1, and so on.

If the list has only three objects, do not attempt to insert it using an index of 5 (which is a position that does not exist yet). You could, however, insert an object into a list using an index of 5 if it had 6 or more objects. An easy way to always make sure you are inserting into the list properly is to use **InsertTail( )**. It is possible to insert into the head of the list or into the middle of the list using an index (and feel free to do so, if required). However, keep in mind that this is more work for the class and, thus, is slower, and it takes more code on your part not to insert into a position that does not exist yet.

In a normal or doubly linked list, call **GetHead()** followed by a number of calls to **GetNext()**. You might also call **GetTail()** followed by a number of calls to **GetPrev()**. In addition, you may want to mix array index calls such as **GetAtIndex()** with linked list methods such as **GetNext**(). When you call a method that is grouped with the get, insert, or delete method groups, the position of the object that is returned is marked as the current object. The next methods return the next object in the list and the previous methods return the previous object in the list from the current object. For example, if you call **GetAtIndex(5)** and then call **GetNext( ),** the object at Index 6 is returned. If you then call **GetNext( )** again, the object at Index 7 is returned, and so on. It is suggested that you do not use this type of coding in your programs because it is not to easy for others to follow.

The list has only one selected object at a time. A selected object in the list is an object you wish to track. By selecting an object in the list, you do not have to track it yourself. A selected object stays selected as you add and delete other objects in the list.To use a selected object, first select an object in the list, possibly add and delete other objects, and later request the selected object from the list. If you have no need for this type of functionality, do not use the selected methods. The class has no selected object by default.

When adding an object to the list, use **InsertTail( )**. You can add an unlimited number of objects like this and do not have to worry about anything else.

If you need an object from the list or wish to examine all the objects in the list, perform the following:

1. Get the number of objects in the list by calling the method **GetNumberOfObjects( )**.
2. Look through the list and retrieve each object until you find the one you want or have processed all of them, as follows:

```
for(x=0; x<CList->GetNumberOfObjects( ); x++)
 {
   CSomeObject = (cast to correct type)
       CList->GetAtIndex(x);
   (process objects)
}
```


An alternate way to easily retrieve an object in the list is to use object names. Remember, all GLI/2 objects have a name. Before inserting the object into the List object, make sure it has a name associated with it using the GLI/2 base class **SetName( )** method. Then, when you want to retrieve the object, query the List object using **GetViaName( )**.

The methods for the List object, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Retrieve methods** – These methods retrieve a pointer to the desired object. You must know what type of object you are retrieving and cast the pointer accordingly before using these methods.
- **Insert methods** – These methods insert a pointer to the desired object into the list at the desired location. The best way to insert into the List object is by using **InsertTail( )**.
- **Delete methods** – These methods remove an object from the list. The List object deletes the object, if requested, when removing it from its list.

- **General methods** – These methods query and set the list's general information.

Table 11 briefly summarizes the methods for the List object.

Table 11: List Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcList( )	Constructor.
	~CcList( )	Destructor.
Retrieve Methods	GetHead( )	Retrieves a pointer to the first object in the list.
	GetNext( )	Retrieves a pointer to the next object in the list.
	GetPrev( )	Retrieves a pointer to the previous object in the list.
	GetTail( )	Retrieves a pointer to the last object in the list.
	GetAtIndex( )	Retrieves a pointer to the object at the given zero based index in the list.
	GetViaName( )	Retrieves a pointer to the first object in the list with the specified name.
	GetSelected( )	Retrieves a pointer to the selected object in the list.
Insert Methods	InsertHead( )	Inserts the given object into the first position in the list.
	InsertTail( )	Inserts the given object at the end of the list.
	InsertAtIndex( )	Inserts the given object at the given index into the list. All other objects after this are moved down in the list.

Table 11: List Object Methods (cont.)

Method Type	Method Name	Method Description
Insert Methods (cont.)	InsertSelected( )	Inserts the given object at the position of the current object into the list and makes this object the selected object. All other objects after this are moved down in the list.
Delete Methods	DeleteHead( )	Deletes the first object in the list.
	DeleteTail( )	Deletes the last object in the list.
	DeleteAtIndex( )	Deletes the object at the given index from the list.
	DeleteViaName( )	Deletes the first object in the list with the given name from the list.
	DeleteSelected( )	Deletes the selected object from the list.
General Methods	GetNumberOfObjects( )	Returns the number of objects in the list.
	SelectObjectAtIndex( )	Selects the object at the given index.
	GetSelectedObjectsIndex( )	Returns the selected object's index.
	GetCurrentObjectsIndex( )	Returns the current object's index.
	SetDestructionType( )	Sets the destruction functionality for the entire list. When an object is removed from the list (either using a delete method or when the list itself is deleted), the list has the option of deleting the object as the object is removed from the list, or simply removing the object from the list. By default, the List object does NOT delete the stored object when removing it from its list of objects.

Constructor and Destructor Methods

This section describes the constructor and destructor for the List object.

CcList() and ~CcList()

Syntax
```
CcList* CList = new CcList( );
Delete CList;
```

Include File C_List.h, if using list class.

Description The standard constructor and destructor for the List object.

Return Values

–1	Unsuccessful.
0	Successful.

Retrieve Methods

These methods retrieve a pointer to the desired object. You must know what type of object you are retrieving and cast the pointer accordingly before using it. The object you successfully retrieve becomes the current object in the list.

GetHead

Syntax
```
CcHLObject* GetHead(void);
```

Include File C_List.h

Description Returns the first object in the list.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

GetNext

Syntax `CcHLObject* GetNext(void);`

Include File C_List.h

Description Returns the next object from the current object in the list.

Notes The current object is set to the desired object on any successful get, insert, or delete operation.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

GetPrev

Syntax `CcHLObject* GetPrev(void);`

Include File C_List.h

Description Returns the previous object from the current object in the list.

Notes The current object is set to the desired object on any successful get, insert, or delete operation.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

GetTail

Syntax

```
CcHLObject* GetTail(void);
```

Include File C_List.h

Description Returns the last object in the list.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

GetAtIndex

Syntax

```
CcHLObject* GetAtIndex(
  const int iIndex);
```

Include File C_List.h

Description Returns the object at the given index in the list.

Parameters

Name:	iIndex
Description:	Zero based index into the list of objects.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

GetViaName

Syntax

```
CcHLObject* GetViaName(
   const char* cName);
```

Include File C_List.h

Description Returns the first object in the list with the given name.

Parameters

Name: cName

Description: The exact name of the object to return.

Notes This method returns the first object in the list with the given exact name. If you have more than one object in the list with the same name, this method always returns the first one.

Return Values

NULL Unsuccessful.

A pointer to the desired object. Successful.

GetSelected

Syntax

```
CcHLObject* GetSelected(void);
```

Include File C_List.h

Description Returns the selected object in the list.

Notes The list by default has no selected object. You must first select an object before you can retrieve it. If no selected object is in the list, this method returns NULL.

Return Values

NULL	Unsuccessful.
A pointer to the desired object.	Successful.

Insert Methods

These methods insert a pointer to the desired object into the list at the desired location. The best way to insert into the List object is using **InsertTail( )**. You do not need to cast the pointer when using the insert methods. The object you successfully insert becomes the current object in the list. This section describes the insert methods.

InsertHead

Syntax

```
int InsertHead(
   CcHLObject* CObject);
```

Include File C_List.h

Description Inserts the given object into first position in the list.

Parameters

Name:	CcHLObject*
Description:	Pointer to the desired object you want inserted into the list.

Notes The first position in the list is the head position. It has an index value of 0.

Return Values

–1	Unsuccessful.
0	Successful.

InsertTail

Syntax

```
int InsertTail(
   CcHLObject* CObject);
```

Include File C_List.h

Description Inserts the given object into the last position in the list.

Parameters

Name: CcHLObject*

Description: Pointer to the desired object that you want inserted into the list.

Notes This is the most reliable and fastest way to insert objects into the list.

Return Values

–1 Unsuccessful.

0 Successful.

InsertAtIndex

Syntax

```
int InsertAtIndex(
  CcHLObject* CObject,
  const int iIndex);
```

Include File C_List.h

Description Inserts the given object at the given index.

Parameters

Name: CcHLObject*

Description: A pointer to the desired object that you want inserted into the list.

Name: IIndex

Description: The zero based index of the position where you want to insert the given object.

Return Values

−1 Unsuccessful.

0 Successful.

InsertSelected

Syntax

```
int InsertSelected(
   CcHLObject* CObject);
```

Include File C_List.h

Description Inserts the given object at the current position in the list and makes this object the selected object.

Parameters

Name: CcHLObject*

Description: Pointer to the desired object that you want inserted into the list.

Notes If you wish to place the object at a specific position in the list and make it the selected object, you can first place it in the list and then make it the selected object by calling **SelectObjectAtIndex( )**. If you do not know the object's index, see **GetCurrentObjectsIndex( )**.

Return Values

−1 Unsuccessful.

0 Successful.

Delete Methods

These methods remove an object from the list. The List object deletes the object, if requested, when removing it from its list. The object that fills the position of the successfully deleted object becomes the current object in the list.

When an object is deleted from the list, all objects following this position are moved up in the list. For example, if 10 objects are in the list, and you delete the object at position 5, objects at indexes 6, 7, 8, and 9 are moved into positions 5, 6, 7, and 8, respectively. If you are deleting a large group of objects be aware of this; it is easier and faster to delete them from the end of the list backwards. This section describes the delete methods in detail.

DeleteHead

Syntax `int DeleteHead(void);`

Include File C_List.h

Description Removes the first object in the list.

Notes The object that is removed from the list is also deleted, if requested. You can request the List object to delete objects by calling **SetDestructionType( )**.

Return Values

–1 Unsuccessful.

0 Successful.

DeleteTail

Syntax `int DeleteTail(void);`

Include File C_List.h

Description Removes the last object in the list.

Notes The object that is removed from the list is also deleted, if requested. You can request the List object to delete objects by calling **SetDestructionType( )**.

Return Values

–1 Unsuccessful.

0 Successful.

DeleteAtIndex

Syntax

```
int DeleteAtIndex(
  const int iIndex);
```

Include File C_List.h

Description Removes the object at the given index from the list.

Parameters

Name: iIndex

Description: The zero based index for the object that you want removed.

Notes The object that is removed from the list is also deleted, if requested. You can request the List object to delete objects by calling **SetDestructionType( )**.

Notes (cont.) All objects in the list after the given index are moved up in the list. For example, if 10 objects are in the list, and you delete the object at position 5, objects at indexes 6, 7, 8, and 9 are moved into positions 5, 6, 7, and 8, respectively. If you are deleting a large group of objects using this method, it is easier and faster to delete them from the end of the list backwards.

Return Values

–1	Unsuccessful.
0	Successful.

DeleteViaName

Syntax

```
int DeleteViaName(
  const char* cName);
```

Include File C_List.h

Description Removes the object with the given name from the list.

Parameters

Name:	CName
Description:	The name of the object that you want to remove from the list.

Notes The object that is removed from the list is also deleted, if requested. You can request the List object to delete objects by calling **SetDestructionType( )**.

Return Values

–1	Unsuccessful.
0	Successful.

DeleteSelected

Syntax `int DeleteSelected(void);`

Include File C_List.h

Description Removes the selected object from the list.

Notes The object that is removed from the list is also deleted, if requested. You can request the List object to delete objects by calling **SetDestructionType( )**. After the selected object is removed from the list, the list no longer contains a selected object.

Return Values

–1	Unsuccessful.
0	Successful.

General Methods

The general methods query and set the list's general information. This section describes the general methods in detail.

GetNumberOfObjects

Syntax `int GetNumberOfObjects(void);`

Include File C_List.h

Description Returns the number of objects in the list.

Return Values

–1	Unsuccessful.
Returns the number of objects in the list.	Successful.

SelectObjectsAtIndex

Syntax

```
int SelectObjectAtIndex(
   const int iIndex);
```

Include File C_List.h

Description Makes the object at the given index the selected object in the list.

Parameters

Name:	iIndex
Description:	The zero based index of the object that you want to be the selected object.

Notes If a selected object already exists in the list, the object is no longer the selected object. Only one selected object can be in the list at any given time.

Return Values

–1	Unsuccessful.
0	Successful.

GetSelectedObjectsIndex

Syntax

```
int GetSelectedObjectsIndex(void);
```

Include File C_List.h

Description Returns the zero based index of the selected object in the list.

Notes If no selected object exists in the list, this method returns –1.

Return Values

–1 Unsuccessful.

The index. Successful.

GetCurrentObjectsIndex

Syntax `int GetCurrentObjectsIndex(void);`

Include File C_List.h

Description Returns the zero based index of the current object in the list.

Return Values

–1 Unsuccessful.

The index. Successful.

SetDestructionType

Syntax `int SetDestructionType(int iType);`

Include File C_List.h

Description Sets the mode of operation for removing objects from the list.

Parameters

Name: iType

Description: Mode of operation for removing objects from the list. It can be one of the following:

- LIST_DONT_DELETE_ON_ DESTRUCTOR – Object is removed only from the list (default).
- LIST_DELETE_ON_DESTRUCTOR –Object is deleted and removed from the list.

Notes The List object by default does not delete the objects it contains in its list. If you call this method with the LIST_DELETE_ON_ DESTRUCTOR parameter, then the List object deletes the objects it contains. When in this mode of operation, the List object deletes the objects when you use any of its delete methods to remove an object from the list. It also deletes all objects in its list when the list object itself is deleted.

Using this functionality, you can allocate and organize a large number of objects. These objects can be organized by using List objects that contain other List objects (and so on) that contain other objects. If the mode of all of these List objects is set to LIST_DELETE_ON_ DESTRUCTOR, then all memory for all List objects and all the objects that they contain is released to the system by deleting the top List object.

Notes (cont.)

Do not delete an object contained in a List object directly if the mode of the List object is set to LIST_DELETE_ON_DESTRUCTOR. This is because the List object tries to delete it again when you delete the List object. If you want to delete an object contained in a list, use one of the list's delete methods. Also, do not have the same object contained in more than one List object if the mode of the List object is set to LIST_DELETE_ON_DESTRUCTOR.

The Blob Analysis tool uses this functionality to track all of its blobs and all of their descendants. Refer to Chapter 5 starting on page 267 for more information on the Blob Analysis tool.

Return Values

–1	Unsuccessful.
0	Successful.

Calibration Objects

Calibration objects convert pixel coordinates to real-world coordinates and pixel areas to real-world areas. Tools use Calibration objects to measure items in images in real-world coordinates. Before a Calibration object can convert pixel coordinates to real-world coordinates, the Calibration object needs to be calibrated. Once calibrated, Calibration objects can save themselves to disk.

The methods for the Calibration objects, grouped by method type, are as follows:

- **Constructor and destructor methods** – Standard methods.
- **Calibration method** – This method calibrates the Calibration object. A Calibration object must be calibrated before it can be used to convert pixel coordinates to real-world coordinates.
- **Conversion methods** – These methods convert pixel coordinates to real-world coordinates and areas.
- **Save and restore methods** – These methods save and restore a Calibration object to and from disk.
- **General methods** – These methods are general calibration methods.

Table 12 briefly summarizes the methods for the Calibration object.

Table 12: Calibration Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcCalibration( )	Constructor.
	CcCalibration( )	Destructor.
Calibration Method	DoCalibration( )	Calibrates the Calibration object.

Table 12: Calibration Object Methods (cont.)

Method Type	Method Name	Method Description
Conversion Methods (cont.)	ConvertPoint( )	Converts a given point in pixels to real-world coordinates.
	GetAreaOfPixel( )	Converts a given point in pixels to a real-world area measurement.
Save and Restore Methods	Save( )	Saves the Calibration object's calibration.
	Open( )	Restores a Calibration object's calibration.
General Methods	SetUnitsOfMeasure( )	Sets the unit of measure used to calibrate the Calibration object.
	GetUnitsOfMeasure( )	Returns the unit of measure used to calibrate the Calibration object.
	GetSizeOfImage( )	Returns the size of the image used to calibrate the Calibration object.

Constructor and Destructor Methods

This section describes the constructor and destructor for the Calibration object.

CcCalibration( ) and ~ CcCalibration( )

Syntax

```
CcCalibration* CCal =
   new CcCalibration( );
Delete CCal;
```

Include File C_Calibr.h, if using a calibration class.

Description The standard constructor and destructor for the object.

Calibration Method

This method calibrates the Calibration object. A Calibration object must be calibrated before it can be used to convert pixel coordinates to real-world coordinates. A Calibration object is calibrated using four pairs of known image points and real-world points. This section describes the calibration methods in detail.

DoCalibration

Syntax

```
int DoCalibration(
  STPOINTS* stImagePoints,
  STPOINTS* stWorldPoints,
  int iNumberOfPoints,
  int iWidthOfImage,
  int iHeightOfImage);
```

Include File C_Calibr.h

Description Calibrates the Calibration object using the given image and real-world coordinates.

Parameters

Name: stImagePoints

Description: Array of four image points given in subpixel locations.

Name: stWorldPoints

Description: Array of four real world points.

Name: iNumberOfPoints

Description: The number of points in the array; this value must be 4.

Name: iWidthOfImage

Description: The width of the image you are calibrating.

Name: iHeightOfImage

Description: The height of the image you are calibrating.

Notes You can use the Calibration object to calibrate and save a Calibration object. Then, open the saved Calibration object from disk using the method **Open( )** to restore the calibration.

This method takes exactly four pairs of pixels and the associated real-world coordinates. The pixel points can be subpixel points to increase your accuracy

Return Values

–1 Unsuccessful.

0 Successful.

Conversion Methods

These methods convert pixel coordinates to real-world coordinates and areas. Subpixel accuracy is used to perform all calculations.

ConvertPoint

Syntax

```
int ConvertPoint(
  STPOINTS* stImagePoint,
  STPOINTS* stWorldPoint);
```

or

```
int ConvertPoint(
  float fImageX,
  float fImageY,
  float* fWorldX,
  float* fWorldY);
```

Include File C_Calibr.h

Description Converts the given pixel point to a real-world coordinate.

Parameters

Name: stImagePoint

Description: Pointer to a STPOINTS structure that contains the given subpixel pixel point to convert to real-world coordinates.

Name: stWorldPoint

Description: Pointer to a STPOINTS structure that receives the real-world coordinates.

Name: fImageX

Description: Subpixel x-pixel point to convert to real-world coordinates.

Name: fImageY

Description: Subpixel y-pixel point to convert to real-world coordinates.

Name: fWorldX

Description: Pointer to a float variable that receives the real world x-coordinate.

Name: fWorldY

Description: Pointer to a float variable that receives the real-world y-coordinate.

Notes Two versions of this method are provided. Each performs the same operation. The first version takes the pixel points in the form of a STPOINTS structure; the second version enters each value in a float variable. Use the version that is more convenient for you.

Notes (cont.) The STPOINTS structure is described as follows:

```
struct STPOINTS {
  float fX,fY;
  };
```

Return Values

–1	Unsuccessful.
0	Successful.

GetAreaOfPixel

Syntax

```
float GetAreaOfPixel(
  int ix,
  int iy);
```

Include File C_Calibr.h

Description Returns the real-world calibrated area for the given pixel location.

Parameters

Name:	ix
Description:	The x-location of pixel for desired area.
Name:	iy
Description:	The y-location of pixel for desired area.

Return Values

–1	Unsuccessful.
The calibrated area for the given pixel location.	Successful.

Save and Restore Methods

The **Save** method saves a Calibration object to disk. The **Open** method restores a Calibration object from disk. This section describes these methods in detail.

Save

Syntax `int Save(char* cFileName);`

Include File C_Calibr.h

Description Saves the Calibration object's calibration to disk.

Parameters

Name: cFileName

Description: Full path name of file in which to save the calibration information.

Return Values

–1 Unsuccessful.

0 Successful.

Open

Syntax `int Open(char* cFileName);`

Include File C_Calibr.h

Description Restores the Calibration object's calibration information from disk.

Parameters

Name: cFileName

Description: Full path name of file from which to restore the calibration information.

Return Values

–1 Unsuccessful.

0 Successful.

General Methods

This section describes the general calibration methods in detail.

SetUnitsOfMeasure

Syntax

```
int SetUnitsOfMeasure(
   char* cNewUnitsOfMeasure);
```

Include File C_Calibr.h

Description Sets the units of measure for the Calibration object.

Parameters

Name: cNewUnitsOfMeasure

Description: Text-based description of the units of measure used to calibrate the object.

Notes The units of measure can be anything you wish and are not used in calculations. These units provide a textual description of measurement that is used to calibrate the object.

Return Values

–1	Unsuccessful.
0	Successful.

GetUnitsOfMeasure

Syntax
```
char* GetUnitsOfMeasure(void);
```

Include File C_Calibr.h

Description Gets the units of measure for the Calibration object.

Notes The units of measure can be anything you wish and are not used in calculations. The units are a textual description of the measurement that is used to calibrate the object.

Return Values

NULL	Unsuccessful.
The textual-based description of the units of measure used to calibrate the object.	Successful.

GetSizeOfImage

Syntax
```
int GetSizeOfImage(
  int* iWidthOfImage,
  int* iHeightOfImage);
```

Include File C_Calibr.h

Description Returns the size of the image that is used to calibrate the object.

Parameters

Name: iWidthOfImage

Description: Pointer to *int* to return the image's width used to calibrate the object.

Name: iHeightOfImage

Description: Pointer to *int* to return the image's height used to calibrate the object.

Notes Each Calibration object is calibrated using an input image. You can then use this Calibration object on all images taken with the same camera setup. This method is provided so that you can check the size of the original image that is used to calibrate the object.

Return Values

–1 Unsuccessful.

0 Successful.

3

Using the Arithmetic Tool API

Overview of the Arithmetic Tool API

The API for the Arithmetic tool has one object only: the CcArithmetic class. This tool performs an arithmetic operation on one or more images (derived from class CcImage), and places the result into an output image. It performs this operation with respect to the given ROI (derived from class CcRoiBase).

The CcArithmetic class uses a standard constructor and destructor and the class methods listed in Table 13.

Table 13: CcArithmetic Class Methods

Method Type	Method Name
Constructor & Destructor Methods	CcArithmetic(void);
	~CcArithmetic(void);
CcArithmetic Class Methods	int Add(CcImage* CImageIn1,CcImage* CImageIn2, CcImage* CImageOut,CcRoiBase* Croi,int iFlag, float fGain, float fOffset,float fLowThreshold, float fHiThreshold);
	int AddRGB(Cc24BitRGBImage* CImageIn1, Cc24BitRGBImage* CImageIn2, Cc24BitRGBImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int AddHSL(Cc24BitHSLImage* CImageIn1, Cc24BitHSLImage* CImageIn2, Cc24BitHSLImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int Sub(CcImage* CImageIn1, CcImage* CImageIn2, CcImage* CImageOut,CcRoiBase* CRoi,int iFlag, float fGain,float fOffset,float fLowThreshold, float fHiThreshold);

Table 13: CcArithmetic Class Methods (cont.)

Method Type	Method Name
CcArithmetic Class Methods (cont.)	int SubRGB(Cc24BitRGBImage* CImageIn1, Cc24BitRGBImage* CImageIn2, Cc24BitRGBImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int SubHSL(Cc24BitHSLImage* CImageIn1, Cc24BitHSLImage* CImageIn2, Cc24BitHSLImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int Mul(CcImage* CImageIn1,CcImage* CImageIn2, CcImage* CImageOut,CcRoiBase* CRoi,int iFlag, float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int MulRGB(Cc24BitRGBImage* CImageIn1, Cc24BitRGBImage* CImageIn2, Cc24BitRGBImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int MulHSL(Cc24BitHSLImage* CImageIn1, Cc24BitHSLImage* CImageIn2, Cc24BitHSLImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int Div(CcImage* CImageIn1, CcImage* CImageIn2,CcImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int DivRGB(Cc24BitRGBImage* CImageIn1, Cc24BitRGBImage* CImageIn2, Cc24BitRGBImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);

Table 13: CcArithmetic Class Methods (cont.)

Method Type	Method Name
CcArithmetic Class Methods (cont.)	int DivHSL(Cc24BitHSLImage* CImageIn1, Cc24BitHSLImage* CImageIn2, Cc24BitHSLImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int LogicalAND(CcImage* CImageIn1, CcImage* CImageIn2,CcImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int LogicalOR (CcImage* CImageIn1, CcImage* CImageIn2,CcImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int LogicalXOR(CcImage* CImageIn1, CcImage* CImageIn2,CcImage* CImageOut, CcRoiBase* CRoi,int iFlag,float fGain,float fOffset, float fLowThreshold,float fHiThreshold);
	int Copy(CcImage* CImageIn1, CcImage* CImageOut,CcRoiBase* CRoi,int iFlag, float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int CopyRGB(Cc24BitRGBImage* CImageIn1, Cc24BitRGBImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);
	int CopyHSL(Cc24HSLRGBImage* CImageIn1, Cc24BitHSLImage* CImageOut,CcRoiBase* CRoi, int iFlag,float fGain,float fOffset,float fLowThreshold, float fHiThreshold);

CcArithmetic Methods

This section describes each method of the CcArithmetic class in detail.

Add/AddRGB/AddHSL

Syntax

```
int Add(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files C_Arith.h

Description Performs the following arithmetic operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain*(CImageIn1
   + CImageIn2) + fOffset
CImageOut = |CImageOut|
//Threshold CImageOut so that:
fLowThreshold <= CImageOut <=
   fHiThreshold
```

This method adds two input images together with respect to the given ROI and takes the absolute value of the resulting data, if it is specified in the *iFlag* parameter. It then performs thresholding on the resulting data, if it is specified in the *iFlag* parameter.

Description (cont.) The order of operations for this method is as follows:

1. Adds two images together.
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.

Description (cont.):
- ARITH_THRESHOLD – Thresholds the resulting data between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes

AddRGB( ) and **AddHSL( )** are identical to **Add( )**, except that they perform the operation on all three color planes at once.

These methods use images derived from the CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. These methods use an ROI derived from the CcRoiBase class. These include all GLI/2 ROIs. They also work with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
Cc24BitRGBImage*CColorImage;
//24-bit Color Image
CcGrayImage256*C8BitImage;
//8-bit grayscale Image
CcGrayImageInt32*C32BitImage;
//32-bit grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/

//Allocate memory for objects
 CColorImage = new
   Cc24BitRGBImage( );
 C8BitImage=new CcGrayImage256( );
 C32BitImage = new
   CcGrayImageInt32( );
 CRectRoi = new CcRoiRect( );

//Initialize ROI
 RECT stROI;
 stROI.bottom = 50;
 stROI.top = 150;
 stROI.left = 50;
 stROI.right = 150;
CRectRoi->
   SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
CColorImage->OpenBMPFile(
   “image1.bmp”);
```

Example (cont.)

```
C8BitImage->OpenBMPFile(
   “image2.bmp”);
C32BitImage->OpenBMPFile(
   “image3.bmp”);
//Perform addition Image3 =
//Image1 + Image2.
 CArith.Add(CColorImage,
   C8BitImage, C32BitImage,
   CRectRoi, ARITH_ABS_VALUE |
   ARITH_THRESHOLD,
//Perform Absolute Value and
//Thresholding
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold between 0 and
      //255
   255);
//Save output to disk
C32BitImage->SaveBMPFile(
   “output.bmp”);
//Free memory
delete CColorImage;
delete C8BitImage;
delete C32BitImage;
delete CRectRoi;
}
```

Sub/SubRGB/SubHSL

Syntax

```
int Sub(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files

C_Arith.h

Description

Performs the following arithmetic operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1 -
   CImageIn2) + fOffset.
CImageOut = |CImageOut|.
//Threshold CImageOut so that:
fLowThreshold <= CImageOut <=
   fHiThreshold.
```

The method subtracts input image 2 from input image 1 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Subtracts the two images.
2. Applies gain and offset.

Description (cont.)

3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes

SubRGB( ) and **SubHSL( )** are identical to **Sub( )**, except that they perform the operation on all three color planes at once.

This method uses images derived from the GLI/2 supplied CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses an ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
Cc24BitRGBImage*CColorImage;
//24-bit Color Image
CcGrayImage256*C8BitImage;
//8-bit grayscale Image
CcGrayImageInt32*C32BitImage;
//32-bit grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/

//Allocate memory for objects
CColorImage=
   new Cc24BitRGBImage( );
C8BitImage=new CcGrayImage256( );
C32BitImage =
   new CcGrayImageInt32( );
CRectRoi = new CcRoiRect( );

//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->
   SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
 CColorImage->
   OpenBMPFile(“image1.bmp”);
```

Example (cont.)

```
 C8BitImage->
   OpenBMPFile(“image2.bmp”);
 C32BitImage->
   OpenBMPFile(“image3.bmp”);

//Perform subtraction Image3 =
//Image1 - Image2.
CArith.Sub(CColorImage,
   C8BitImage, C32BitImage,
   CRectRoi, ARITH_THRESHOLD,
//Perform Thresholding only
1, //Set gain to 1
0, //Set offset to 0
0, //Threshold between 0 and 255
255);
//Save output to disk
C32BitImage->
   SaveBMPFile(“output.bmp”);
//Free memory
delete CColorImage;
delete C8BitImage;
delete C32BitImage;
delete CRectRoi;
}
```

Mul/MulRGB/MulHSL

Syntax

```
int Mul(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files C_Arith.h

Description Performs the following arithmetic operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1 *
   CImageIn2) + fOffset
CImageOut = |CImageOut|
//Threshold CImageOut so that:
fLowThreshold <= CImageOut <=
   fHiThreshold
```

This method multiplies input image 1 with input image 2 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Multiplies the two images.
2. Applies gain and offset.

Description (cont.)

3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

CImageIn1

Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD –Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes **MulRGB( )** and **MulHSL( ) are** identical to **Mul( )**, except that they perform the operation on all three color planes at once.

This method uses images derived from the GLI/2 supplied CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses an ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImage256*C8BitImage2;
//8-Bit grayscale Image 2
CcGrayImageInt32*C32BitImage;
```

Example (cont.)

```
//32-Bit grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/
//Allocate memory for objects
 C8BitImage1=
   new CcGrayImage256( );
 C8BitImage2=
    new CcGrayImage256( );
 C32BitImage=
    new CcGrayImageInt32( );
 CRectRoi= new CcRoiRect( );
//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->
   SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
C8BitImage1->
   OpenBMPFile(“image1.bmp”);
C8BitImage2->
   OpenBMPFile(“image2.bmp”);

   C32BitImage->OpenBMPFile(“image
   3.bmp”);
//Perform multiplication Image3 =
//Image1 X Image2. Take two 8-bit
//images, multiply them, and place
//result into a 32-bit image so
//that no precision is lost.
```

Example (cont.)

```
CArith.Mul(C8BitImage1,
   C8BitImage2, C32BitImage,
   CRectRoi,
   0,//Do not threshold or take
     //absolute value
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold values unused
   0);

//Save output to disk
C32BitImage->
   SaveBMPFile("output.bmp");
//Free memory
delete C8BitImage1;
delete C8BitImage2;
delete C32BitImage;
delete CRectRoi;
}
```


Div/DivRGB/DivHSL

Syntax

```
int Div(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  cRoiBase* cRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files C_Arith.h

Description

Performs the following arithmetic operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1 /
   CImageIn2) + fOffset.
CImageOut = |CImageOut|.
//Threshold CImageOut so that:
 fLowThreshold <= CImageOut <=
   fHiThreshold.
```

Divides input image 1 by input image 2 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Divides the two images.
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes

DivRGB( ) and **DivHSL( )** are identical to **Div( )**, except that they perform the operation on all three color planes at once.

This method uses images derived from the GLI/2 supplied CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses an ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1	Unsuccessful.
0	Successful

Example

The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImage256*C8BitImage2;
//8-Bit grayscale Image 2
CcGrayImageFloat*CFloatImage;
//Floating-point grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/
//Allocate memory for objects
 C8BitImage1=
   new CcGrayImage256( );
```

Example (cont.)

```
C8BitImage2=
   new CcGrayImage256( );
CFloatImage=
   new CcGrayImageFloat( );
CRectRoi= new CcRoiRect( );
//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->
   SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
 C8BitImage1->
   OpenBMPFile(“image1.bmp”);
 C8BitImage2->
   OpenBMPFile(“image2.bmp”);
 CFloatImage->
   OpenBMPFile(“image3.bmp”);
//Perform Division Image3
//= Image1 / Image2.
//Take two 8-bit images, divide
//them, and place result into a
//floating-point image so that you
//do not loose any precision.
CArith.Div(C8BitImage1,
   C8BitImage2, CFloatImage,
   CRectRoi,ARITH_ABS_VALUE,
//Do not threshold but take
//absolute value
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold values unused
   0);
```

Example (cont.)

```
//Save output to disk
 CFloatImage->
   SaveBMPFile("output.bmp");
//Free memory
delete C8BitImage1;
delete C8BitImage2;
delete CFloatImage;
delete CRectRoi;
}
```

LogicalAnd

Syntax

```
int LogicalAND(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files

C_Arith.h

Description

Performs the following operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1 &
   CImageIn2) + fOffset.
CImageOut = |CImageOut|.
//Threshold CImageOut so that:
fLowThreshold <= CImageOut <=
   fHiThreshold.
```

Description (cont.) This method performs a logical bitwise AND with input image 1 and input image 2 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Bitwise ANDs the two images (as 32-bit integers).
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes All values are converted to 32-bit integers before performing the logical bitwise AND operation.

Notes (cont.) This method uses images derived from the GLI/2 supplied CcImage class. These include binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses a ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImage256*C8BitImage2;
//8-Bit grayscale Image 2
CcGrayImageFloat*CFloatImage;
//Floating-point grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
/*End of Dec Section*/
//Allocate memory for objects
 C8BitImage1 =
   new CcGrayImage256( );
C8BitImage2 =
   new CcGrayImage256( );
 CFloatImage =
   new CcGrayImageFloat( );
 CRectRoi = new CcRoiRect( );
```

Example (oont.)

```
 //Initialize ROI
 RECT stROI;
 stROI.bottom = 50;
 stROI.top = 150;
 stROI.left = 50;
 stROI.right = 150;
 CRectRoi->SetRoiI
//Open images from disk (or get
//image data from frame grabber)
 C8BitImage1->
   OpenBMPFile(“image1.bmp”);
 C8BitImage2->
   OpenBMPFile(“image2.bmp”);
 CFloatImage->
   OpenBMPFile(“image3.bmp”);

ImageCord(
   (VOID*)&stROI);
//Perform bitwise AND;
//Image3 = Image1 AND Image2.
CArith.LogicalAND(C8BitImage1,
   C8BitImage2, CFloatImage,
   CRectRoi,ARITH_ABS_VALUE,
   //Do not threshold but take
   //absolute value
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold values unused
   0);

//Save output to disk
 CFloatImage->SaveBMPFile
   (“output.bmp”);
```

Example (cont.)

```
//Free memory
delete C8BitImage1;
delete C8BitImage2;
delete CFloatImage;
delete CRectRoi;
}
```

LogicalOR

Syntax

```
int LogicalOR(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files

C_Arith.h

Description

Performs the following operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1
    | CImageIn2) + fOffset
CImageOut = |CImageOut|
//Threshold CImageOut so that:
   fLowThreshold <= CImageOut <=
   fHiThreshold
```

Description (cont.)

This method performs a logical bitwise OR operation with input image 1 and input image 2 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Bitwise ORs the two images (as 32-bit integers).
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.

If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes All values are converted to 32-bit integers before performing a logical bitwise OR operation.

Notes (cont.) This method uses images derived from the GLI/2 supplied CcImage class. These include binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses a ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImage256*C8BitImage2;
//8-Bit grayscale Image 2
CcGrayImageFloat*CFloatImage;
//Floating point grayscale Image
CcRoiRect* CRectRoi;
//Where operation will take place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/
//Allocate memory for objects
 C8BitImage1 =
   new CcGrayImage256( );
 C8BitImage2 =
   new CcGrayImage256( );
 CFloatImage =
   new CcGrayImageFloat( );
```

Example (cont.)

```
CRectRoi = new CcRoiRect( );

//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->SetRoiImageCord(
   (VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
 C8BitImage1->OpenBMPFile(
   “image1.bmp”);
 C8BitImage2->OpenBMPFile(
   “image2.bmp”);
 CFloatImage->OpenBMPFile(
   “image3.bmp”);
//Perform bitwise OR; Image3 =
//Image1 OR Image2.
CArith.LogicalOR(C8BitImage1,
   C8BitImage2,CFloatImage,
   CRectRoi,
   0,//Do not threshold or
   //absolute value
   1,//Set gain to 1
   0,//Set offset to 0
   0,//Threshold values unused
   0);

//Save output to disk
CFloatImage->SaveBMPFile(
   “output.bmp”);
```

Example (cont.)

```
//Free memory
delete C8BitImage1;
delete C8BitImage2;
delete CFloatImage;
delete CRectRoi;
}
```

LogicalXOR

Syntax

```
int LogicalXOR(
  CcImage* CImageIn1,
  CcImage* CImageIn2,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files

C_Arith.h

Description

Performs the following operation with respect to the given ROI (*CRoi*):

```
CImageOut = fGain * (CImageIn1 ^
   CImageIn2) + fOffset.
CImageOut = |CImageOut|.
//Threshold CImageOut so that:
   fLowThreshold <= CImageOut <=
   fHiThreshold.
```

Description (cont.)

Performs a logical bitwise exclusive-OR with input image 1 and input image 2 with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

The order of operations for this method is as follows:

1. Bitwise XORs the two images (as 32-bit integers).
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageIn2

Description: Image derived from class CcImage and used as image input 2 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is used unless it was specified by ARITH_THRESHOLD.

Notes All values are converted to 32-bit integers before performing a logical bitwise XOR operation.

Notes (cont.) This method uses images derived from the GLI/2 supplied CcImage class. These include binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses a ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImage256*C8BitImage2;
//8-Bit grayscale Image 2
CcGrayImageFloat*CFloatImage;
//Floating point grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/
//Allocate memory for objects
C8BitImage1 =
   new CcGrayImage256( );
C8BitImage2 =
   new CcGrayImage256( );
```

Example (cont.)

```
CFloatImage =
   new CcGrayImageFloat( );
CRectRoi =
   new CcRoiRect( );

//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->
   SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get
//image data from frame grabber)
C8BitImage1->
   OpenBMPFile(“image1.bmp”);
C8BitImage2->
   OpenBMPFile(“image2.bmp”);
 CFloatImage->
   OpenBMPFile(“image3.bmp”);
//Perform bitwise XOR; Image3 =
//Image1 XOR Image2.

CArith.LogicalXOR(C8BitImage1,
   C8BitImage2, CFloatImage,
   CRectRoi,0,
   //Do not threshold or absolute
   //value
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold values unused
   0);
//Save output to disk
 CFloatImage->SaveBMPFile(
   “output.bmp”);
```

Example (cont.)

```
 //Free memory
delete C8BitImage1;
delete C8BitImage2;
delete CFloatImage;
delete CRectRoi;
}
```

Copy/CopyRGB/CopyHSL

Syntax

```
int Copy(
  CcImage* CImageIn1,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  int iFlag,
  float fGain,
  float fOffset,
  float fLowThreshold,
  float fHiThreshold);
```

Include Files C_Arith.h

Description Performs the following operation with respect to the given ROI (CRoi):

```
CImageOut = fGain * (CImageIn1) +
   fOffset
CImageOut = |CImageOut|
Threshold CImageOut so that:
   fLowThreshold <= CImageOut <=
   fHiThreshold
```

This method copies input image 1 into the output image with respect to the given ROI. It takes the absolute value of the resulting data, if it was specified in the *iFlag* parameter. It also performs thresholding on the resulting data, if it was specified in the *iFlag* parameter.

Description (cont.) The order of operations for this method is as follows:

1. Copies the input image to the output image.
2. Applies gain and offset.
3. If specified, takes the absolute value of the resulting data.
4. If specified, thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Parameters

Name: CImageIn1

Description: Image derived from class CcImage and used as image input 1 in the equation.

Name: CImageOut

Description: Image derived from class CcImage and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iFlag

Description: Specifies the extra actions to take. The following values can be combined using the bitwise OR operator:

- ARITH_ABS_VALUE – Takes the absolute value of the resulting data.
- ARITH_THRESHOLD – Thresholds the resulting data to between *fLowThreshold* and *fHiThreshold*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Name: fHiThreshold

Description: High threshold limit; this value is not used unless it was specified by ARITH_THRESHOLD.

Notes

CopyRGB( ) and **CopyHSL( )** are identical to **Copy( )**, except that they perform the operation on all three color planes at once.

This method uses images derived from the GLI/2 supplied CcImage class. These include binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses an ROI derived from the GLI/2 supplied CcRoiBase class. These include all GLI/2 ROIs. It also works with your own images or ROIs derived from these classes.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256*C8BitImage1;
//8-Bit grayscale Image 1
CcGrayImageFloat*CFloatImage;
//Floating point grayscale Image
CcRoiRect* CRectRoi;
//Where operation takes place
CcArithmeticCArith;
//Object to perform operation
/*End of Dec Section*/

//Allocate memory for objects
C8BitImage1 =
   new CcGrayImage256( );
CFloatImage =
   new CcGrayImageFloat( );
CRectRoi = new CcRoiRect( );
//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->SetRoiImageCord((VOID*)&
   stROI);
//Open images from disk (or get
   image data from frame grabber)
C8BitImage1->OpenBMPFile(
   “image1.bmp”);
CFloatImage->OpenBMPFile(
   “output.bmp”);
```

Example (cont.)

```
//Perform copy; Output Image =
//Image1.
CArith.Copy(C8BitImage1,
   CFloatImage, CRectRoi, 0,
   //Do not threshold or absolute
   //value
   1, //Set gain to 1
   0, //Set offset to 0
   0, //Threshold values unused
   0);

//Save output to disk
CFloatImage->SaveBMPFile(
   “output.bmp”);
//Free memory
delete C8BitImage1;
delete CFloatImage;
delete CRectRoi;
}
```

4

Using the AVI Player Tool API

Overview of the AVI Player Tool API

The API for the AVI tool has one object only: the CcAVI class. This class allows you to read and write AVI files. In addition, you can open an existing .AVI file and read frames one at a time from the file into a custom application and you can create a new AVI file and write frames to this file from within a custom application. The CcAVI class does not allow you to edit .AVI files.

The CcAVI class uses a standard constructor and destructor and the class methods listed in Table 14.

Table 14: CcAVI Class Methods

Method Type	Method Name
Constructor & Destructor Methods	CcAVI(void);
	~CcAVI(void);
CcAVI Class Methods	bool SetColorImageType(int iType);
	int Open(LPCSTR szFileName);
	int Create(LPCSTR szFileName, int iImageType, int iWidth, int iHeight);
	int Close();
	BOOL IsOpenForReading();
	BOOL IsOpenForWriting();
	int GetFrameCount(int *piFrameCount);
	int GetFrameDimensions(int *piWidth, int *piHeight);
	int GetFrameType(int *piFrameType);

Table 14: CcAVI Class Methods (cont.)

Method Type	Method Name
CcAVI Class Methods (cont.)	int GetCompatibleImage(CcImage **ppImage);
	int ReadFrame(CcImage *pImage, int iFrame);
	int WriteFrame(CcImage *pImage);

CcAVI Member Methods

This section describes each method of the CcAVI class in detail.

SetColorImageType

Syntax

```
bool SetColorImageType(int iType
   );
```

Include Files C_Avi.h

Description Specifies the type of color image to generate.

Parameters

Name: iType

Description: Defines the type of color image. Use DEF_COLOR_RGB for RGB images; use DEF_COLOR_HSL for HSL images.

Notes Images are stored on disk in RGB format; however, when a file is accessed, you can import the images into GLI/2 in either RGB or HSL format.

Return Values

TRUE Successful.

FALSE Unsuccessful.

Example The following is a sample code fragment:

```
CcAVI AVI; // AVI object instance.
bool Result;
//Set the image type to RGB
Result = AVI.SetColorImageType(
   DEF_COLOR_RGB);
```

Example (cont.)

```
if (!Result)
{
// Operation failed - handle error.
}
```

Open

Syntax

```
int Open(LPCSTR szFileName
   );
```

Include Files C_Avi.h

Description Opens an existing .AVI file.

Parameters

Name: szFileName

Description: A pointer to a zero-terminated character string that contains the fully qualified name (path plus file name) of the .AVI file to open.

Notes None

Return Values

< 0 The AVI file could not be opened.

0 Successful.

Example The following is a sample code fragment:

```
CcAVI AVI; // AVI object instance.
int Result;

Result = AVI.Open("C:\\MyAviFiles
   \\MyAviFile.avi");
if ( Result < 0 )
{
// Operation failed - handle error.
}
```

Create

Syntax

```
int Create(
   LPCSTR szFileName,
   int iImageType,
   int iWidth,
   int iHeight
   );
```

Include Files C_Avi.h

Description Creates a new .AVI file with the specified file name. The file can store images of the specified image type, width, and height.

Parameters

Name: szFileName

Description: A pointer to a zero-terminated character string that contains the fully qualified name (path plus file name) of the .AVI file to create.

Name: iImageType

Description: The type of image to write to the .AVI file. It can be either IMAGE_TYPE_08BIT_GS, IMAGE_TYPE_24BIT_RGB, or IMAGE_TYPE_24BIT_HSL. No other image types are currently supported.

Name: iWidth

Description: The width of the images that will be written to the .AVI file.

Name: iHeight

Description: Specifies the height of the images that will be written to AVI file.

Notes None

Return Values

< 0 The .AVI file specified by *szFileName* cannot be created, an unsupported image type was specified for *iImageType*, or *iWidth* and / or *iHeight* have negative values.

0 Successful.

Example The following is a sample code fragment:

```
CcAVI AVI; // AVI object instance.
int iImageType =
   IMAGE_TYPE_08BIT_GS;
int iWidth = 640;
int iHeight = 480;
int iResult;
// Create a new AVI file that can
// hold 8-bit grayscale images with
// dimensions 640x480.
iResult = AVI.Create
   (“C:\\MyAviFiles\\
       MyAviFile.avi”,
       iImageType, iWidth,
       iHeight);
if ( iResult < 0 )
{
// Operation failed - handle error.
}
```


Close

Syntax

```
int Close(
   );
```

Include Files C_Avi.h

Description Closes an .AVI file that was opened using the **Open** method or created using the **Create** method.

Parameters None

Notes None

Return Values

< 0 No .AVI file is open.

0 Successful.

Example The following is a sample code fragment:

```
CcAVI AVI; // AVI object instance.
int Result;

Result = AVI.Close();
if ( Result < 0 )
{
// Operation failed - handle error.
}
```

IsOpenForReading

Syntax

```
BOOL IsOpenForReading(
   );
```

Include Files C_Avi.h

Description Determines whether an .AVI file is open for reading.

Parameters None

Notes An .AVI file that is loaded using a call to the **Open** method is considered to be open for reading. An .AVI file can either be open for reading or open for writing, but not both. Therefore, a program cannot write frames to an .AVI file if it was loaded using a call to **Open**. To write to an .AVI file, a new file must first be created using the **Create** method.

Return Values

FALSE The current AVI file is not open for reading.

TRUE The current AVI file is open for reading.

Example The following is a sample code fragment:

```
// Image object for reading from
// AVI file.
CcImage *pImage;
CcAVI AVI; // AVI object instance.
int Result;
Result = AVI.IsOpenForReading();
if ( Result == TRUE )
{
// Get an image object that is
// compatible with the frames in
// the AVI file.
if ( AVI.GetCompatibleImage
   ( &pImage ) < 0 )
{
// Operation failed - handle error.
}

// Read frame zero from the AVI
// file.
```

Example (cont.)

```
if (AVI.ReadFrame(pImage, 0) < 0)
{
// Operation failed - handle error.
}
// Delete image object when done.
delete pImage;
}
```

IsOpenForWriting

Syntax

```
BOOL IsOpenForWriting(
   );
```

Include Files C_Avi.h

Description Determines whether an .AVI file is open for writing.

Parameters None

Notes An .AVI file that is created using a call to the **Create** method is considered to be open for writing. An .AVI file can either be open for reading or open for writing, but not both. Therefore, a program cannot read frames from an .AVI file if it was created using a call to **Create**. To read from an .AVI file, an existing file must first be opened by calling the **Open** method.

Return Values

FALSE The current AVI file is not open for writing.

TRUE The current AVI file is open for writing.

Example The following is a sample code fragment:

```
// Image object for writing to
// AVI file.
CcImage *pImage;
CcAVI AVI; // AVI object instance.
int Result;
Result = AVI.IsOpenForWriting();
if ( Result == TRUE )
{
// Get an image object that is
// compatible with the AVI file
// format.

if ( AVI.GetCompatibleImage(
   &pImage ) < 0 )
{
// Operation failed - handle error.
}

// Add data to image object ...
// Write image to AVI file.
if ( AVI.WriteFrame( pImage ) < 0 )
{
// Operation failed - handle error.
}

// Delete image object when done.
delete pImage;
}
```

GetFrameCount

Syntax

```
int GetFrameCount(
   int *piFrameCount
   );
```

Include Files C_Avi.h

Description Returns the number of frames in the current .AVI file.

Parameters

Name: piFrameCount

Description: A pointer to the integer that contains the frame count.

Notes None

Return Values

< 0 The frame count cannot be obtained from the current .AVI file, and / or the input argument is NULL.

0 Successful.

Example The following is a sample code fragment:

```
// Holds frame count.
int iFrameCount;
int iResult;
CcAVI AVI; // AVI object instance.

iResult = AVI.GetFrameCount
   ( &iFrameCount );
if ( iResult < 0 )
{
// Operation failed - handle error.
}
```

GetFrameDimensions

Syntax

```
int GetFrameDimensions(
   int *piWidth,
   int *piHeight
   );
```

Include Files C_Avi.h

Description Returns the dimensions of the frames in the current .AVI file.

Parameters

Name: piWidth

Description: A pointer to the integer that contains the frame width.

Name: piHeight

Description: A pointer to the integer that contains the frame height.

Notes None

Return Values

< 0 The frame dimensions cannot be obtained from the current .AVI file, and / or one or more of the input arguments is NULL.

0 Successful.

Example The following is a sample code fragment:

```
// Holds frame width and height.
int iWidth, iHeight;
int iResult;
CcAVI AVI; // AVI object instance.

iResult = AVI.GetFrameDimensions
   ( &iWidth, &iHeight );
```

Example (cont.)

```
if ( iResult < 0 )
{
// Operation failed - handle error.
}
```

GetFrameType

Syntax

```
int GetFrameType(
   int *piFrameType
   );
```

Include Files C_Avi.h

Description Returns the type of the frames in the current .AVI file.

Parameters

Name: piFrameType

Description: A pointer to the integer that contains the frame type. Possible types are IMAGE_TYPE_08BIT_GS, IMAGE_TYPE_24BIT_RGB, and IMAGE_TYPE_24BIT_HSL.

Notes None

Return Values

< 0 The frame type cannot be obtained from the current .AVI file, and / or the input argument is NULL.

0 Successful.

Example The following is a sample code fragment:

```
int iFrameType;// Holds frame type.
CcAVI AVI; // AVI object instance.
int iResult;

iResult = AVI.GetFrameType
   ( &iFrameType );
if ( iResult < 0 )
{
// Operation failed - handle error.
}
```


GetCompatibleImage

Syntax

```
int GetCompatibleImage(
   CcImage **ppImage
   );
```

Include Files C_Avi.h

Description Returns an image object that is compatible with the format of the current .AVI file (compatible image type, width, and height).

Parameters

Name: ppImage

Description: A pointer to a pointer to a CcImage object that contains the newly created image object.

Notes You can use the image object returned by this method in subsequent calls to **ReadFrame**. Make sure that you free all image objects obtained through calls to this method.

Return Values

< 0 The frames in the .AVI file cannot be imported using images of type `IMAGE_TYPE_08BIT_GS`, `IMAGE_TYPE_24BIT_RGB`, `IMAGE_TYPE_24BIT_HSL`, and / or the input argument is NULL.

0 Successful.

Example The following is a sample code fragment:

```
// Image object for writing to
// AVI file.
CcImage *pImage;
CcAVI AVI; // AVI object instance.
int Result;
Result = AVI.GetCompatibleImage
   ( &pImage );
if ( Result < 0 )
{
// Operation failed - handle error.
}

// Delete image object when done.
delete pImage;
```

ReadFrame

Syntax

```
int ReadFrame(
   CcImage *pImage,
   int iFrame
   );
```

Include Files C_Avi.h

Description Returns the specified frame from the current .AVI file and places it in the specified image object.

Parameters

Name: pImage

Description: A pointer to a pointer to a CcImage object that contains the frame returned from the .AVI file.

Name: iFrame

Description: The number of the frame that you want to return. The value can range from 0 to *n* – 1, where *n* is the total number of frames in the .AVI file.

Notes You can read frames only from .AVI files that are open for reading. Refer to **IsOpenForReading** for more information.

Return Values

< 0 The specified frame is invalid (out of range), and/or the input argument is NULL.

0 Successful.

Example The following is a sample code fragment:

```
// Image object for writing to
// AVI file.
CcImage *pImage;
CcAVI AVI; // AVI object instance.

// Get a compatible image object.
if ( AVI.GetCompatibleImage
   ( &pImage ) == 0 )
{
// Make sure AVI file is open for
// reading.
```

Example (cont.)

```
  if ( AVI.IsOpenForReading() )
  {
// Read frame 0 from the AVI file.
    if (AVI.ReadFrame(pImage, 0)
       < 0 )
    {
// Operation failed - handle error.
    }
  }
// Delete the image object.
delete pImage;
}
```

WriteFrame

Syntax

```
int WriteFrame( CcImage *pImage
   );
```

Include Files C_Avi.h

Description Writes the image in the specified image object to the current .AVI file.

Parameters

Name: pImage

Description: A pointer to a pointer to a CcImage object that you want to write to the .AVI file.

Notes The image is appended to the end of the .AVI file.

You can write images only to .AVI files that are open for writing. Refer to **IsOpenForWriting** for more information.

Return Values

< 0 The format of the specified image object is not compatible with that of the .AVI file, and/or the input argument is NULL.

0 Successful.

Example The following is a sample code fragment:

```
// Image object for writing to AVI
// file.
CcImage *pImage;
CcAVI AVI; // AVI object instance.

// Get a compatible image object.
if ( AVI.GetCompatibleImage
   ( &pImage ) == 0 )
{
// Make sure AVI file is open for
// reading.


  if ( AVI.IsOpenForWriting() )
  {
// Fill in image with data ...
// Write frame 0 from the AVI file.
    if (AVI.WriteFrame(pImage)
       < 0 )
    {
// Operation failed - handle error.
    }
  }

// Delete the image object.
delete pImage;
}
```

5

Using the Blob Analysis Tool API

Overview of the Blob Analysis Tool API

The API for the Blob Analysis tool uses several GLI/2 API objects. Therefore, it is recommended that you read Chapter 2 before reading this chapter.

The Blob Analysis API contains two objects: the CcBlobFinder class and the CcBlob class. The CcBlobFinder class uses a binary mask image to produce a list of CcBlob classes. You cannot create a CcBlob class directly; you must use a CcBlobFinder class to find and create blobs.

The CcBlobFinder class uses a standard constructor and destructor and the class methods listed in Table 15.

Table 15: CcBlobFinder Methods

Method Type	Method Name
Constructor & Destructor	CcBlobFinder(void);
	~CcBlobFinder(void);
CcBlobFinder Class Methods	int SetMinBlobSize(int iBlobSize);
	int GetMinBlobSize(void);
	int SetMaxBlobSize(int iBlobSize);
	int GetMaxBlobSize(void);
	int SetLevel(int iNewLevel);
	int GetLevel( );
	int FindChildren(int iFind);
	int GrowBlobs(CcImage* cInputImage, CcBinaryImage* cMaskImage,RECT *pstROI);
	CcList* GetBlobList(void);

The CcBlob class uses a standard destructor, but the constructor is private; the class methods are listed in Table 16.

Table 16: CcBlob Methods

Method Type	Method Name
Constructor & Destructors	–[a]
	~CcBlob(void);
CcBlob Class Methods	CcBlob* GetParent(void);
	RECT* GetBoundingRect(void);
	PIXELGROUPING* GetPerimeterPG(void);
	STCHAINCODE* GetPerimeterChainCode(void);
	int CalculateAllInfo(CcCalibration* CCalibration = NULL);
	STBLOBSTATS* GetBlobStats(void);
	CcRoiFreeHand* GetFreehandROI( );
	CcList* GetChildBlobList(void);
	int GetNumOfChildBlobs(void);
	int DelChildrenOnDestructor(BOOL bFlag);

a. The constructor is private.

Note that the Blob Finder object is very stack intensive and requires a large stack to grow large blobs. If you are using the Blob Analysis API in a custom tool with GLI/2, you are already attached to an application (GLI/2) with a large stack and do not need to do anything else. If you are writing a custom application, you need to increase the stack size for the application. If you are using Visual C/C++, you can do this easily using program settings under the Link tab in the output section as shown in Figure 1.

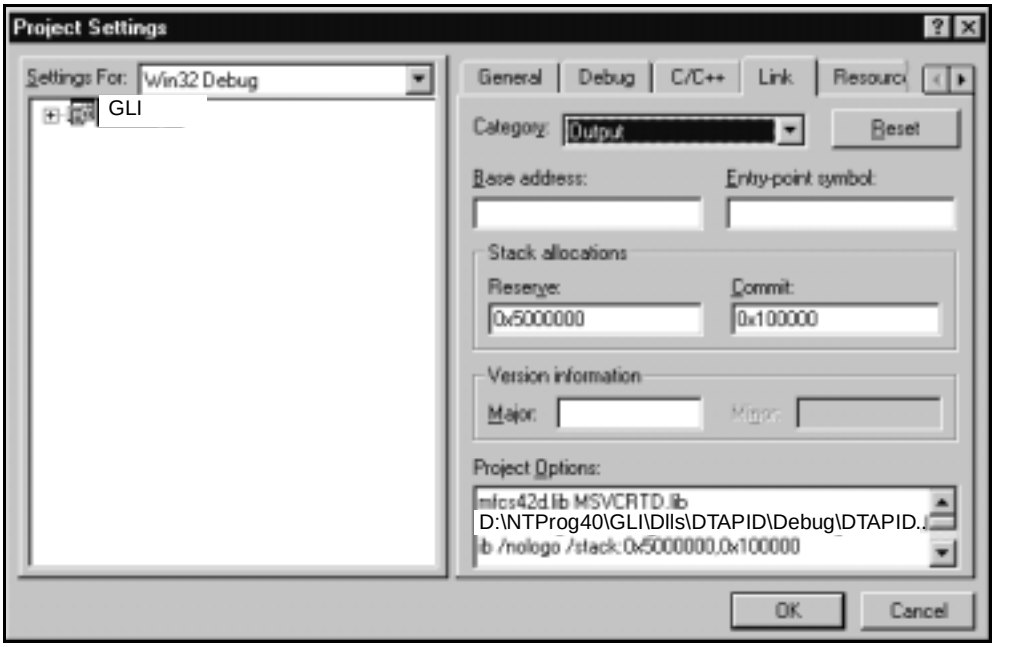

Figure 1: Program Settings

Set the reserve to something large such as 0x5000000 and the commit to something like 0x10000. This commits the stack to a large value but gives it room to grow, if needed, while growing a very large blob. In other applications, such as Visual Basic, which do not allow the stack to be changed, you can increase the stack size by using an MFC function call, such as **AfxBeginThread**.

CcBlobFinder Methods

This section describes each method of the CcBlobFinder class in detail.

SetMinBlobSize

Syntax `int SetMinBlobSize(int iBlobSize);`

Include File C_Blobf.h

Description Sets the minimum blob parent area (number of pixels) that a blob can have to be considered a blob. The blob is discarded if it has a parent area less than this value.

Parameters

Name: iBlobSize

Description: Minimum parent area to be considered a blob.

Notes The parent area is the area of the blob not including its children. It is the number of pixels in the blob, described in the *GLOBAL LAB Image/2 User's Manual.*

Return Values

–1 Unsuccessful.

0 Successful.

GetMinBlobSize

Syntax `int GetMinBlobSize(void);`

Include File C_Blobf.h

Description Gets the minimum blob parent area that a blob can have to be considered a blob. The blob is discarded if it has a parent area less than this value.

Notes The parent area is the area of the blob not including its children. It is the number of pixels in the blob, described in the *GLOBAL LAB Image/2 User's Manual.*

Return Values

–1 Unsuccessful.

0 Successful.

SetMaxBlobSize

Syntax `int SetMaxBlobSize(int iBlobSize);`

Include File C_Blobf.h

Description Sets the maximum blob parent area that a blob can have to be considered a blob. The blob is discarded if it has a parent area less than this value.

Parameters

Name: iBlobSize

Description: Maximum parent area to be considered a blob.

Notes The parent area is the area of the blob not including its children. It is the number of pixels in the blob, described in the *GLOBAL LAB Image/2 User's Manual.*

Return Values

–1	Unsuccessful.
0	Successful.

GetMaxBlobSize

Syntax `int GetMaxBlobSize(void);`

Include File C_Blobf.h

Description Returns the maximum blob parent area that a blob can have to be considered a blob. The blob is discarded if it has a parent area less than this value.

Notes The parent area is the area of the blob not including its children. It is the number of pixels in the blob, described in *GLOBAL LAB Image/2 User's Manual.*

Return Values

–1	Unsuccessful.
Maximum blob size.	Successful.

SetLevel

Syntax `int SetLevel(int iNewLevel);`

Include File C_Blobf.h

Description Sets the calculation level for the parameters that are calculated for the grown blobs.

Parameters

Name:	iNewLevel
Description:	A value between 0 and 4.

Notes If only a few blob options are desired, you can find blobs more quickly by using a lower calculation level. The lower the calculation level, the fewer parameters are calculated. By default, all parameters are calculated using a calculation level of 4.

Return Values

–1	Unsuccessful.
Maximum blob size.	Successful.

GetLevel

Syntax `int GetLevel(void);`

Include File C_Blobf.h

Description Returns the calculation level for the parameters that are calculated for the grown blobs.

Notes If only a few blob options are desired, you can find blobs more quickly by using a lower calculation level. The lower the calculation level, the fewer parameters are calculated. By default, all parameters are calculated using a calculation level of 4.

Return Values

–1	Unsuccessful.
Calculation level.	Successful.

FindChildren

Syntax `int FindChildren(int iFind);`

Include File C_Blobf.h

Description Specifies how child blobs are grown.

Parameters

Name: iFind

Description: Specifies one of the following values, which are defined in C_Blobf.h header file:

- 0 – **Bi-dir with children** – Child blobs are calculated and grown on both sides of the object.
- 1 – **Children** –Child blobs are calculated and grown on one side of the object.
- 2 – **No Children** – Child blobs are not grown or displayed. This option speeds up the overall growing of the blobs. This is useful if only the blob's perimeter ROI is important, if you don't care about child blob information, or if you know that there are no child blobs to be found.

Notes If you select **Bi-dir with children**, the found blobs display list shows levels of child blobs. If you select **Children**, the found blobs display list shows each child as a separate blob.

If more than one level of child blobs exists and you select **No Children**, the parameter totals do not include the information contained in the ungrown child blobs. In some cases, this is what is desired; in other cases, it may be an incorrect value.

Return Values

–1	Unsuccessful.
0	Successful.

GrowBlobs

Syntax

```
int GrowBlobs(
  CcImage* cInputImage,
  CcBinaryImage* cMaskImage,
  RECT* pstROI);
```

Include File C_Blobf.h

Description Finds the blobs within the given ROI.

Parameters

Name:	cInputImage
Description:	Pointer to the image in which you want to find blobs; it can be any image type.
Name:	cMaskImage
Description:	Pointer to a binary image to be used as a mask for finding the blobs.
Name:	pstROI
Description:	Pointer to a RECT structure used for the active ROI.

Notes The *cMaskImage* image is the same as the binary mask image described in the *GLOBAL LAB Image/2 User's Manual.*

Notes (cont.) The *pstROI* parameter is a pointer to a windows RECT structure and is most likely determined by an active RECT ROI class. The *cMaskImage* parameter is usually a thresholded resultant binary image of *cInputImage*. See Chapter 20 starting on page 649 for more information.

Return Values

–1	Unsuccessful.
0	Successful.

GetBlobList

Syntax CcList* GetBlobList(void);

Include File C_Blobf.h

Description Gets the list of CcBlob classes found after calling **GrowBlobs( )**.

Notes After creating the list of blobs by calling **GrowBlobs( )**, you can get a pointer to this list by calling this method. This method returns a CcList*. CcList* is a GLI/2 API-supplied class that contains a list of GLI/2 objects. You must cast any pointers returned by the CcList* methods. For more information on the CcList class, see page 11. For an example of how to use this method and a CcList class, see the example program at the end of this section.

The CcBlobFinder class always creates this list of blobs, but does not destroy the blobs or the list, since you want to free the memory for the CcBlobFinder class but use the newly found blobs.

Notes (cont.)

You are responsible for deleting both the list and all the blobs.

You can delete the list and all the blobs easily by setting up the returned list to delete its objects, and then deleting the returned list. By default, the list is NOT set up to delete all of its objects on its own destruction. However, all of its objects (the parent blobs) are set, by default, to delete all their children. Thus, by deleting the returned list, you can free all memory for all lists and all blobs created by the CcBlobFinder class.

Consider this example (also see the example code at the end of this chapter and the custom tool example “Blob1”):

```
CListBlob->SetDestructionType
   (LIST_DELETE_ON_DISTRUCTOR);
delete CListBlob;
```

Return Values

NULL	Unsuccessful.
Pointer to CcBlob classes.	Successful.

CcBlob Methods

This section describes each method of the CcBlob class in detail.

GetParent

Syntax CcBlob* GetParent(void);

Include File C_Blob.h

Description Returns a pointer to the parent of this blob, if it has one. If it is a top-level blob, the blob has no parent blob, and returns NULL.

Notes When finding blobs, the CcBlobFinder class finds all child blobs of all blobs. The level of child blobs is unlimited. Each parent blob has a list (a CcList) of all of its child blobs. Each child blob may also be a parent blob of yet another layer of blobs, and so on.

Return Values

NULL	Unsuccessful.
Pointer to this blob's parent blob.	Successful.

GetBoundingRect

Syntax `RECT * GetBoundingRect(void);`

Include File C_Blob.h

Description Returns a pointer to the bounding rectangle for this blob.

Notes A bounding rectangle is the smallest rectangle that totally encloses the blob's freehand ROI.

Return Values

NULL	Unsuccessful.
Pointer to this blob's bounding rectangle.	Successful.

GetPerimeterPG

Syntax

```
PIXELGROUPING* GetPerimeterPG(
   void);
```

Include File C_Blob.h

Description Gets a pointer to a pixel-grouping structure that describes the perimeter of the blob.

Notes This method returns the perimeter of the blob in the form of a pixel-grouping structure. The pixel-grouping structure groups pixels given in x, y coordinates starting from the lower left-hand corner of the image; these pixel comprise the perimeter of the blob.

Return Values

NULL	Unsuccessful.
Pointer to this blob's perimeter.	Successful.

GetPerimeterChainCode

Syntax

```
STCHAINCODE* GetPerimeterChainCode
   (void);
```

Include File C_Blob.h

Description Gets a pointer to a chain-code structure that describes the perimeter of the blob.

Notes A chain-code structure is an array of values that describes the chain-code of the perimeter.

Return Values

NULL	Unsuccessful.
Pointer to this blob's perimeter.	Successful.

CalculateAllInfo

Syntax

```
int CalculateAllInfo(
   CcCalibration* CCalibration =
   NULL);
```

Include File C_Blob.h

Description Calculates all blob information for the blob and its children.

Parameters

Name:	CCalibration
Description:	Pointer to a Calibration object.

Returned Values

–1	Unsuccessful.
0	Successful.

Example Blob information is described in detail in the *GLOBAL LAB Image/2 User's Manual*. This method calculates all the blob information that is given. If it is provided, a Calibration object is used to calculate all parameters in calibrated units. If a Calibration object is not given, all parameters are calculated in pixels.

Example (cont.)

```
struct STBLOBSTATS{
//Parent Information
float fParentArea;
float fParentXCentroid;
float fParentYCentroid;

in iParentNumOfPixels;

float fROIArea;
float ParentAreaToROIRatio;

float fParentGrayAverage;
float fParentRedAverage;
float fParentGreenAverage;
float fParentBlueAverage;

float fYatMaxX;
float fYatMinX;
float fXatMaxY;
float fXatMinY;

float fXDifference;
float fYDifference;
float fBoundingBoxArea;
float fParentBoxRatio;

//Child(Hole) Info
int iNumOfChildren;
float fChildArea;
float fTotalArea;
float fChildRatio;
int iTotalNumOfPixels;

float fTotalXCentroid;
float fTotalYCentroid;
```

Example (cont.)

```
float fParentGrayTotal;
float fParentRedTotal;
float fParentGreenTotal;
float fParentBlueTotal;

int iParentSumX;
int iParentSumXX;
int iParentSumXY;
int iParentSumY;
int iParentSumYY;

float fMaxX;
float fMaxY;
float fMinX;
float fMinY;

float fTotalAreaToROIRatio;

float fTotalGrayAverage;
float fTotalRedAverage;
float fTotalGreenAverage;
float fTotalBlueAverage;
float fTotalGrayTotal;
float fTotalRedTotal;
float fTotalGreenTotal;
float fTotalBlueTotal;
int iTotalSumX;
int iTotalSumXX;
int iTotalSumXY;
int iTotalSumY;
int iTotalSumYY;
float fTotalBoxRatio;

//Perimeter Info
float fPerimeter;
float fXPerimeter;
```

Example (cont.)

```
float fYPerimeter;
float fRoundness;
float fPPDA;

//Center of Mass Info
float fAvgRadius;
float fMaxRadius;
float fMinRadius;
float fCDistance;
float fMaxRadiusAngle;
float fMinRadiusAngle;
float fDiffRadiusAngle;
float fRadiusRatio;
};
```

GetBlobStats

Syntax

```
STBLOBSTATS* GetBlobStats(void);
```

Include File

C_Blob.h

Description

Returns a pointer to the blob information structure.

Notes

All blob information calculated is returned to the calling program using a pointer to a structure called STBLOBSTATS. This structure contains valid information only if you have called **CalculateAllInfo( )**, described on page 281.

For a detailed view of the structure STBLOBSTATS, view the header file C_Blob.h found in the \GLI\Include subdirectory.

Return Values

NULL	Unsuccessful.
Pointer to the blob information.	Successful.

GetFreehandROI

Syntax

```
CcRoiFreeHand*
   GetFreehandROI(void);
```

Include File C_Blob.h

Description Creates and returns a pointer to a freehand ROI that outlines the perimeter of the blob.

Notes The first time it is called, this method creates a new freehand ROI object that describes the perimeter of the blob. Each additional time this method is called, the same freehand ROI pointer is returned.

You are responsible for freeing the memory for this ROI. For example, if you call this method twice, you will receive the same freehand ROI pointer to the same ROI object. You must free the memory for this ROI object only once.

Once you delete the ROI object that is returned to you using this method, you should not call this method again. If you do call this method after you deleted the ROI object, the same pointer is returned but the pointer is invalid since you deleted the object.

Return Values

NULL	Unsuccessful.
Pointer to a freehand ROI.	Successful.

GetChildBlobList

Syntax `CcList* GetChildBlobList(void);`

Include File C_Blob.h

Description Returns a pointer to the list of child blobs for this blob.

Notes All blobs contain a list of its child blobs. This list is a GLI/2 API object called a CcList. If it has no children, the blob still has a list of child blobs but the list is empty. Remember that this environment is object-oriented, and thus, a blob is a blob. A blob can be viewed as both a parent to its children and a child of its parent. A top-level blob does not have a parent.

If you delete a blob, you must remove the blob from its parent's list (if it has a parent) and delete all of the blob's children to avoid memory leaks. If you do not remove the blob from its parent's list, the system could crash if the parent tries to use this blob.

If you delete a blob and do not delete its child blobs, memory leaks occur because these children are normally deleted by their parent when the parent is deleted.

Notes (cont.) A simple way to remove a blob from its parent list is to use the list to delete the blob. A CcList object, by default, deletes the object if the object is removed from the list using any of the delete methods. Also, the blob deletes all of its children, by default.

Return Values

NULL	Unsuccessful.
Pointer to a CcList of child blobs.	Successful.

GetNumofChildBlobs

Syntax `int GetNumOfChildBlobs(void);`

Include File C_Blob.h

Description Returns the total number of child blobs that belong to this blob.

Notes The total number of child blobs refers to all levels (descendants) of blobs under this blob, not just this blob's immediate children. If you want the immediate number of children that belong to this blob, you must first get a pointer to the list of child blobs, and then ask the list how many children it contains.

CcList is a GLI/2 API object.

Returned Value

NULL	Unsuccessful.
Number of child blobs that belong to this blob.	Successful.

Example This example returns both the number of immediate children for the blob *CThisBlob* and the total number of descendants for the blob *CThisBlob*.

```
void GetChildren(CcBlob CThisBlob,
   int* iAllChildren,
   int* iChildren)
{
CcList* CChildList;
//Return the total number of
//descendants for the blob in the
//variable iAllChildren.

 *iAllChildren = CThisBlob->
   GetNumOfChildBlobs( );

//Return the immediate number of
//children for the blob in the
//variable iChildren
//Get a pointer to the list of
//child blobs

 CChildList = CThisBlob->
    GetChildBlobList( );

//Ask the list how many children
//are in the list

*iChildren = CChildList->
   GetNumberOfObjects( );
}
```

DeleteChildrenOnDestructor

Syntax

```
int DelChildrenOnDestructor(
   BOOL bFlag);
```

Include File C_Blob.h

Description Determines if the child blobs are deleted when the parent blob is deleted.

Parameters

Name: bFlag

Description: Sets a flag to one of the following:

- TRUE – Deletes all children of this blob (default).
- FALSE – Does not delete child blobs.

Notes By default, you need only delete the parent blob's list to free all memory for all blobs found by the CcBlobFinder class. You can do this easily by deleting the list containing the top-level parent blobs that are returned by the class CcBlobFinder. If you wish, you can delete the blobs yourself by telling each blob not to delete its children using this method.

Returned Value

–1 Unsuccessful.

0 Successful.

Example Program Using the Blob Analysis Tool API

This example program takes a binary mask image (CImageMask) and find all the blobs greater than 30 pixels in the given ROI (CRoi). The roundest blob's value (considering parents only) is returned.

Note: This example is made from code fragments from the Blob Analysis tool with error checking removed. In an actual program, you should check return values and pointers.

The code for this example is also implemented as a custom tool under \GLI\Tools\Examples\Blob1.

```
float FindRoundestBlob(CcImage* CImageIn,
   CcBinaryImage* CImageMask, CcRoiBase* CRoi)
{
CcList* CListBlob;
//List of child blobs found by the CcBlobFinder
class
int x;
float fRoundness;
CcBlob* CBlob;
STBLOBSTATS* stInfo;

//Check type of Mask image to be a Binary image
if(CImageMask->GetImageType( ) !=
IMAGE_TYPE_BINARY)
 {
 ::MessageBox(::GetFocus( ),"The Mask Image must be
    a Binary Image","Error",MB_OK);
 return(-1);
 }
```

```
//Check type of ROI to be a Rectangular ROI
if(CRoi->GetROIType( ) != ROI_RECT)
  {::MessageBox(::GetFocus( ),"ROI must be a
   Rectangular ROI","Error",MB_OK);
   return(-1);}

//FIND BLOBS
//Create a new blob finder class
CcBlobFinder* CBlobFinder = new CcBlobFinder( );
//Set blob parameters
CBlobFinder->SetMinBlobSize(30);
//Find
CBlobFinder->GrowBlobs(CImageIn,CImageMask,
   (RECT*)CRoi->GetRoiImageCord( ));
//Get pointer to list of found blobs
CListBlob = CBlobFinder->GetBlobList( );
//Free memory for blob finder class
delete CBlobFinder;
 //Find Roundest blob - search parent blobs only
fRoundness = -1;
for(x=0; x<CListBlob-> GetNumberOfObjects( ); x++)
 {

//Get next blob in list - do not forget to cast
pointer!
   CBlob = (CcBlob*)CListBlob->GetAtIndex(x);
   //Calculate all information for blob
   CBlob-> CalculateAllInfo( );
   //Get pointer to information structure
   stInfo = CBlob-> GetBlobStats( );
   //Save roundest blob value
   if(fRoundness < stInfo-> fRoundness)
    fRoundness = stInfo-> fRoundness;
}
```

```
//Free memory created by CcBlobFinder class
//Set Destruction type to delete all elements in
the list
CListBlob->SetDestructionType(LIST_DELETE_ON_
   DISTRUCTOR);
delete CListBlob;
return(fRoundness);
}
```

6

Using the Custom Script Tool API

Introduction

The Custom Script tool was created as a general-purpose programming tool for nonprogrammers. Emphasis is placed on performing a number of complex tasks very easily. In keeping with the ease-of-use philosophy, the Custom Script tool is a program interpreter rather than a compiler. Interpreters are programs that read commands as text, executing them as encountered. To change a program requires editing the command or program file only.

Compilers, on the other hand, read program text files and write a file of computer instructions to disk. Changes to compiled programs are more time consuming and the level of knowledge to create even simple tasks is many magnitudes greater than that of a Custom Script program. However, compiled programs offer a slight program performance and also offer a much wider range of programming possibilities. If you wish to create a compiled program, see the GLI/2 API, described in Chapter 2 starting on page 11. The GLI/2 API is an object-oriented API that you can use with Visual C/C++.

As an interpreted programming language, the Custom Script tool processes instructions one line at a time directly from the program file. Minimal processing is done to check for errors in logic or syntax, since speed is always a major concern in imaging applications. Generally speaking, interpreters, such as the Custom Script tool and the BASIC interpreter, which comes with most PCs, are easier to use and more forgiving than programs that are processed through other methods. Therefore, most novice and casual programmers find working with an interpreter less frustrating and more productive. Some loss of capabilities and speed occurs as compared to programs that are compiled, but when used appropriately, these conditions have little or no impact. Experience will determine which method is better for your use.

The Custom Script tool provides the following features:

- It is easy to use and understand;
- It is flexible;
- It provides minimal error checking;
- It provides automatic conversion from one data type to another (see page 296); and
- It provides simplified interfaces for vision and motion.

Anatomy of a Typical Custom Script Program

The Custom Script tool features items such as structured blocks, flow control, variable data types, and a file management system. A unique command set consisting of keywords and functions is provided for fast code execution and for performing a wide variety of tasks.

The Custom Script tool allows for global variables, redefineable data types, automatic redimensioning, and dynamic label creation. Data types can be declared anywhere in the main body of the program or in subroutines. Variable types can be explicitly declared or can be defined by first usage, as follows:

```
X = REAL ! Explicit real
I = 5 ! Implicit integer
S = "TEXT" ! Implicit string
```

Variables can be either uppercase or lowercase. Names of keywords and functions form a reserved word list and cannot be used as data variables in the body of the program. Singly subscripted arrays can be declared by using brackets [ ]. The first subscript always begins with 0 and the value contained in the declaration statement determines the maximum size, shown as follows:

```
X[4] = INTEGER
```

In this example, *X* is an array of four integers whose subscripts are 0, 1, 2, and 3. One very important feature to note is that any reference to a subscript that is out of bounds for the current maximum array element resizes the array and reinitialize all elements, in this case to 0. A reference to X[4] in the example resizes the X array to 5 elements and sets X[0] through X[3] to 0.

Custom Script programs look much like BASIC programs and usually have one statement of code per line. Exclamation points define a comment statement. Programs are executed by selecting the Custom Script tool from the GLI/2 tool box, and then entering the script file name.

Because the Custom Script tool ignores white space (tabs, blanks, and line breaks) in your program file, you have the freedom to arrange your code in almost any style. However, most programmers follow a few de facto rules that have evolved to promote readability. A typical program file must have one complete statement per line. It is common practice to indent statements inside looping functions so they are vertically aligned. Directives may also be given to the Custom Script tool to perform nonexecutable functions, such as including files.

Data Types

Unfortunately, computers and computer languages are not very intelligent. For example, we know that the number one can be represented as either a 1 or 1.0 or 1.00.To a computer, 1 and 1.0 are two distinctly different types of data. A whole number is an integer data type. An integer falls into the range of +32767 to –32768. If the whole number is larger than this, the data type is a long. A long data type has a range of +2147483647 to –2147483648. Numbers larger than this or numbers containing decimal points are called real numbers. Characters are represented by the string data type. Textlists are arrays of strings. There is also a file pointer data type, called FILE, which is used internally, and a data type for hexadecimal numbers, called HEXNUM, which can be declared by using a dollar sign ($).

These basic data types and their sizes are summarized as follows:

- UNSIGNED – ! 16 bits
- INTEGER – ! 16 bits
- LONG – ! 32 bits
- REAL – ! 64 bits
- STRING – ! Variable
- TEXTLIST – ! Variable
- FILE – ! Internal use only
- HEXNUM – ! 8 characters, $FFFFFFFF

In the Custom Script tool, most of the transactions between data types are transparent to both the programmer and the operator. This is done according to the following set of rules:

- Explicitly programmed data is converted to the INTEGER data type if the data does not begin and end with a double quote, does not contain a period, does not start with a $, and contains only numbers.
- If the data begins and ends with a double quote, or has no quotes but has characters other than numbers, the data is converted to the STRING data type.
- If the data starts with $, the data is converted to the HEXNUM data type.
- If no beginning and ending double quotes are present, and the data consists of a period and numbers only, the data is converted to the REAL data type.
- If a variable is used for the first time, the variable is set to the type of data being assigned to it. (See the first rule.)

- If a variable is being assigned a value either through another variable, an intrinsic function, or from an explicitly programmed value and this variable already has a data type, then the incoming data is converted to the same type as the declared variable. For example, if variable S$ is of type STRING, S$ = “Result:” + (14.2*4) ends up as a the following string of text stored in the variable S$: Result: 56.8.
- Data can be lost or truncated when converting from a large data type to a smaller data type, such as LONG to INTEGER or REAL to LONG, or REAL to INTEGER.
- Data type setting for variables can be done at any time to specifically set the desired data type.To set a variable type, simply assign to it one of the supported data types. For example: num = REAL sets the data type to REAL for the variable *num* and removes any data that may have been stored in *num*. The variable *num* can be an existing variable of a different data type.
- A colon (:) indicates labels. Labels must be the first and only text on a line. Label names following a GOTO or a GOSUB function can be computed by enclosing the expression in parentheses, but the final evaluation must be a character string.

Operators

The Custom Script tool supports three basic groups of operators:

- Math,
- Logic, and
- String.

These operators are described in more detail in the following subsections.

Math Operators

Table 17 lists the math operators used by the Custom Script tool.

Table 17: Math Operators

Operator	Sample	Description
+	X+Y	Addition
-	X-Y	Subtraction
*	X*Y	Multiplication
/	X/Y	Division
^	X^2	Exponentiation
&	X&Y	Bitwise AND
¾	X|Y	Bitwise OR
~	~X	Bitwise complement
@	X @ Y	Bitwise exclusive OR
SHR	X SHR Y	X-shifted right by Y-positions (e.g. 8 SHR 2=2)
SHL	X SHL Y	X-shifted left by Y-positions (e.g. SHL 4=32)

Operators &, |, ~, @, SHR, and SHL are referred to as bitwise operators using Base 2. Table 18 shows examples for each of these operators.

Table 18: Examples of Bitwise Operators

Bitwise Operator	Example
&	9 & 5 = 1
|	3|8 = 11
@	11 @ 3 = 8
SHR	64 SHR 2 = 16
SHL	8 SHL 3 = 64

Like other languages, the Custom Script tool has precedence rules that determine the order of evaluations in expressions that contain more than one operator. Here are the general rules:

- When two operators have unequal precedence, the operator with the higher precedence is evaluated first.
- Operators with equal precedence are evaluated from left to right.
- You can change the normal order of precedence by enclosing an expression in parentheses.

When constructing calculations, it is important to consider exactly how the calculation should be done. By default, the Custom Script tool finds the inner most group of parentheses, solves that part of the calculation, and then moves on to the next inner most set of parentheses. When no more parentheses are found, the expression is processed from left to right, extracting the higher-order operators first for solution. This extraction process is called operator precedence. All math operators and all logical operators (see page 301) are assigned a precedence level. When processing a calculation, the significance or precedence determines the order of solution of each individual operator. Table 19 lists the five levels of precedence and the operators that are contained within each level.

Table 19: Operator Precedence

Level of Precedence	Operator					
Level 1	~	NOT				
Level 2	+	-	*	/	^	
Level 3	&	|	@			
Level 4	<	>	=	<=	>=	<>
Level 5	AND	OR	SHR	SHL	IN	

By examining how a given calculation should be solved, you can insert parentheses into the expression to force the Custom Script tool to solve the calculation in the intended manner.

For example, 2 + 3 * 3 and (2 + 3) * 3 results in 15. However, 2 + (3 * 3) results in 11, since multiplication is done first. Whenever in doubt, include parentheses.

Note: Do not rely on operator precedence alone for correct expression evaluations.

Logical Operators

Table 20 lists the logical operators used in the Custom Script tool.

Table 20: Logical Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
=	Equal
<>	Not equal
AND	Logical and
OR	Logical or
NOT	Logical not

Logical operators evaluate to a TRUE or FALSE result. The Custom Script tool treats 0 as FALSE and any nonzero value as TRUE. As shown in Table 19 on page 301, logical operators are among the last operators to be examined when evaluating an expression.

Although not always true, these operators are usually used as part of a conditional branching statement, **IF (...) THEN**, where the expression between the parentheses contains one or more logical operators. Some examples of usage are as follows:

```
IF (4>5) THEN! Hopefully this won't print.
MESSAGEBOX("4 IS GREATER THAN 5!!", "CUSTOM
   SCRIPT", MB_OK)
IF (X and Y) THEN
! Do if both X and Y are not zero GOTO DONE
! (0). TRUE can be a negative number or even a
! fraction.
```

```
IF(NOT EXIST(“SOMEFILE.DAT”)) THEN
MESSAGEBOX("FILE NOT FOUND","CUSTOM SCRIPT",MB_OK)
! Operators and keywords that
! return a value can be part of an expression. NOT
! is a single or UNARY operator.
! When using NOT with other logical operators, the
! other operator ALWAYS precedes the NOT.
! For example: X AND NOT Y is okay. However,
! X NOT AND Y is incorrect.
```

String Operators

Some special features were incorporated into the Custom Script tool to help manipulate strings. As just shown in the previous section, logical comparison of strings is an intrinsic capability. In addition, the plus (+) and minus (–) operators can also be used with strings. The plus sign is used to join or concatenate two or more strings. The minus sign is used to “subtract” the second string from the first, if the second string is in the first string.

For example: S= “I am” + “ happy to write this.” creates “I am happy to write this.” in the variable *S*. Or, S = “This is happy work.” – “happy” creates “This is work.” in the variable *S*. In addition, the keyword INSTR extracts any portion of a string.

The Custom Script tool also allows each individual character of a string to be read from or written to. By allowing this, characters can be converted to or from lowercase or changed entirely. To achieve this, the string is “subscripted” using the left and right brackets, [ ]. Between the brackets is the location in the string of the desired character. The Custom Script tool starts subscript numbering at 0. Therefore, if string variable *S* contains “TEST”, the first letter is addressed as S[0], while the last letter is addressed as S[3].

The value obtained when reading a subscripted string is an integer. So, if the string variable *S* contains “APPLE”, then S[0] returns the integer value 65. Consider the following examples:

```
Reading a string subscript.
S = “APPLE” ! I is now an integer variable with
value 65
I = S[0]! S[0] has the ASCII value of 65
MESSAGEBOX(“I=”,+I,"CUSTOM SCRIPT",MB_OK)
! This would print: I=65
MESSAGEBOX(“I=”,+CHR(I),"Custom Script",MB_OK)
! This would print: I=A
```

Writing to a subscript string variable also requires that an integer value be used, since the Custom Script tool does not have a single character type. Therefore, using the previous example, S[0]=97 results in *S* containing “APPLE”.

Think of the *S* array as follows:

Table 21: The S Array

S Element	External	Internal
S[0]	A	65
S[1]	P	80
S[2]	P	80
S[3]	L	76
S[4]	E	69

CAUTION:

It is imperative that no values are beyond the end of the string. Doing so, could result in catastrophic results!

Two ways are provided to stay within the length of a string. The first way is to obtain the string length using the keyword TEXTLEN or by checking the value read from the subscripted variable. If that value is 0, then this is one place past the end of the string. The following two sample programs convert a string variables' lower case characters to uppercase characters.

Case Conversion 2...

```
S = "This is the string to be converted to all
uppercase."
```

Case Conversion 1...

```
S = "This is the string to be converted to all
uppercase."
L = TEXTLEN(S)
POS=0
   WHILE(POS<L)
! Remember the last position is length-1.
      I=S[POS]
      IF(I>96 OR I<123)THEN
      ! Is this a lower case letter?
        S[POS] = I-32 ! Convert to upper case.
      WEND POS = POS + 1
! Increment POS by one –Special
! WEND feature.
:DONE
! Finished – DONE is a label as shown
! by the leading colon.
POS=0
REPEAT
I=S[POS]
   IF(I>96 OR I<123)THEN
! Is this a lower case letter?
     S[POS] =I-32 ! Convert to upper case.
     POS = POS+1 ! Increment POS.
```

```
UNTIL(S[POS]=0)
! Zero is one position past the last character.
```

The first program uses the length method while the second checks for a character value. Either method works fine, just be certain that nothing is written past the end of the string.

As shown, both examples perform the same operation.To a great degree, it is a matter of preference as to which method to use. The Custom Script tool provides the function **UPCASE** to convert from lowercase to uppercase letters.

Programming Considerations

This section describes things to consider when programming using the Custom Script tool.

Expressions

An expression is any combination of operators, variables, functions, and explicitly programmed values that return a value.

By way of definition, a variable is nothing more than a symbol that contains information. The programer writing the Custom Script program assigns variables. A variable is always assigned its value with an equal sign. Variable name lengths can be 32 characters or less (see page 296).

Some examples of variables are as follows:

```
NUMBER = 5 ! The value 5 is contained in the
!variable named NUMBER. Upper and lower case
!differences are ignored. So NUMBER, Number, and
!NUMber are all the same.

MESSAGEBOX("NUMBER =",+NUMBER,"CUSTOM
   SCRIPT",MB_OK)
! This line displays NUMBER= 5. Because NUMBER was
! not used before, the data type associated with
! NUMBER is INTEGER. This is because the
! value assigned did not have a leading and
! trailing quote and no decimal point was used in
! the value.
```

Explicitly programmed values are written into the Custom Script program. In the previous example, 5 is an explicit value.

Expressions always return a numerical value. Therefore, the result of the expression can be assigned to a variable, used in another expression, used by a keyword, or simply ignored. Expressions can be simple, such as 1 + 1, or they may be very complex with several groupings or parentheses. Custom Script executes faster when a single complex expression is used, rather than a series of simple expressions.

Expressions cannot exceed one text line. However, the Custom Script tool can read a line that is up to 255 characters long. If the expression being created is quite long, it may be necessary to split it into two or more expressions. Be sure that the resulting expressions still achieve what the correct operation. Sometimes it is easier to troubleshoot a program when expressions do not get overly complex. Once you feel confident that everything is working properly, combining some expressions together may speed up performance or may help clarify the program.

Some examples of different expressions are as follows:

Program 1:

```
X=3
Y=5
Z=((X & Y) | 8) ^ 3
! This program results in 729. X & Y = 1. 1 | 8 = 9.
! Finally 9 ^ 3 = 729, i.e., 9 cubed.
```

Program 2:

```
SETDP (6) ! Set displayed decimal places to 6.
DEGREE = (2*PI)/360
! Get the number of radians in one degree.
MESSAGEBOX("DEGREE:"+DEGREE,"",MB_OK)
!Print the value on the screen.
ANGLE=30 ! Set ANGLE to 30 degrees.
NUM=SIN(ANGLE*DEGREE) ! Get the SIN of 30 degrees.
MESSAGEBOX("SINE OF"+ANGLE+":"+NUM,"",MB_OK)
! Print the answer. SIN OF 30:0.500000.
```

Program 3:

```
! This program demonstrates logical comparisons
! using strings.
! All comparisons are based upon lexicographical
! (alphabetical) comparisons. Refer to a standard
! ASCII chart when analyzing the results from a
! string comparison.
S1="AAAA"
S2="ZZZZZZZZZ"
MESSAGEBOX("S1=S2:"+(S1=S2),"",MB_OK)
! This prints: S1=S2:0
MESSAGEBOX("S1<S2:"+(S1<S2),"",MB_OK)
! This prints: S1<S2:1
MESSAGEBOX("S1>S2:"+(S1>S2),"",MB_OK)
! This prints: S1>S2:0
```

```
MESSAGEBOX("S1<>S2"+(S1<>S2),"",MB_OK)
! This prints: S1<>S2:1
```

As can be seen by these sample programs, expressions are used in a variety of ways for a variety of purposes. Most often, expressions are used for evaluation in an **IF** statement or as part of a **MESSAGEBOX** statement when creating text to be displayed on the screen.

Branching

Branching stops program execution at the current line and restarts it at the point specified. Two keywords are used for branching: **GOTO** and **GOSUB**. **GOTO** is called an unconditional branch because it changes the program flow without regard to where it came from. On the other hand, **GOSUB** remembers the current program location and returns to the next line after that point when the keyword **RETURN** is encountered.

A label is any text that begins with a colon and contains any printable ASCII character except a space. When using the label name after a **GOTO** and **GOSUB**, a leading colon is not required. A **GOTO** and a **GOSUB** can reference the same label.

The following example illustrates the use of **GOTO** and **GOSUB**:

```
!AN EXAMPLE OF GOTOS AND GOSUBS
IF(MESSAGEBOX("PRESS OK TO SKIP",
   "CUSTOM SCRIPT",MB_OKCANCEL)=IDOK) THEN
   GOTO SKIP
ELSE
  GOSUB NO_SKIP
GOTO REDO
:NO_SKIP
 MESSAGEBOX("OK PRESSED","CUSTOM SCRIPT",MB_OK)
:SKIP
```

Be sure the label exists or an error is generated. If a **RETURN** keyword is encountered without a corresponding **GOSUB**, an error is generated.

The location of the label does not matter. However, the Custom Script tool does search forward to the end of the program and then continues from the beginning to find the label. Therefore, if the program is quite large, execution is slightly faster if the label is after the branching statement.

Looping

Loops repeatedly execute the same lines of program statements until some condition is satisfied. Two sets of keywords are used for loops: **REPEAT UNTIL** and **WHILE WEND**. As shown in earlier examples, these are two very useful functions.

The essential difference between **REPEAT** loops and **WHILE** loops is **REPEAT** checks the conditional statement after performing one pass through the loop and **WHILE** checks the conditional statement before performing any loop statements.

Any valid program statements may be used within a loop, including **GOTO** and **GOSUB**. When writing loops, just be sure that the loop can be exited!Two short examples demonstrate the differences:

WHILE example:

```
I = 0
DONE=0 ! Initialize a terminal variable.
WHILE(NOT DONE)
  WHILE(I < 100)
  I = I + 1
  WEND
  DONE= (I > 100)
WEND ! WHILE end.
```

REPEAT example

```
I = 0
REPEAT ! Enter the loop.
  WHILE(I < 50) ! WHILE in REPEAT is okay.
  I = I + 1
  WEND
  DONE=(1 > 50)
UNTIL (DONE)! Check terminal variable.
```

In certain circumstances it is desirable or necessary to either increment or decrement a variable by some value. In other programming languages this is accomplished with a **FOR .. NEXT** loop. Rather than adding additional keywords, the Custom Script tool has added additional capability to the **WEND** keyword. If a valid program statement follows the **WEND** keyword (on the same line), that statement is executed. Normally, this statement would be used to change the value of the variable, but it does not have to be used in this manner.

The following example illustrates the use of the **WEND** function:

```
I=33 ! Set an initial value.
  WHILE(I<500) ! Check the value of I
   WEND I=I+25 ! Increment I by 25 each pass.
```


Some previous examples also demonstrate the use of **WEND** in this manner. Be sure to initialize the terminal variable before starting the loop.

Date and Time

Two date keywords and two time keywords are provided for use in your programs. **DATE$** returns the current system date in text or STRING form. The format is always month, day, and year. For example, 1/1/97 is Jan 1, 1997. The STRING or text form of time is **TIME$**. The clock is a 24-hour clock, so 1 p.m. is hour 13. The format is: hh:mm:ss. In other words, the text for 2:45 p.m. is 14:45:00.

A numeric form for these two keywords are **DATE** and **TIME**. The date is returned in the form yyyymmdd, such as 19970101 for Jan 1, 1997. The returned value from **TIME** is the number of seconds since midnight. This is useful when timing an event or setting a timeout value for a response.

Trigonometric Functions

Three trigonometric functions are available: **SIN**, **COS**, and **TAN**. The value passed to these functions is an angle in radians. The value returned is the sine, cosine, or tangent of that angle. The intrinsic function **PI** is available to convert angles between radians and degrees. (There are 2 PI radians in 360 degrees. One radian equals 360 / (2*PI).)

Restrictions

Keep the following list in mind to avoid the common mistakes that can be made when creating a Custom Script program:

- Output of a variable before assignment of any kind results in that variable type to be defaulted to a real number.
- Since there is no level of hierarchy within any given level of precedence, use parenthesis freely to force evaluation in the way you intend it to be done. This is particularly true when using the arithmetic functions.
- Remember that X[4] is a four deep array having subscripts 0, 1, 2, and 3. Any reference beyond the maximum subscript, in this case 3, resizes the array to the new maximum value, and reinitialize all previous values.
- Do not write characters beyond the end of a string variable as adjacent memory locations may be overwritten. Certain functions also rely on the last character of a given text string to be a 0.
- Code runs faster using complex expressions. Construct and debug simple expressions, and then combine them.
- Be sure that the looping routines **REPEAT** and **WHILE** have a legitimate path for them to exit. Failure to do this results in your Custom Script program “hanging”.
- Make sure that all of your **GOTO**s point to a valid label: a text name that is preceded by a colon. Failure to do this results in a run-time error.

Keywords and Functions

Table 22 briefly describes the keywords and functions used in the Custom Script tool.

Table 22: Keywords and Functions in the Custom Script Tool

Function Type	Function Names	Descriptions
Math	ABS	Returns the absolute value of a number.
	COS	Returns the cosine of a value.
	SIN	Returns the sine of a value.
	TAN	Returns the tangent of a value.
	MEAN	Returns the statistical mean of a group of values.
	MEDIAN	Returns the statistical median of a group of values.
	KURTOSIS	Returns the statistical kurtosis of a group of values.
	SIGMA	Returns the statistical sigma of a group of values.
	SKEW	Returns the statistical skew of a group of values.
	STD_DEV	Statistical standard deviation of a group of values.
	PI	Returns the value of PI.
	SQRT	Returns the square root of a value.

Table 22: Keywords and Functions in the Custom Script Tool

Function Type	Function Names	Descriptions
String	CHR	Converts a number to a text character.
	IN	Determines if one text string is contained in another string.
	INSTR	Returns a text line that is a substring of a string.
	SETDP	Assigns the number of decimal points to be used when converting a number to a string.
	TEXTLEN	Returns the number of characters in a text string.
	UPCASE	Returns the uppercase of a text string.
	MESSAGEBOX	Shows a standard message box.
Date/Time	DATE	Returns the system's date.
	DATE$	Returns the system's date in text format.
	TIME	Returns the system's time.
	TIME$	Returns the system's time in text format.
File	OPEN	Opens a file for reading/writing.
	CLOSE	Closes a file.
	READ	Reads from a file.
	WRITE	Writes to a file.
	EOF	Test for the end of file condition.
	EXIST	Checks to see if a file exists.
	ERASE	Deletes a given file.
	CHG_PATH	Changes the default program directory.

Table 22: Keywords and Functions in the Custom Script Tool

Function Type	Function Names	Descriptions
Program Flow Control	IF THEN ELSE	Basic If – Then – Else Logic.
	WHILE WEND	Basic While-Wend Logic.
	REPEAT UNTIL	Basic Repeat-Until Logic.
	GOSUB	Calls a subroutine.
	GOTO	Performs a GoTo jump.
	END	Ends a program.
	EXIT	Exits from a program.
	DELAY	Delays a program for a given period.
	RETURN	Returns from a subroutine.
Data Logging	OPENLOGBOX	Opens a log box for logging data.
	CLOSELOGBOX	Closes a log box.
	WRITELOGBOX	Writes text into a log box.
	CLEARLOGBOX	Clears a log box.

ABS

Syntax ABS(NUM)

Description Returns the absolute value of a number.

Parameters

Name: NUM

Description: Any valid number.

Example

```
I = ABS(-3)! I = 3
R = ABS(-543.77)! R = 543.77
L = ABS(25)! L = 25
```

COS

Syntax `COS(angle)`

Description Computes and returns the cosine of the given angle.

Parameters

Name: Angle

Description: The angle, supplied in radians, for which you want the cosine.

Example

```
COS(PI) ! WILL RETURN –1
```

SIN

Syntax `SIN(angle)`

Description Calculates the sine of an angle, and returns a real number between –1 and 1.

Parameters

Name: Angle

Description: The angle, supplied in radians, for which you want the sine.

Example

```
S=SIN(PI/2)
!PLACE THE VALUE 1.0 IN VARIABLE S
```

TAN

Syntax `TAN(angle)`

Description Computes the tangent of a given angle.

Parameters

Name: Angle

Description: The angle, supplied in radians, for which you want the tangent.

Example

```
T=TAN(PI/4)
! PLACE THE VALUE 1.0 IN THE
! VARIABLE T
```

MEAN

Syntax `MEAN(“data”,count)`

Description Returns the mean value for a set of data.

Parameters

Name: Data

Description: The name of the array variable in quotes.

Name: Count

Description: The number of elements to use for calculating the arithmetic mean.

Example

```
DATA[100] = REAL
GOSUB FILL_DATA_ARRAY
M = MEAN(“DATA”,100)
MESSAGEBOX(“MEAN:”+M” “,MB_OK)
```

MEDIAN

Syntax `MEDIAN(“data”,count)`

Description Returns the median value for a set of data.

Parameters

Name: Data

Description: The name of the array variable in quotes.

Name: Count

Description: The number of data points to use.

Example

```
DATA[100] = REAL
GOSUB FILL_DATA_ARRAY
M = MEDIAN(“DATA”,100)
MESSAGEBOX(“MEDIAN:”+M” “,MB_OK)
```

KURTOSIS

Syntax `KURTOSIS("data",count)`

Description Indicates mathematically the shape of the distribution curve for a given set of data points.

Parameters

Name: Data

Description: The name of the array variable in quotes.

Name: Count

Description: The number of points in the array to use. *Count* must not exceed the length of the array.

Example

```
DATA[100] = REAL
GOSUB FILL_DATA_ARRAY
K=KURTOSIS("DATA",75)
!USE THE FIRST 75 POINTS
MESSAGEBOX("KURTOSIS:"+K,
   "CUSTOM SCRIPT",MB_OK)
```

SIGMA

Syntax `SIGMA(“data”,count)`

Description Returns the third deviation point for a set of data.

Parameters

Name: Data

Description: The name of the array variable in quotes. The array must be at least as long as the count.

Name: Count

Description: Specifies the number of data points to use.

Example

```
DATA[100] = REAL
GOSUB FILL_DATA_ARRAY
R = SIGMA(DATA,100)
```

SKEW

Syntax `SKEW("data",count)`

Description For a given set of data points, determines the mathematical skewness of those data points. The returned value is a REAL number.

Parameters

Name: Data

Description: The name of the array variable in quotes.

Name: Count

Description: Specifies the number of data points to use.

Example

```
DATA[7] = REAL
R = SKEW("DATA",7)
```

STD_DEV

Syntax `STD_DEV("data",count)`

Description Computes the standard deviation for a set of data and returns a value. The returned value is a REAL number.

Parameters

Name: Data

Description: The name of the array variable in quotes.

Name: Count

Description: Specifies the number of data points to use.

Example

```
DATA[20] = REAL
 GOSUB FILL_DATA_ARRAY
R = STD_DEV("DATA",20)
```

PI

Syntax `PI`

Description Returns the value of PI. This function is useful for converting angles expressed in degrees to radians. Radians is the required format for trigonometric functions. The returned value is 3.141592654.

Example

```
SETDP(9)
MESSAGEBOX("PI:"+PI," ",MB_OK)
!3.141592654 APPEARS ON THE SCREEN
```

SQRT

Syntax `SQRT(num)`

Description Returns the square root of a number.

Parameters

Name: Num

Description: A positive number.

Example

```
MESSAGEBOX("THE SQUARE ROOT OF
   FOUR IS", +SQRT(4)," ", MB_OK)
```

CHR

Syntax `CHR(#)`

Description Converts a number to a text character.

Parameters

Name: #

Description: A number between 0 and 255 that is converted to a ASCII character.

Example

```
MESSAGEBOX("LINE 1" + CHR(10) +
   CHR(13) + "LINE 2", "TITLE",
   MB_OK)
```

IN

Syntax `a IN b`

Description Determines if one text string is contained in another. If string *a* is contained in string *b*, a nonzero value is returned.

Parameters

Name: a

Description: A string value.

Name: b

Description: A string value.

Example

```
IF(“W” IN “Ww”) THEN
MESSAGEBOX("W IS A SUBSTRING OF
   Ww"," ",MB_OK)
```

INSTR

Syntax `INSTR (start,stop,string)`

Description Returns a text line that is a substring of a string.

Parameters

Name: Start

Description: The position to start the substring.

Name: Stop

Description: The position to stop the substring.

Name: String

Description: The string from which to extract the substring.

Notes The first position of string is 0.

Example

```
S=INSTR(3,7,"TEST LINE")
! EXTRACT THE SUBSTRING "T LIN"
! AND PUTTING IT IN S.
```

SETDP

Syntax `SETDP(# of decimal places)`

Description Assigns the number of decimal points to use when converting a number to a string.

Parameters

Name: # of decimal places

Description: For string purposes only, determines the number of decimal places to set.

Example

```
SETDP(1)
MESSAGEBOX("PI:" + PI,
   "CUSTOM SCRIPT",MB_OK)
!SHOWS 3.1
```

TEXTLN

Syntax `L=TEXTLEN(string)`

Description Returns the number of characters in a text string.

Parameters

Name: String

Description: The string to count.

Name: L

Description: The number of characters in the string.

Example

```
L=TEXTLEN(“TEXT”)
! SET THE VARIABLE L EQUAL TO 4.
```

UPCASE

Syntax `UPCASE(text)`

Description Returns the uppercase version of a text string.

Parameters

Name: Text

Description: A text string, which can contain both uppercase and lowercase characters.

Example

```
S=UPCASE(“TEST LINE.”
! PUT INTO S: “TEST LINE.”
```

MESSAGEBOX

Syntax

```
MESSAGEBOX(“Message data”,
   “box title”, windows_command)
```

Description Displays a standard Windows message box.

Parameters

Name: Messagebox

Description: Returns a constant which represents the button pushed. That is, the OK button returns IDOK, and MB_YESNO returns IDYES or IDNO.

Name: box title

Description: The title of the Windows message box.

Name: windows_command

Description: A Windows command. Typical commands are MB_OK or MB_YESNO.

Example

```
R = MESSAGEBOX(“THIS IS A TEST”,
   “SCRIPT”, MB_YESNO)
IF(R = IDNO) THEN
EXIT
```

DATE

Syntax `DATE`

Description Returns the system date in the numeric format: *yyyymmdd*, where *yyyy* represents the year, *mm* represents the month, and *dd* represents the day.

Example

```
D=DATE
! JANUARY 1, 2000 IS
! RETURNED AS D=20000101
```

DATE$

Syntax `S=DATE$`

Description Returns the system date in string format.

Example

```
S=DATE$
! RETURNS THE DATE JANUARY 1, 2000
! AS JAN 1, 2000
```

TIME

Syntax `TIME`

Description Returns the number of seconds since midnight.

Example

```
S=TIME
! PLACE THE CURRENT TIME VALUE IN
! VARIABLES.
```

TIME$

Syntax `TIME$`

Description Returns the system time in the following text format: *hh:mm:ss*, where *hh* represents the hour, *mm* represents the minute, and *ss* represents the number of seconds.

Example MESSAGEBOX("TIME:"+TIME$, " ",MB_OK)

OPEN

Syntax `OPEN(filename)`

Description Opens a text file and returns a reference number for reading and writing to a file.

Parameters

Name: Filename

Description: The name of the text file. It can be any valid text file name.

Example

```
FP = OPEN(“C:\AUTOEXEC.BAT”)
S=READ(FP)
CLOSE(FP)
```

CLOSE

Syntax `CLOSE(filevar)`

Description Closes a file with the file referenced by the variable. If this function is successful, a value of 0 is returned.

Example

```
CLOSE(FP)
! CLOSE THE FILE WHOSE FILE
IF (FP=0) THEN ! POINTER IS FP.
MESSAGEBOX("CLOSE OK"," ",MB_OK)
! THE FILE WAS CLOSED SUCCESSFULLY
```

READ

Syntax `READ(filevar)`

Description Returns one sequential line of text from a file.

Parameters

Name: filevar

Description: The name of the text file from which to read.

Example

```
FP = OPEN("MYTEXT.TXT")
S=READ(FP)
! READ ONE LINE FROM THE FILE
! REFERENCED BY FP
! PUTTING THE TEXT IN VARIABLE S.
```

WRITE

Syntax `OK=WRITE(fp, string)`

Description Writes string data to an open file.

Parameters

Name: fp

Description: A previously opened file.

Name: String

Description: A variable, expression, or literal text.

Name: OK

Description: Set to nonzero if the write is successful.

Example

```
WRITE(FP, “THIS IS A STRING”)
```

EXIST

Syntax

```
YES=EXIST(filename)
```

Description Returns a value indicating whether a specified file exists.

Parameters

Name: Filename

Description: The name of the file to check.

Name: Yes

Description: If the file exists, *Yes* is a nonzero value. If the file does not exist, *Yes* equals 0.

Example

```
YES=EXIST(“B:\TEMP.DAT”)
! YES IS NONZERO
! IF B:\TEMP.DAT EXISTS.
! YES=0 IF B:\TEMP.DAT DOES NOT
   EXIST.
```

ERASE

Syntax `ERASE(filename)`

Description Removes a named file from the disk.

Parameters

Name: filename

Description: The name of the file to delete.

Example

```
ERASE("D:\MYDIR\FOOBAR")
! ERASES FILE FOOBAR FROM D:\MYDIR
```

CHG_PATH

Syntax `CHG_PATH(path)`

Description Changes the current drive and directory to the indicated path. If this function is successful, a nonzero value is returned. If this function is unsuccessful, a value of 0 is returned.

Parameters

Name: path

Description: The drive and directory path that you want to change to.

Example

```
CHG_PATH("C:\MYDIR")
```

EOF

Syntax `EOF(file #)`

Description End of file indicator. EOF returns 0 at the end of the file and –1 when past the end of the file.

Parameters

Name: file #

Description: The file reference number that was returned when the file was opened.

Example

```
FP = OPEN("C:\AUTOEXEC.BAT")
S = READ(FP)
IF(EOF(FP)) THEN CLOSE(FP)
```

IF THEN ELSE

Syntax

```
IF(expression) THEN
  Do something
ELSE
  Do something
```

Description Conditional program branch. If the **IF** expression is not 0, execute the statements after **THEN**. Otherwise, if the **ELSE** keyword is present, execute the statements that follow the **ELSE**. If more than one statement follows either the **IF** or the **ELSE**, a paired set of braces must be used.

Example

```
IF(1>2)THEN
  MESSAGEBOX("1 IS GREATER THAN
   2", " ",MB_OK)
   GOTO EXIT
}
ELSE
  MESSAGEBOX("1 IS NOT GREATER
   THAN 2"," ",MB_OK)
```

WHILE WEND

Syntax

```
WHILE(expression)
  Do something
WEND
```

Description

While the **WHILE** expression is nonzero, executes the statements between **WHILE** and **WEND**.

Notes

This function is similar to **REPEAT UNTIL,** except that the test to evaluate the expression takes place at the top of the loop.

Example

```
DONE=0
WHILE(NOT DONE)
 do something
WEND
```

REPEAT UNTIL

Syntax

REPEAT
UNTIL(*expression*)

Description

REPEAT UNTIL waits for expression to become nonzero. Any statements between **REPEAT** and **UNTIL** are repetitively executed.

Example

```
I = 0
REPEAT
  I = I + 1
UNTIL (I >10 )
```

GOSUB

Syntax `GOSUB label RETURN`

Description **GOSUB** indicates that the program should branch to the specified label. When the **RETURN** statement is encountered, program execution resumes at the next program line after the **GOSUB**.

Parameters

Name: label

Description: A string that specifies where the program branches to.

Example

```
GOSUB PRINT_PAGE
Next line of code
:PRINT_PAGE
do something
RETURN
```

GOTO

Syntax `GOTO label`

Description **GOTO** indicates that the program should branch to the specified label. This is an unconditional branch.

Parameters

Name: label

Description: A string that specifies where the program branches to.

Example

```
GOTO DONE
...
:DONE
```

END

Syntax `END`

Description Ends the currently executing Custom Script program.

Notes The last line of a program must be **END** or **EXIT**.

Example `END`

EXIT

Syntax `EXIT`

Description Unconditionally stops the Custom Script program.

Notes The last line of a program must be **END** or **EXIT**.

Example `EXIT`

DELAY

Syntax `DELAY(time)`

Description Stops the Custom Script program from executing for a specified time.

Parameters

Name: time

Description: The time in seconds.

Example

```
DELAY(.5)
!DELAY FOR .5 SECOND
```

RETURN

Syntax `RETURN`

Description Returns program control to the statement after the **GOSUB** call.

Example

```
RETURN
```

OPENLOGBOX

Syntax

```
OPENLOGBOX(X, Y, WIDTH, HEIGHT,
   READONLY)
```

Description Opens a log window and returns a reference number to the open log window.

Parameters

Name: X

Description: The horizontal position of the upper-left corner of the window.

Name: Y

Description: The vertical position of the upper-left corner of the window.

Name: WIDTH

Description: The width of the window.

Name: HEIGHT

Description: The height of the window.

Name: READONLY

Description: If nonzero, only the Custom Script program an write to the window. If 0, the operator can enter text using the keyboard or the program.

Notes More than one window can be opened at one time.

Example

```
LOG[10] = INTEGER
! SET UP AN ARRAY TO HANDLE UP TO
! 10 DIFFERENT LOG WINDOWS
LOG[0] = OPENLOGBOX(10,200,300,
   150,YES)
!X=10; Y=200; WIDTH=300;
! HEIGHT=150; READONLY
WRITELOGBOX("THIS IS AN IMPORTANT
   LINE OF TEXT",LOG[0])
! DISPLAY SOME TEXT
PRINTLOG(LOG[0])
! PRINT TO THE PRINTER
CLEARLOGBOX(LOG[0])
! CLEAR OUT THE WINDOW
CLOSELOGBOX(LOG[0])
! CLOSE THE WINDOW
```

CLOSELOGBOX

Syntax CLOSELOGBOX(*LOGNUM*)

Description Closes an open log window.

Parameters

Name: LOGNUM

Description: The reference number returned from the **OPENLOGBOX** function.

Example

```
LOG[10] = INTEGER
! SET UP AN ARRAY TO HANDLE UP
! TO 10 DIFFERENT LOG WINDOWS
LOG[0] = OPENLOGBOX(10,200,300,
   150,YES)
```

Example (cont.)

```
!X=10; Y=200; WIDTH=300;
!=HEIGHT=150; READONLY
WRITELOGBOX("THIS IS AN IMPORTANT
   LINE OF TEXT",LOG[0])
! DISPLAY SOME TEXT
PRINTLOG(LOG[0])
! PRINT TO THE PRINTER
CLEARLOGBOX(LOG[0])
! CLEAR OUT THE WINDOW
CLOSELOGBOX(LOG[0])
! CLOSE THE WINDOW
```

WRITELOGBOX

Syntax `WRITELOGBOX("Text",`*`LOGNUM`*`)`

Description Writes text in an open log window.

Parameters

Name: LOGNUM

Description: The reference number returned from the **OPENLOGBOX** function.

Example

```
LOG[10] = INTEGER
! SET UP AN ARRAY TO HANDLE UP
! TO 10 DIFFERENT LOG WINDOWS
LOG[0] = OPENLOGBOX(10,200,300,
   150,YES)
!X=10; Y=200; WIDTH=300;
!HEIGHT=150; READONLY
WRITELOGBOX("THIS IS AN IMPORTANT
   LINE OF TEXT",LOG[0])
! DISPLAY SOME TEXT
PRINTLOG(LOG[0])
! PRINT TO THE PRINTER
```

Example (cont.)

```
CLEARLOGBOX(LOG[0])
! CLEAR OUT THE WINDOW
CLOSELOGBOX(LOG[0])
! CLOSE THE WINDOW
```

CLEARLOGBOX

Syntax CLEARLOGBOX(*LOGNUM*)

Description Clears the text from an open log window.

Parameters

Name: LOGNUM

Description: The reference number returned from the **OPENLOGBOX** function.

Example

```
LOG[10] = INTEGER
! SET UP AN ARRAY TO HANDLE UP TO
! 10 DIFFERENT LOG WINDOWS
LOG[0] = OPENLOGBOX(10,200,300,
   150,YES)
!X=10; Y=200; WIDTH=300;
!HEIGHT=150; READONLY
WRITELOGBOX("THIS IS AN IMPORTANT
LINE OF TEXT",LOG[0])
! DISPLAY SOME TEXT
PRINTLOG(LOG[0])
! PRINT TO THE PRINTER
CLEARLOGBOX(LOG[0])
! CLEAR OUT THE WINDOW
CLOSELOGBOX(LOG[0])
! CLOSE THE WINDOW
```

7

Using the Edge Finder Tool API

Overview of the Edge Finder Tool API

The API for the Edge Finder tool has one object only: the CcEdgeFinder class. The CcEdgeFinder class is designed to work within the GLI/2 environment. Its primary goal is to extract points, edges, or contours from binary images. You can then use the found points, edges, or contours to perform a multitude of measurements.

The procedure for finding edges is as follows:

1. Acquire an image of the desired object.
2. Binarize the image.
3. Supply input ROIs that either enclose the desired contour or go across the desired point or contour.
4. Specify the parameters used in the extraction process.
5. Extract the points, edges, or contours using the CcEdgeFinder class.

The class supports rectangle, line, ellipse, poly line, freehand line, poly freehand, and freehand input ROIs. Point ROIs are not supported.

Typically, you provide a line, ellipse, poly line, freehand line, poly freehand, or freehand input ROI to the CcEdgeFinder class whenever you want to produce a point or edge and not an enclosed contour. In such cases, the class produces one or more point, freehand line, or freehand output ROIs. If you want to produce a single enclosed output ROI (freehand ROI), you provide a rectangle input ROI. Enclosed output ROIs are suitable, for example, for area and perimeter measurements.

The CcEdgeFinder class uses a standard constructor and destructor and the class methods listed in Table 23.

Table 23: CcEdgeFinder Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcEdgeFinder(void);
	~CcEdgeFinder(void);
CcEdgeFinder Class Methods	BOOL SetInputRoi(CcRoiBase *InputRoi);
	BOOL SetMaskImage(CcBinaryImage *CMaskImage);
	BOOL SetObjectColor(int iObjectColor);
	BOOL SetSearchRadius(int iSearchRadius);
	BOOL SetMinObjectSize(int iMinObjectSize);
	BOOL SetMaxObjectSize(int iMaxObjectSize);
	BOOL SetMultiEdgeOption(int iOption);
	CcRoiBase** FindEdgesEx (int *iNumOfEdges);

CcEdgeFinder Methods

This section describes each method of the CcEdgeFinder class in detail.

SetInputRoi

Syntax

```
BOOL SetInputRoi(
   CcRoiBase *InputRoi );
```

Include File C_EdgeFinder.h

Description Specifies the rectangle, line, ellipse, poly line, freehand line, poly freehand, or freehand input ROI.

Parameters

Name: InputRoi

Description: Pointer to a GLI/2 ROI class. It can be either a CcRoiLine, CcRoiRect, CcRoiPolyLine, CcRoiFreeHandLine, CcRoiEllipse, CcRoiFreeHand, or CcRoiPolyFreeHand pointer, cast to the CcRoiBase pointer.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Notes In GLI/2, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in GLI/2 is defined as follows: left = x, top = y1, right = x1, bottom = y.

Notes (cont.) In contrast, the origin of the image in Windows is the upper, left corner of the image, by default. Therefore, a rectangle in Windows is defined as follows: left = x, top = y, right = x1, bottom = y1.

Point ROIs are not supported.

Example The following is a sample code fragment:

```
cRoiLine *CRoiLine=new CcRoiLine;
RECT          Line;
BOOL          bStatus;
CcEdgeFinder  CEdgeFinder;

//Line going from point 2,2 to
//10,10
Line.bottom=2;
Line.top=10;
Line.left=2;
Line.right=10;
//Set the line ROI
CRoiLine->SetRoiImageCord(
   (VOID*) &Line);

//Specify the input line ROI
bStatus=CEdgeFinder.SetInputRoi(
   (CcRoiBase *)&CRoiLine);
```


SetMaskImage

Syntax

```
BOOL SetMaskImage(
   CcBinaryImage *CMaskImage );
```

Include File C_EdgeFinder.h

Description Specifies the binary image from which edges are extracted.

Parameters

Name: CMaskImage

Description: A pointer to an image from which edges are extracted. The image must be binarized (all pixels must have a value of either 0 or 1).

Return Values

TRUE Image was valid.

FALSE Image was invalid.

Example The following is a sample code fragment:

```
CcBinaryImage *CMaskImage;
CcEdgeFinder   CEdgeFinder;
BOOL           Status;
//Fill the above image however you
//wish
. . . .
//Pass it to the Edge Finder class
Status = CEdgeFinder.SetMaskImage(
  CMaskImage);
```

SetObjectColor

Syntax

```
BOOL SetObjectColor(
   int iObjectColor);
```

Include File C_EdgeFinder.h

Description Specifies the color of the object (white or black) that contains the edge you are searching for. The edge is placed within the object.

Parameters

Name: iObjectColor

Description: Object color. The value can be 0 (white) or 1 (black).

Return Values

TRUE Successful.

FALSE Unsuccessful.

Example The following is a sample code fragment:

```
int             iColor;
CcEdgeFinder    CEdgeFinder;
BOOL            Status;

// Set color to black
iColor=1;
Status=CEdgeFinder.SetObjectColor(
     iColor);
```

SetSearchRadius

Syntax

```
BOOL SetSearchRadius(
   int iSearchRadius);
```

Include File C_EdgeFinder.h

Description For line, ellipse, poly line, freehand line, poly freehand, or freehand input ROIs, specifies the number of pixels to include in an edge.

Description (cont.) For example, assume that a line input ROI is placed across the desired edge. The pixel that belongs to the edge and is exactly under the line input ROI is collected first. Then, the number of pixels specified by *iSearchRadius* is collected first to one side of the line ROI and then to the other side of the line ROI. The total number of pixels contained in the generated edge is (2 x *iSearchRadius*) + 1.

Parameters

Name: iSearchRadius

Description: The number of pixels to include in a point or edge. For example, if *iSearchRadius* equals 0, point ROIs are generated. If *iSearchRadius* is between 1 and the total number of pixels in the edge, freehand line ROIs are generated. If *iSearchRadius* is greater than the total number of pixels in the edge, freehand ROIs are generated.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Notes Point and rectangle ROIs are not supported.

Example The following is a sample code fragment:

```
int             iSearchRadius;
CcEdgeFinder    CEdgeFinder;
BOOL            Status;
// Set the radius
iSearchRadius=10;
```

Example (cont.)

```
// Specify that 21 pixels should
// be found in the edge
Status=CEdgeFinder.SetSearchRadius
   (iSearchRadius);
```

SetMinObjectSize

Syntax

```
BOOL SetMinObjectSize(
   int iMinObjectSize);
```

Include File

C_EdgeFinder.h

Description

For rectangle input ROIs only, specifies the total number of pixels in an object, below which the object is rejected by the algorithm. This value, combined with *iMaxObjectSize*, allows you to focus on a particular object if you are forced to enclose more than a single object in the rectangle ROI.

Parameters

Name: iMinObjectSize

Description: The minimum object size, specified as the total number of pixels. The value must be greater than or equal to 4.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
int            iMinObjectSize;
CcEdgeFinder   CEdgeFinder;
BOOL           Status;
// Set minimum object size
iMinObjectSize=10;
// Specify the number of pixels
//below which the object is
//rejected
Status=CEdgeFinder.
   SetMinObjectSize(
   iMinObjectSize);
```

SetMaxObjectSize

Syntax

```
BOOL SetMaxObjectSize(
   int iMaxObjectSize);
```

Include File C_EdgeFinder.h

Description For rectangle input ROIs only, specifies the total number of pixels in an object, above which the object is rejected by the algorithm. This value, combined with *iMinObjectSize*, allows you to focus on a particular object if you are forced to enclose more than a single object in the rectangle ROI.

Parameters

Name: iMaxObjectSize

Description: The maximum object size, specified as the total number of pixels. The value must be greater than or equal to 4.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
int            iMaxObjectSize;
CcEdgeFinder   CEdgeFinder;
BOOL           Status;
// Set minimum object size
iMaxObjectSize=10;
// Specify the number of pixels
// above which the object is
// rejected
Status=CEdgeFinder.
   SetMaxObjectSize(
   iMaxObjectSize);
```

SetMultiEdgeOption

Syntax

```
BOOL SetMultiEdgeOption(
   int iOption);
```

Include File C_EdgeFinder.h

Description For line, ellipse, poly line, freehand line, poly freehand, or freehand input ROIs, specifies the edges to find.

Parameters

Name: iOption

Description: Specifies one of the following values:

- 0 – FIRST_EDGE – Only the left-most edge is found.

Description (cont.):

- 1 – LAST_EDGE – Only the right-most edge is found.
- 2 – FALLING_EDGE – All falling edges (black to white transitions) are found.
- 3 – RISING_EDGE – All rising edges (white to black transitions) are found.
- 4 – ALL_EDGE – All edges are found.

Return Values

TRUE *iOption* is greater than or equal to 0 and less than or equal to 4.

FALSE *iOption* is less than 0 or greater than 4.

Notes Currently this method does not support rectangle or point input ROIs.

Example The following is a sample code fragment:

```
CcEdgeFinder   CEdgeFinder;
int iOption;
iOption = ALL_EDGE;
BOOL           Status;

//Find all edges.
Status = CEdgeFinder.
   SetMultiEdgeOption(iOption);
```

FindEdgesEx

Syntax

```
CcRoiBase** FindEdgesEx(
   int *iNumOfEdges);
```

Include File C_EdgeFinder.h

Description

Generates one or more ROIs representing the found edges, points, or contours. For line, ellipse, poly line, poly freehand, freehand, or freehand line ROIs, point, freehand, or freehand line ROIs are generated.

For a rectangle ROI, a single freehand ROI is generated.

Parameters

Name: iNumOfEdges

Description: A pointer to an integer that records the number of found edges.

Return Values

A list of pointers to the ROIs that describe the edges.	Edges were detected.
NULL	No edge was detected.

Notes

You cannot generate multiple ROIs from a rectangle ROI.

Example

The following is a sample code fragment:

```
CcRoiBase      *CNewRois;
CcRoiBase      *CInputRoi;
CcBinaryImage  *CImageMask;
int             iSearchRadius;
BOOL            bSearchDirection;
int            iColor,
                iMinObjectSize,
                iMaxObjectSize;
CcEdgeFinder    CEdgeFinder;

//Set the above variables
//appropriately!
. . . .
```

Example (cont.)

```
// Set class input parameters
CEdgeFinder.SetInputRoi(
   CInputRoi);
CEdgeFinder.SetMaskImage((
     CcBinaryImage *)CImageMask);
CEdgeFinder.SetSearchRadius(
     iSearchRadius);
CEdgeFinder.SetObjectColor(
   iColor);
CEdgeFinder.SetMinObjectSize(
     iMinObjectSize);
CEdgeFinder.SetMaxObjectSize(
     iMaxObjectSize);
CEdgeFinder.SetMultiEdgeOption(
   iMultiEdgeOption);
// Generate the new ROIs
CNewRoi = CEdgeFinder.FindEdgesEx(
   &iNumOfEdges);


//Verify the output
if (CNewRois ==NULL)
{
    Error("Failed to generate a
   new ROI.");
}
for (int I=0;I<iNumOfEdges;I++)
{
   if (CnewROIs[I]!=NULL)
   {
   . . . .
   }
}
```

8

Using the File Manager Tool API

Overview of the File Manager Tool API

The API for the File Manager tool has one object only: the CcFileConv class. This tool opens multiple file formats so that images can be used with GLI/2, and saves an GLI/2 CcImage image class in a standard BMP or TIFF file format.

For further information on CcImage objects, refer to the example program at the end of this chapter.

The CcFileConv class uses a standard constructor and destructor and the class methods listed in Table 24.

Table 24: CcFileConv Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcFileConv( );
	~ CcFileConv( );
CcFileConv Class Methods	CcImage* LoadImage(char* cFileName,int iGrayScaleFlag);
	int SaveImage(CcImage* CImage,char* cFileName, short nFlag);
	int SetSizeOptions(int iWidthFlag);

CcFileConv Methods

This section describes each method of the CcFileConv class in detail.

LoadImage

Syntax

```
CcImage* LoadImage(
  char* cFileName,
  int iGrayScaleFlag);
```

Include File C_Fconv.h

Description Loads an image file from disk.

Parameters

Name: cFileName

Description: Full path name of the file to open.

Name: iGrayScaleFlag

Description: If the image in the file is a grayscale image, open the image as one of the following:

- LOAD_AS_8BIT – Creates an 8-bit grayscale image and opens the image into it.
- LOAD_AS_16BIT – Creates a 16-bit grayscale image and opens the image into it.
- LOAD_AS_32BIT – Creates a 32-bit grayscale image and opens the image into it.
- LOAD_AS_FLOAT – Creates a floating-point grayscale image and opens the image into it.

Description (cont.):

- LOAD_AS_RGB – Creates a 24-bit RGB color image and opens the image into it.
- LOAD_AS_HSL – Creates a 24-bit HSL image and opens the image into it.

Notes

The **LoadImage( )** method creates the image, opens the image file, and returns an GLI/2 CcImage image pointer. If the image you are opening is a 24-bit color image, the *iGrayScaleFlag* is ignored.

It is your responsibility to free the memory for the returned image (or make sure something else frees the memory for the image). If you need to free the memory for the image, use the delete operator.

If you are creating a custom tool to be used in conjunction with the GLI/2 main application, you can add the image to the main application's image list. In this situation, the main application frees the memory for you when the application terminates. For information on creating custom tools, see Chapter 21 starting on page 661. For information on the main application, see Chapter 2 starting on page 11.

Return Values

NULL	Unsuccessful.
A valid image pointer.	Successful.

SaveImage

Syntax

```
int SaveImage(
  CcImage* CImage,
  char* cFileName,
  short nFlag);
```

Include File C_Fconv.h

Description Saves an GLI/2 Image object to disk.

Parameters

Name: CImage

Description: Pointer to the image to save.

Name: cFileName

Description: Full path name of the file to save.

Name: nFlag

Description: Flag that determines the file format and compression to use when saving the image; the value for *nFlag* can be one of the following:

- FILETYPE_BMP – Saves the image as a standard Windows bitmap file (BMP).
- FILETYPE_TIFF_NO_COMPRESSION – Saves the image as a standard TIFF file with no compression.
- FILETYPE_TIFF_DEFAULT – Saves the image as a standard TIFF file with compression set to automatic.
- FILETYPE_TIFF_PACKBITS – Saves the image as a standard TIFF file with compression set to “run-length encode”.

Notes The CImage parameter is a pointer to an GLI/2 CcImage object. This can be a pointer to any derived image type: 8-bit grayscale, 16-bit grayscale, 32-bit grayscale, floating-point grayscale, or 24-bit color. The *nFlag* parameter is used with all image types. If the image type is 32-bit grayscale or floating-point grayscale, use only the FILETYPE_BMP flag. If you try to save a 32-bit or floating-point grayscale image with any of the TIFF options, **SaveImage( )** fails. **SaveImage( )** does not free the Image object.

Return Values

–1	Unsuccessful.
0	Successful.

SetSizeOptions

Syntax

```
int SetSizeOptions(
   int iWidthFlag);
```

Include File C_Fconv.h

Description Sets how a file is opened if its width is not divisible by four. It has no effect on images with a width divisible by four.

Parameters

Name: iWidthFlag

Description: Flag determines how to open an image whose width is not divisible by four; its value can be one of the following:

- IMAGE_WIDTH_TRIM – Trims the width of the image so that the width is divisible by four. For example, if your image is 457 pixels wide, the new width of the image is 456. The extra pixels are discarded. The height of the image is not effected.
- *IMAGE_WIDTH_EXACT – The default value of *iWidthFlag*. If the image is not divisible by four, the image fails to open. This option allows only images divisible by four to be opened.
- IMAGE_WIDTH_ADD – Adds to the width of the image so that the width is divisible by four. For example, if your image is 457 pixels wide, the new width of the image is 460. The extra pixels added to the width have a value of 0. The height of the image is not effected.

Notes GLI/2 images must be divisible by four. This is due to the way GLI/2 accesses images in memory.

Return Values

NULL Unsuccessful.

A valid image pointer. Successful.

Example Program Using the File Manager Tool API

This example opens three different images in three different file formats (PCX, TIFF, and BMP), and saves all three images as compressed TIFF files. The images are opened as 8-bit grayscale images if they are grayscale images or as 24-bit color images if they are color images. If they are color images, the LOAD_AS_8BIT flag is ignored. After they are opened (or loaded), you can use the images as normal GLI/2 Image objects.

Note: This example is made from code fragments with error checking removed. In an actual program, you should check return values and pointers.

```
int OpenSaveImages(void)
{
CcImage* CImage1;
CcImage* CImage2;
CcImage* CImage3;
CcFileConv CFileConv;

//Open the three images
CImage1 = CFileConv.LoadImage(“C:\\Image1.pcx”,
   LOAD_AS_8BIT);
CImage2 = CFileConv.LoadImage(“C:\\Image2.tif”,
   LOAD_AS_8BIT);
CImage3 = CFileConv.LoadImage(“C:\\Image3.bmp”,
   LOAD_AS_8BIT);

//Note: You could now do something with the
images...
```

```
//Save all images as compressed TIFF files
CFileConv.SaveImage(CImage1,"C:\\Image1.tif",
   FILETYPE_TIFF_PACKBITS);
CFileConv.SaveImage(CImage2,"C:\\Image2.tif",
   FILETYPE_TIFF_PACKBITS);
CFileConv.SaveImage(CImage3,"C:\\Image3.tif",
   FILETYPE_TIFF_PACKBITS);

//We must now free the memory for the images
because the //LoadImage( ) method allocated memory
delete CImage1;
delete CImage2;
delete CImage3;
}
```

9

Using the Filter Tool API

Overview of the FilterTool API

The API for the Filter tool has one object only: the CcConvolution class. This tool performs a convolution on a input image (derived from class CcImage), and places the result in an output image. This operation is performed with respect to the given ROI (derived from class CcRoiBase).

The CcConvolution class uses a standard constructor and destructor and the class methods listed in Table 25.

Table 25: CcConvolution Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcConvolution( );
	~CcConvolution( );
CcConvolution Class Methods	int SetKernel(STKERNEL* stKer1,STKERNEL* stKer2);
	int GetKernel(STKERNEL* stKer1,STKERNEL* stKer2);
	int DoConvolution(CcImage* CImageIn, CcImage* CImageOut, CcRoiBase* CRoi,float fGain, float fOffset,float fDivide, float fLowThreshold, float fHiThreshold, int iThresholdFlag);
	int DoConvolutionRGB(Cc24BitRGBImage* CImageIn, Cc24BitRGBImage* CImageOut,CcRoiBase* CRoi, float fGain,float fOffset,float fDivide,float fLowThreshold, float fHiThreshold,int iThresholdFlag);
	int DoConvolutionHSL(Cc24BitHSLImage* CImageIn, Cc24BitHSLImage* CImageOut,CcRoiBase* CRoi, float fGain,float fOffset,float fDivide,float fLowThreshold, float fHiThreshold,int iThresholdFlag);
	int RestoreKernel(char* cFileName);
	int SaveKernel(char* cFileName);

CcConvolution Methods

This section describes each method of the CcConvolution class in detail.

SetKernel

Syntax

```
int SetKernel(
  STKERNEL* stKer1,
  STKERNEL* stKer2);
```

Include File C_Convlu.h

Description Sets kernel 1 and kernel 2 for the performed convolution.

Parameters

Name: stKer1

Description: Pointer to structure of type STKERNEL.This parameter holds information for kernel 1.

Name: stKer2

Description: Pointer to a structure of type STKERNEL. This parameter holds information for kernel 2.

Notes This method sets the kernels that are used by the class when the method **DoConvolution( )** is called.

The kernels are of type STKERNEL and are defined as follows:

```
struct KernelTag {
int iWidth;
int iHeight;
int iXCenterOffset;
int iYCenterOffset;
int iKernel[7][7];
```

Notes (cont.)

```
int iKernelFlag;
};
typedef KernelTag STKERNEL;
```

The entries for this structure are as follows:

- **iWidth** – The width of the kernel in pixels.
- **iHeight** – The height of the kernel in pixels.
- **iXCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the x-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **iYCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the y-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **Kernel[7][7]** – A 7 x 7 array of values to hold the coefficients of the kernel. Depending on the width and height of the kernel, not all of these values can be used.
- **iKernelFlag** – A flag to determine whether to use only kernel 1 in the convolution or to use both kernel 1 and kernel 2 in the convolution. Make sure this flag is the same for both kernels. This flag can take one of the following values:
  - CONVLU_SINGLE_KERNEL – Use kernel 1 only.
  - CONVLU_TWO_KERNEL – Use both kernels.

Return Values

–1	Unsuccessful.
0	Successful.

GetKernel

Syntax

```
int GetKernel(
  STKERNEL* stKer1,
  STKERNEL* stKer2);
```

Include File C_Convlu.h

Description Returns the settings of kernel 1 and kernel 2 that are used in the performed convolution.

Parameters

Name: stKer1

Description: Pointer to a structure of type STKERNEL. This parameter holds information for kernel 1.

Name: stKer2

Description: Pointer to a structure of type STKERNEL. This parameter holds information for kernel 2.

Notes This method returns the kernels that are used by the class when the method **DoConvolution( )** is called. The kernels are of type STKERNEL and are defined as follows:

```
struct KernelTag {
int iWidth;
int iHeight;
int iXCenterOffset;
int iYCenterOffset;
int Kernel[7][7];
```

Notes (cont.)

```
int iKernelFlag;
};
typedef KernelTag STKERNEL;
```

The entries for this structure are as follows:

- **iWidth** – The width of the kernel in pixels.
- **iHeight** – The height of the kernel in pixels.
- **iXCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the x-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **iYCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the y-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **Kernel[7][7]** – A 7 x 7 array of values to hold the coefficients of the kernel. Depending on the width and height of the kernel, not all of these values can be used.
- **iKernelFlag** – A flag to determine whether to use only kernel 1 in the convolution or to use both kernel 1 and kernel 2 in the convolution. Make sure this flag is the same for both kernels. This flag can take one of the following values:
  - CONVLU_SINGLE_KERNEL – Use kernel 1 only.
  - CONVLU_TWO_KERNEL – Use both kernels.

Return Values

–1	Unsuccessful.
0	Successful.

DoConvolution/DoConvolutionRGB/DoConvolutionHSL

Syntax

```
int DoConvolution(
  CcImage* CImageIn,
  CcImage* CImageOut,
  CcRoiBase* CRoi,
  float fGain,
  float fOffset,
  float fDivide,
  float fLowThreshold,
  float fHiThreshold,
  int iThresholdFlag);
```

Include File C_Convlu.h

Description Performs the convolution for the given image with respect to the given ROI.

Parameters

Name: CImageIn

Description: Image derived from the CcImage class and used as the input image.

Name: CImageOut

Description: Image derived from the CcImage class and used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: fLowThreshold

Description: Low threshold limit; this parameter is not used unless it is specified by *iThresholdFlag*.

Name: fGain

Description: Gain that is applied to the resulting data.

Name: fOffset

Description: Offset that is applied to the resulting data.

Name: fDivide

Description: Division that is applied to the resulting data.

Name: fLowThreshold

Description: Low threshold limit; this parameter is not used unless it is specified by *iThresholdFlag*.

Name: fHiThreshold

Description: High threshold limit; this parameter is not used unless it is specified by *iThresholdFlag*.

Name: iThresholdFlag

Description: Flag that determines whether thresholding is performed. A value of 1 indicates that thresholding is performed. A value of 0 indicates that thresholding is not performed.

Notes This method performs a convolution on the given input image with respect to the given ROI. It places the output in the given output image. After calculating the convolution, *fGain* and *fOffset* are always applied to the output value. The output value is then thresholded to the *fLowThreshold* and *fHiThreshold* limits, providing that the *iThresholdFlag* is set to 1.

Notes (cont.)

Within the CcConvolution class are private methods that are called by this method, provided that certain conditions are met. These private methods are called for speed of execution.

The conditions that produce faster execution of a convolution are the following:

- Input image is one of the following: 8-bit grayscale, 32-bit grayscale, or floating-point grayscale.
- ROI is rectangular or elliptical.
- 3. Kernel is a 3 x 3 centered kernel.

(You can have a dual-kernel convolution and still meet these criteria; both kernels must be 3 x 3 and centered.)

You do not have to do anything special to invoke the faster methods; the class does it automatically.

Returned Values

–1	Unsuccessful.
0	Successful.

RestoreKernel

Syntax

```
int RestoreKernel(
   char* cFileName);
```

Include File C_Convlu.h

Description Restores the kernels that were saved on disk.

Parameters

Name: cFileName

Description: Full path name of a file that contains the kernels you wish to restore.

Notes This method opens a set of kernels (kernel 1 and kernel 2) that were stored in the file *cFileName*. It restores all the information for kernel 1 and kernel 2 that is defined in the structure STKERNEL, not just the coefficients of the kernels.

Returned Values

–1 Unsuccessful.

0 Successful.

SaveKernel

Syntax `int SaveKernel(char* cFileName);`

Include File C_Convlu.h

Description Saves the kernels to disk.

Parameters

Name: cFileName

Description: Full path name of a file that is created to hold the kernel information.

Notes This method saves the set of kernels (kernel 1 and kernel 2) used by the class CcConvolution to disk. It saves all the information given in the structure STKERNEL, not just the kernel coefficients. You can later retrieve this information using **RestoreKernel( )**.

Returned Values

–1	Unsuccessful.
0	Successful.

Example Program Using the Filter Tool API

This example program performs a Sobel filter operation on an 8-bit input image with respect to a given rectangular ROI. It then places the output into a newly-created, blank 32-bit image and saves this image to disk.

Note: This example is made from code fragments from the Filter tool with error checking removed. In an actual program, you should check return values and pointers.

```
int SomeFunction(void)
{
CcConvolution* CFilter;
//Object to perform convolution
CcGrayImage256* CImageIn;
//8-bit grayscale input image
CcGrayImageInt32* CImageOut;
//32-bit grayscale output image
CcRoiRect* CRoi;
//Rectangular ROI
int iHeight,iWidth;
RECT stROI;

//Create objects
CFilter = new CcConvolution( );
CImageIn = new CcGrayImage256( );
CImageOut = new CcGrayImageInt32( );
CRoi = new CcRoiRect( );

//Open input image from disk
CImageIn->OpenBMPFile(“image1.bmp”);
```

```
//Create blank image of same size as input image
   for output image
CImageIn->GetHeightWidth(&iHeight,&iWidth);
CImageOut->MakeBlankBMP(iHeight,iWidth,0,”Output”);

//Create rectangular ROI
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRoi->SetRoiImageCord((VOID*)&stROI);

//Open Sobel kernel to perform Sobel filter
CFilter->RestoreKernel(“Sobel.ker”);

//Run the filter (gain of 1, offset of 0,
//no thresholding)
CFilter->DoConvolution(CImageIn,CImageOut,CRoi,1,0,
   1,0,0,0);

//Save output image to disk
CImageOut->SaveBMPFile(“Output.bmp”);

//Free memory
delete CFilter;
delete CImageIn;
delete CImageOut;
delete Croi;

return(0);

}
```

10

Using the Histogram Tool API

Overview of the Histogram Tool API

The API for the Histogram tool has one object only: the CcHistogram class. This tool creates a histogram from an input image (derived from class CcImage) with respect to a given ROI (derived from class CcRoiBase). The CcHistogram class is derived from the CcCurve GLI/2 class. You can use the methods of the CcCurve class to access the histogram data. For further information on these objects, refer to the example program at the end of this chapter.

The CcHistogram class uses a standard constructor and destructor and the class methods listed in Table 26.

Table 26: CcHistogram Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcHistogram( );
	~CcHistogram( );
CcHistogram Class Methods	int MakeHistogram(CcImage* CImage,CcRoiBase* CRoi);
	int Normalize(void);
	STHISTSTATS* GetStats(float fStart=-1,float fStop=-1);

CcHistogram Methods

This section describes each method of the CcHistogram class in detail.

MakeHistogram

Syntax

```
int MakeHistogram(
  CcImage* CImage,
  CcRoiBase* CRoi);
```

Include File C_Hist.h

Description Creates a histogram of the image with respect to the given ROI.

Parameters

Name: CImage

Description: Image derived from the CcImage class and used as the input image.

Name: CRoi

Description: ROI area in which to perform the operation.

Notes This method uses images derived from the GLI/2-supplied CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. This method uses an ROI derived from the GLI/2-supplied CcRoiBase class. These include the rectangle, line, elliptical, and freehand ROIs. It also works with your own images or ROIs derived from these classes.

Notes (cont.) The CcHistogram class is derived from the CcCurve class. After making a histogram, you can add it to the list of curves of a graph class and then easily display the graph containing the histogram in any window.

To access the histogram data, call the methods of the CcCurve class. For more information, refer to the example program at the end of this chapter.

Return Values

–1 Unsuccessful.

0 Successful.

Normalize

Syntax `int Normalize(void);`

Include File C_Hist.h

Description Normalizes the histogram created using the method **MakeHistogram( )**.

Notes This method normalizes the histogram owned by this class. To normalize a histogram, each point in the histogram is divided by the total number of pixels that comprise the histogram. This value is then multiplied by 100. The total number of pixels in the histogram is the number of pixels enclosed by the ROI with which the histogram was created.

Return Values

–1 Unsuccessful.

0 Successful.

GetStats

Syntax

```
STHISTSTATS* GetStats(
  float fStart = -1,
  float fStop = -1);
```

Include File C_Hist.h

Description Returns the statistics for the histogram.

Parameters

Name: fStart

Description: The starting position to use when calculating the statistics.

Name: fStop

Description: The ending position to use when calculating the statistics.

Notes You must first create the histogram by calling **MakeHistogram( )** before calling this method. If you want the statistics for the entire histogram, you can call this method with no parameters. The *fStart* and *fStop* parameters correspond to the stat bars, described earlier in the chapter.

The returned histogram statistics structure is defined as follows:

```
struct stHistStatsTag {
float fMin;
//Lowest value in histogram with
//nonzero value
float fMax;
//Highest value in histogram with
//nonzero value
```

Notes (cont.)

```
float fMean;
//Average value in histogram
float fStdDev;
//Standard Deviation of histogram
float fTotalPixels;
//Total number of pixels in
//histogram
float fSelPixels;
//Selected number of pixels in
//histogram
float fPercentSel;
//Percent of pixels in histogram
//that are selected
};
typedef struct stHistStatsTag
   STHISTSTATS;
```

Return Values

NULL	Unsuccessful.
A pointer to a histogram statistics structure.	Successful.

Example Program Using the Histogram Tool API

This example program compares the same region in two 8-bit images, looking for which area is brighter above a threshold value of 50.

Note: This example is made from code fragments with error checking removed. In an actual program, you should check return values and pointers.

```
int SomeFunction(void)
{
CcHistogram* cHist;
//Object to perform histogram
CcGrayImage256* CImageIn1;
//8-bit grayscale input image1
CcGrayImage256* CImageIn2;
//8-bit grayscale input image2
CcRoiRect* Roi;
//Rectangular ROI
RECT stROI;

STPOINTS* stPoints;
//Pointer to histogram data
int x;
//Temp variable
float fBrightValue1;
//Brightness values
float fBrightValue2;

//Create objects
CHist = new CcHistogram( );
CImageIn1 = new CcGrayImage256( );
CImageIn2 = new CcGrayImage256( );
```

```
CRoi = new CcRoiRect( );
//Open input images from disk
CImageIn1->OpenBMPFile(“image1.bmp”);
CImageIn2->OpenBMPFile(“image2.bmp”);

//Create rectangular ROI
stROI.bottom = 50; stROI.top = 150;
stROI.left = 50; stROI.right = 150;
CRoi->SetRoiImageCord((VOID*)&stROI);

//Create histogram of input image 1
CHist->MakeHistogram(CImageIn1,CRoi);

//Calculate brightness value for image 1
//Get pointer to histogram data
stPoints = CHist->GetCurveData( );
//Calculate value
fBrightValue1 = 0;
for(x=0; x<CHist->GetNumberOfPoints( ); x++)
{
if(stPoints[x].fX > 50)
fBrightValue1 += stPoints[x].fY;
}

//Create histogram of input image 2
CHist->MakeHistogram(CImageIn2,CRoi);

//Calculate brightness value for image 2
//Get pointer to histogram data
stPoints = CHist->GetCurveData( );
//Calculate value
fBrightValue2 = 0;
for(x=0; x<CHist->GetNumberOfPoints( ); x++)
{
if(stPoints[x].fX > 50)
fBrightValue2 += stPoints[x].fY;
}
```

```
//Free memory
delete CHist;
delete CImageIn1;
delete CImageIn2;
delete Croi;

//Tell user which is brighter
if(fBrightValue2 > fBrightValue1)
 MessageBox(0,”Image 2”,”Answer”,MB_OK);
else if(fBrightValue2 < fBrightValue1)
 MessageBox(0,”Image 1”,”Answer”,MB_OK);
else
 MessageBox(0,”Equal”,”Answer”,MB_OK);

return(0);
}
```

11

Using the Image Modifier Tool API

Overview of the Image Modifier Tool API

The API for the Image Modifier tool has one object only: the CcImgMod class. This tool provides crop, flip / rotate, and scale operations to manipulate images.

The CcImgMod class uses a standard constructor and destructor and the class methods listed in Table 27.

Table 27: CcImgMod Object Methods

Method Type	Method Name
Constructor & Destructor Method	CcImgMod();
	~CcImgMod();
CcImgMod Class Methods	int Crop(CcImage* CImageIn, CcImage* cImageOut, CcRoiBase* CRoi, int iFillValue, bool bKeepOrigSize);
	int FlipRotate(CcImage* CImageIn, CcImage* CImageOut, int iOperation, int iRotateAmount);
	int Scale(CcImage* CImageIn, CcImage* CImageOut, int iScaleFactor, int iFillValue);

CcImgMod Methods

This section describes each method of the CcImgMod class in detail.

Crop

Syntax

```
int Crop(CcImage* CImageIn,
   CcImage* CImageOut,
   CcRoiBase* CRoi,
   int iFillValue,
   Bool bKeepOrigSize);
```

Include File C_ImgMod.h

Description Crops an image.

Parameters

Name: CImageIn

Description: Image that is derived from class CcImage and used as the input image.

Name: CImageOut

Description: Image that is derived from class CcImage and used as the output image.

Name: CRoi

Description: The ROI in which to perform the crop operation. This can be a rectangle, ellipse, poly freehand, or freehand ROI.

Name: iFillValue

Description: Specifies the background color of the image. Value range from 0 (black) to 255 (white).

Name: bKeepOrigSize

Description: If TRUE, the output image is the same size as the input image. The area inside the ROI is cropped and the rest of the image is the color specified by *iFillValue*.

If FALSE, the output image is the size of the smallest rectangle that surrounds the entire ROI. Any area in the output image that is not inside the ROI is set to the color specified by *iFillValue*.

Notes Rectangle and ellipse ROIs are saved and/or recreated automatically in a script; however, poly freehand and freehand ROIs are not saved or recreated automatically in a script.

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
Void SomeFunction(void)
{
//Start of Dec Section
//8-bit grayscale images
CcGrayImage256* C8BitImageIn;
CcGrayImage256* C8BitImageOut;

//Where operation takes place
CcRoiRect* CRectRoi;
int iFillValue;
BOOL bKeepOrigSize;
//End of Dec Section
```

Example (cont.)

```
//Allocate memory for objects
C8BitImageIn = new
   CcGrayImage256();
C8BitImageOut = new
   CcGrayImage256();
CRectRoi = new CcRoiRect();

//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectROI->SetRoiImageCord((VOID*)
   &stROI);

//Open image from disk (or get
//image data from frame grabber
C8BitImageIn->OpenBMPFile(
   “InImage.bmp”);

//Set fill value to black
iFillValue = 0;

//Do not keep the original size
//Make output image the same size
//as CRectRoi)
bKeepOrigSize = FALSE;

//Crop the image
CImgMod.Crop(C8BitImageIn,
   C8BitImageOut, CRectRoi,
   iFillValue, bKeepOrigSize);
//Save output to disk
C8BitImageOut->SaveBMPFile(
   “OutImage.bmp”);
```

Example (cont.)

```
//Free memory
delete C8BitImageIn;
delete C8BitImageOut;
delete CRectRoi;
}
```

FlipRotate

Syntax

```
int FlipRotate(CcImage* CImageIn,
   CcImage* CImageOut,
   int iOperation,
   int iRotateAmount);
```

Include File C_ImgMod.h

Description Flips an image horizontally or vertically, or rotates an image by 90, 180, or 270 degrees.

Parameters

Name: CImageIn

Description: Image that is derived from class CcImage and used as the input image.

Name: CImageOut

Description: Image that is derived from class CcImage and used as the output image.

Name: iOperation

Description: Specifies one of the following operations:

- FLIPROTATE_FLIP_HORZ – Flips the image horizontally.
- FLIPROTATE_FLIP_VERT – Flips the image vertically.
- FLIPROTATE_ROTATE – Rotates the image by *iRotateAmount*.

Name: iRotateAmount

Description: Specifies the amount of rotation in degrees to apply to the input image. The value can be 90, 180, or 270.

Notes None

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
Void SomeFunction(void)
{
//Start of Dec Section
//8-bit grayscale images
CcGrayImage256* C8BitImageIn;
CcGrayImage256* C8BitImageOut;
int iOperation;
int iRotateAmount;
//End of Dec Section

//Allocate memory for objects
C8BitImageIn = new
   CcGrayImage256();
C8BitImageOut = new
   CcGrayImage256();


//Open image from disk (or get
//image data from frame grabber
C8BitImageIn->OpenBMPFIle(
   “InImage.bmp”);
//Rotate the image by 90 degrees
iOperation = FLIPROTATE_ROTATE;
iRotateAmount = 90;
```

Example (cont.)

```
CImgMod.FlipRotate(C8BitImageIn,
   C8BitImageOut,iOperation,
   iRotateAmount);

//Save output to disk
C8BitImageOut->SaveBMPFile(
   "OutImage.bmp");

//Free memory
delete C8BitImageIn;
delete C8BitImageOut;
}
```

Scale

Syntax

```
int Scale(CcImage* CImageIn,
   CcImage* CImageOut,
   int iScaleFactor,
   int iFillValue);
```

Include File C_ImgMod.h

Description Scales an image by 25, 50, 100, 200, or 400 percent.

Parameters

Name: CImageIn

Description: Image that is derived from class CcImage and used as the input image.

Name: CImageOut

Description: Image that is derived from class CcImage and used as the output image.

Name: iScaleFactor

Description: Specifies the scale factor (in percent) that is applied to the input image. Values can be 25, 50, 100, 200, or 400.

Name: iFillValue

Description: Specifies the background color of the image. This parameter is required only for images that are reduced in size (scale factor is 25 or 50). Values range from 0 (black) to 255 (white).

Notes None

Return Values

–1 Unsuccessful.

0 Successful.

Example The following is a sample code fragment:

```
Void SomeFunction(void)
{
//Start of Dec Section
//8-bit grayscale images
CcGrayImage256* C8BitImageIn;
CcGrayImage256* C8BitImageOut;
int iScaleFactor;
int iFillValue;
//End of Dec Section

//Allocate memory for objects
C8BitImageIn = new
   CcGrayImage256();
C8BitImageOut = new
   CcGrayImage256();
```

Example (cont.)

```
//Open image from disk (or get
//image data from frame grabber
C8BitImageIn->OpenBMPFIle(
   “InImage.bmp”);

//Scale the image by 50%
iScaleFactor = 50;
```

12

Using the Line Profile Tool API

Overview of the Line Profile Tool API

The API for the Line Profile tool has one object only: the CcLineProfile class. This tool creates a line profile for an input image (derived from class CcImage) with respect to a given line ROI (derived from class CcRoiBase). The CcLineProfile class is derived from the CcCurve GLI/2 class. You can use the methods of the CcCurve class to access the line profile data. For further information on these objects, refer to the example program at the end of this chapter.

The CcLineProfile class uses a standard constructor and destructor and the class methods listed in Table 28.

Table 28: CcLineProfile Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcLineProfile( );
	~CcLineProfile( );
CcLineProfile Class Methods	int MakeProfile(CcImage* CImage,CcRoiLine* CRoi, int iAverage);
	int AverageProfile(int iAverage);
	int TakeDerivative(int iDelta);
	int GainAndOffset(float fGain,float fOffset);
	PIXELGROUPING* GetPixelLocationsAll(void);
	PIXELGROUPING* GetPixelLocationsCenter(void);
	float GetLineDistance(float fPixelLocationStart, float fPixelLocationEnd,CcCalibration* CalibrationObject);
	float GetStraightDistance(float fPixelLocationStart, float fPixelLocationEnd,CcCalibration* CalibrationObject);

Table 28: CcLineProfile Object Methods (cont.)

Method Type	Method Name
CcLineProfile Class Methods (cont.)	int GetExactPoint(float fPixelLocation,float* fExactX, float* fExactY,CcCalibration* CalibrationObject);
	float FindUPEdge(int iEdgeNumber,float fLoNoiseLimit, float fHiNoiseLimit);
	float FindDNEdge(int iEdgeNumber,float fLoNoiseLimit, float fHiNoiseLimit);
	float FindBestEdge(int iDirection = ANY_EDGE);

CcLineProfile Methods

This section describes each method of the CcLineProfile class in detail.

MakeProfile

Syntax

```
int MakeProfile(
  CcImage* CImage,
  CcRoiLine* CRoi,
  int iAverage);
```

Include File C_LProf.h

Description Creates a line profile of the image with respect to the given ROI.

Parameters

Name: CImage

Description: Image derived from the CcImage class and used as an input image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iAverage

Description: Number of pixels on each side of the center pixel to be averaged with the center pixel (the width of the line profile).

Notes This method uses images derived from the GLI/2-supplied CcImage class. These include 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit color images. It also works with your own images derived from these classes. For more information, see Chapter 2 on page 11.

Notes (cont.)

The CcLineProfile class is derived from the CcCurve class. After making it, you can add the line profile to the list of curves of a graph class and then easily display the graph containing the line profiles in any window.

To access the line profile data, you can call the methods of the CcCurve class. For more information, see Chapter 2 on page 11, and the example program at the end of this chapter.

When the line profile is calculated, each point on the line ROI (called a center point) is calculated separately. If the value for *iAverage* is 0, then only the points that lie directly on the line ROI are used in the line profile calculation; its width is 1 pixel wide. If the value for *iAverage* is 1, then three points are used in the calculation of each center point; the center point lying directly on the line ROI, one pixel above (or right of) the center point, and one point below (or left of) the center point. The averaged points are points taken perpendicular to the line ROI at the center point.

Return Values

–1	Unsuccessful.
0	Successful.

AverageProfile

Syntax `int AverageProfile(int iAverage);`

Include File C_LProf.h

Description Smooths the line profile by averaging each point in the line profile with its neighbors.

Parameters

Name: iAverage

Description: The number of neighbor points on each side of the center point to include in the averaging.

Notes Each point in the line profile is averaged with its neighbor points on each side. The *iAverage* parameter is the number of neighbors (on each side of the point being averaged) included in the averaging calculation.

Return Values

–1 Unsuccessful.

0 Successful.

TakeDerivative

Syntax `int TakeDerivative(int iDelta);`

Include File C_LProf.h

Description Takes a pseudo-derivative of the line profile and places the result back into the line profile.

Parameters

Name: iDelta

Description: The number of neighbor points on each side of the center point included in the calculation.

Notes This method finds the slope of the line profile at each point in the line profile, and then replaces the line profile with its pseudo-derivative. A value of 1 for *iDelta* includes the center point and each of its neighbors in the slope calculation.

Return Values

–1 Unsuccessful.

0 Successful.

GainAndOffset

Syntax

```
int GainAndOffset(
  float fGain,
  float fOffset);
```

Include File C_LProf.h

Description Applies a gain and offset to the line profile.

Parameters

Name: fGain

Description: The gain that is applied to the line profile.

Name: fOffset

Description: The offset that is applied to the line profile.

Notes This method applies the given gain and offset to each point in the line profile.

Return Values

–1 Unsuccessful.

0 Successful.

GetPixelLocationsAll

Syntax

```
PIXELGROUPING*
   GetPixelLocationsAll(void);
```

Include File C_LProf.h

Description Returns every pixel used in the calculation of the line profile.

Return Values

The points as a pixel-grouping structure.

GetPixelLocationsCenter

Syntax

```
PIXELGROUPING*
   GetPixelLocationsCenter(void);
```

Include File C_LProf.h

Description Returns pixels that were used in the calculation of the line profile as the center pixels.

Return Values

Returns the points as a pixel-grouping structure.

GetLineDistance

Syntax

```
float GetLineDistance(
  float fPixelLocationStart,
  float fPixelLocationEnd,
  CcCalibration*
   CalibrationObject);
```

Include File C_LProf.h

Description Returns the line distance from the starting point to the ending point measured along the line profile.

Parameters

Name: fPixelLocationStart

Description: The position along the line profile's x-axis at which to start taking the measurement.

Name: fPixelLocationEnd

Description: The position along the line profile's x-axis at which to stop taking the measurement.

Name: CalibrationObject

Description: A pointer to a Calibration object to use if you want the measurement in calibrated units. If this value is NULL, the measurement is in pixels.

Notes *The fPixelLocationStart* and *fPixelLocationEnd* points along the x-axis of the profile correspond exactly to the minimum and maximum measurement bars described earlier in this chapter.

Both the starting point and ending points correspond to pixel locations in the image that were originally used to create the line profile. This is the distance from the starting point to the ending point measured along the line profile, which is not necessarily the straight distance from the starting point to the ending point. To measure the straight distance between the starting and ending points, use **GetStraightDistance( )**.

Notes The *fPixelLocationStart* and *fPixelLocationEnd* points are input with subpixel accuracy and are measured from the first point in the line profile, which always has a value of 0.

Return Values

Returns the points as a pixel-grouping structure.

GetStraightDistance

Syntax

```
float GetStraightDistance(
  float fPixelLocationStart,
  float fPixelLocationEnd,
  CcCalibration*
   CalibrationObject);
```

Include File C_LProf.h

Description Returns the straight distance from the starting point to the ending point.

Parameters

Name: fPixelLocationStart

Description: The position along the line profile's x-axis at which to start taking the measurement.

Name: fPixelLocationEnd

Description: The position along the line profile's x-axis at which to stop taking the measurement.

Name: CalibrationObject

Description: A pointer to a Calibration object to use if you want the measurement in calibrated units. If this value is NULL, the measurement is in pixels.

Notes

The *fPixelLocationStart* and *fPixelLocationEnd* points along the x-axis of the profile correspond exactly to the minimum and maximum measurement bars described earlier in this chapter.

Both the starting point and ending points correspond to pixel locations in the image that were originally used to create the line profile. This is the straight distance from the starting point to the ending point, which is not necessarily the distance from the starting point to the ending point measured along the line profile. To measure the distance between the starting and ending points along the line profile, use the method **GetLineDistance( )**.

The *fPixelLocationStart* and *fPixelLocationEnd* points are input with subpixel accuracy and are measured from the first point in the line profile, which always has a value of 0.

Return Values

Returns the points as a pixel-grouping structure.

GetExactPoint

Syntax

```
int GetExactPoint(
  float fPixelLocation,
  float* fExactX,
  float* fExactY,
  CcCalibration*
   CalibrationObject);
```

Include File

C_LProf.h

Description Returns the x,y calibrated image position for the given location in the line profile.

Parameters

Name: fPixelLocation

Description: The desired position along the line profile's x-axis.

Name: fExactX

Description: The subpixel x-location in the image that corresponds to the given *fPixelLocation*.

Name: fExactY

Description: The subpixel y-location in the image that corresponds to the given *fPixelLocation*.

Name: CalibrationObject

Description: A pointer to the Calibration object to use if you want the returned point in calibrated units. If this value is NULL, the returned point is in pixels.

Notes The *fPixelLocation* point along the x-axis of the profile corresponds exactly to the minimum or maximum measurement bars described earlier in this chapter.

Each point in the line profile (*fPixelLocation*) corresponds to a pixel location in the image that was originally used to create the line profile. This method returns the corresponding pixel location (*fExactX*, *fExactY*) for the given line profile location (*fPixelLocation*). All measurements have subpixel accuracy.

Notes (cont.) *fPixelLocation* is an input with subpixel accuracy and is measured from the first point in the line profile, which always has a value of 0.

Return Values

–1	Unsuccessful.
0	Successful.

FindUPEdge

Syntax

```
float FindUPEdge(
  int iEdgeNumber,
  float fLoNoiseLimit,
  float fHiNoiseLimit);
```

Include File C_LProf.h

Description Returns the subpixel location in the line profile for the desired up edge.

Parameters

Name:	iEdgeNumber
Description:	The desired up edge in the line profile.
Name:	fLoNoiseLimit
Description:	The value of the low noise limit.
Name:	fHiNoiseLimit
Description:	The value of the high noise limit.

Notes

Before calling this method, first make a line profile by calling the method **MakeProfile( )**, then take its second derivative by calling the method **TakeDerivative( )** twice. The position in the line profile (with a second derivative) that crosses the 0 y-axis is an edge.

You may have unwanted noise edges due to noise in your images. To eliminate these noise edges, enter a high and low noise limit (*fLoNoiseLimit* and *fHiNoiseLimit*).

An up edge is where the second derivative line profile crosses the 0 y-axis with a positive slope. The curve must start below the low noise limit, cross the 0 y-axis, and continue a constant positive slope until it reaches the high noise limit.

Return Values

–1	No edge was found.
The subpixel location of the found edge.	Edge was found.

FindDNEdge

Syntax

```
float FindDNEdge(
  int iEdgeNumber,
  float fLoNoiseLimit,
  float fHiNoiseLimit);
```

Include File

C_LProf.h

Description

Returns the subpixel location in the line profile for the desired down edge.

Parameters

Name:	iEdgeNumber
Description:	The desired down edge in the line profile.
Name:	fLoNoiseLimit
Description:	The value of the low noise limit.
Name:	fHiNoiseLimit
Description:	The value of the high noise limit.

Notes

Before calling this method, first make a line profile by calling the method **MakeProfile( )**, then take its second derivative by calling the method **TakeDerivative( )** twice. The position in the line profile (with a second derivative) that crosses the 0 y-axis is an edge.

You may have unwanted noise edges due to noise in your images. To eliminate these noise edges, enter a high and low noise limit (*fLoNoiseLimit* and *fHiNoiseLimit*).

A down edge is where the second derivative line profile crosses the 0 y-axis with a negative slope. The curve must start above the high noise limit, cross the 0 y-axis, and continue a constant negative slope until it reaches the low noise limit.

Return Values

–1	No edge was found.
The subpixel location of the found edge.	Edge was found.

FindBestEdge

Syntax

```
float FindBestEdge(
   int iDirection = ANY_EDGE);
```

Include File C_LProf.h

Description Returns the subpixel location in the line profile for the most distinct / largest edge.

Parameters

Name: iDirection

Description: The direction of the desired edge, which can be one of the following:

- ANY_EDGE – Finds the best edge in any direction.
- UP_EDGE_ONLY – Finds the best up edge.
- DN_EDGE_ONLY – Finds the best down edge.

Notes Before calling this method, first make a line profile by calling the method **MakeProfile( )**, then take its second derivative by calling the method **TakeDerivative( )** twice. The position in the line profile (with a second derivative) that crosses the 0 y-axis is an edge.

You may have several edges along the line profile. This method finds the best edge along the entire profile in the given direction.

A down edge is where the second derivative line profile crosses the 0 y-axis with a negative slope. An up edge is where the second derivative line profile crosses the 0 Y-axis with a positive slope.

Return Values

–1	No edge was found.
The subpixel location of the found edge.	Edge was found.

Example Program Using the Line Profile Tool API

This example looks for the first true down edge that the given line ROI crosses within the given image. This example is intended to show you how you could locate an edge automatically as you did interactively using the Line Profile tool, described in the *GLOBAL LAB Image/2 User's Manual*. The example returns the coordinate of the edge, in image coordinates.

Note: This example is made from code fragments with error checking removed. In an actual program, you should check return values and pointers. This example follows the example given in the *GLOBAL LAB Image/2 User's Manual*.

```
POINT* FindFirstEdge(CcImage* CImage,
   CcRoiLine* CRoi)
{
CcLineProfile* CLProfile;
//Object to perform a line profile
STPOINTS* stPoints;
//Pointer to profile curve data
static POINT PointReturn;
//POINT structure to hold edge coordinates

//Allocate objects
CLProfile = new CcLineProfile( );

//Create line profile with a line width of 11
(5+5+1)
CLProfile->MakeProfile(CImageIn1,CRoi,5);

//Smooth line profile by averaging each point with
//its neighbors
```

```
//This will average each pixel with 9 neighbors on
//‘each’ side (19 points in all)
CLProfile->AverageProfile(9);

//Take second derivative of profile to denote edges
//Use center point and 1 pixel on each side of
//center point in slope calculation
CLProfile->TakeDerivative(1); //Take first
CLProfile->TakeDerivative(1); //Take again to make
//second

//Smooth derivative of profile as not to get a
//false edge
CLProfile->AverageProfile(9);

//Apply a gain of 100 to find zero-crossing easier
CLProfile->GainAndOffset(100,0);

//Get data for derivative
stPoints = CLProfile->GetCurveData( );

//Search through data looking for a zero-crossing
//of the second derivative. We will look for a
//negative slope and a crossing larger than 20.
//This is to denote an edge and not return a false
//edge due to noise.
for(x=0; x< CLProfile->GetNumberOfPoints( )-1; x++)
{
 //Check for zero crossing with negative slope
 //(down edge)
 if( ( stPoints[x].fY > 0) && (stPoints[x+1].
   fY < 0) )
 //Is it a large zero crossing or just noise
 if( (stPoints[x].fY - stPoints[x+1]) >= 20)
  {
//Copy point of crossing
PointReturn.x=stPoints[x].fX;
```

```
PointReturn.y=stPoints[x].fY;
//Free memory
delete CLProfile;
return(&PointReturn);
  }
}
//Free memory
delete CLProfile;

return(NULL);
}
```

13

Using the Measurement Tool API

Overview of the Measurement Tool API

The API for the Measurement tool has one object only: the CcRoiGauge class. The CcRoiGauge class is designed to work within the GLI/2 environment. It is used to perform measurements on various ROI objects. The results of the measurements are returned in pixels, degrees, or the measurement units you specified in a calibration object. This class can accept ROIs from more than one image, allowing for measurements using more than one camera (in this case, calibration is required).

Any output from the CcRoiGauge class is passed in the *stMResult* structure:

```
typedef struct
{
   float fResult;
}  stMResult;
```

The *fResult* variable is used to pass the result of a measurement operation. It contains a measurement value in either pixels or calibrated units (if a calibration object is provided).

The CcRoiGauge class uses a standard constructor and destructor and the class methods listed in Table 29.

Table 29: CcRoiGauge Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcRoiGauge(void);
	~CcRoiGauge(void);

Table 29: CcRoiGauge Object Methods (cont.)

Method Type	Method Name
CcRoiGauge Class Methods	BOOL SetRoi1(CcRoiBase * InputRoi);
	BOOL SetRoi2(CcRoiBase * InputRoi);
	BOOL SetRoi3(CcRoiBase * InputRoi);
	BOOL SetImage1(CcImage * InputImage);
	BOOL SetImage2(CcImage * InputImage);
	BOOL SetImage3(CcImage * InputImage);
	BOOL SetAngle(float Angle);
	MinDistance();
	MaxDistance();
	AvgDistance();
	XCoordinate();
	YCoordinate();
	Width();
	Height();
	AngleAtMiddlePoint();
	AngleFromXaxis();
	Area();
	Perimeter();
	Distance();
	DirectedDistance();
	LineLength();
	IntersectionAngle();
	MinDirectedDistance();

Table 29: CcRoiGauge Object Methods (cont.)

Method Type	Method Name
CcRoiGauge Class Methods (cont.)	MaxDirectedDistance();
	MinOppositeDistance();
	MaxOppositeDistance();
	MinPerpendicularDistance();
	MaxPerpendicularDistance();
	Roundness();
	GreyAverage();
	RedAverage();
	GreenAverage();
	BlueAverage();
	HueAverage();
	SatAverage();
	LumAverage();
	GrayValue();
	RedValue();
	GreenValue();
	BlueValue();
	HueValue();
	SatValue();
	LumValue();
	XIntersection();

Table 29: CcRoiGauge Object Methods (cont.)

Method Type	Method Name
CcRoiGauge Class Methods (cont.)	YIntersection();
	StMResult * GetResults();
	CcList * GetMethodList();

CcRoiGauge Methods

This section describes each method of the CcRoiGauge class in detail.

SetRoi1

Syntax

```
BOOL SetRoi1(
   CcRoiBase * InputRoi );
```

Include File C_RoiGauge.h

Description Specifies input ROI number 1.

Parameters

Name: InputRoi

Description: Pointer to a GLI/2 ROI class. All ROIs are supported.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcRoiLine     *CRoiLine=new
   CcRoiLine;
RECT           Line;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;

//Line going from point 2,2 to
//10,10
Line.bottom=2;
Line.top=10;
Line.left=2;
Line.right=10;
```

Example (cont.)

```
//Set the line ROI
CRoiLine->SetRoiImageCord((VOID*)
    &Line);
//Specify input ROI 1
bStatus=CRoiGauge.SetRoi1(
    (CcRoiBase *)&CRoiLine);
```

SetRoi2

Syntax

```
BOOL SetRoi2(
   CcRoiBase * InputRoi );
```

Include File C_RoiGauge.h

Description Specifies input ROI number 2.

Parameters

Name: InputRoi

Description: Pointer to a GLI/2 ROI class. All ROIs are supported.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcRoiLine *CRoiLine=new CcRoiLine;
RECT           Line;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;
//Line going from point 2,2 to
//10,10
Line.bottom=2;
Line.top=10;
```

Example (cont.)

```
Line.left=2;
Line.right=10;

//Set the line ROI
CRoiLine->SetRoiImageCord((VOID*)
  &Line);
//Specify input ROI 2
bStatus=CRoiGauge.SetRoi2(
     (CcRoiBase *)&CRoiLine);
```

SetRoi3

Syntax

```
BOOL SetRoi3(
   CcRoiBase * InputRoi );
```

Include File C_RoiGauge.h

Description Specifies input ROI number 3.

Parameters

Name: InputRoi

Description: Pointer to GLI/2 ROI class. All ROIs are supported.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcRoiLine *CRoiLine=new CcRoiLine;
RECT           Line;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;
//Line going from point 2,2 to
//10,10
```

Example (cont.)

```
Line.bottom=2; Line.top=10;
   Line.left=2; Line.right=10;

//Set the line ROI
CRoiLine->SetRoiImageCord((VOID*)
  &Line);
//Specify input ROI 3
bStatus=CRoiGauge.SetRoi3(
    (CcRoiBase *)&CRoiLine);
```

SetImage1

Syntax

```
BOOL SetImage1(
   CcImage * InputImage );
```

Include File C_RoiGauge.h

Description Specifies input image number 1.

Images are used only for color / grayscale pixel-averaging measurements. Any calibration object attached to an image is retrieved and used by the Measurement tool.

Parameters

Name: InputImage

Description: Pointer to a GLI/2 Image class.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcImage       *CImage;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;

//Fill the image somehow
. . . .
//Specify input image 1
bStatus=CRoiGauge.SetImage1(CImage
   );
```

SetImage2

Syntax

```
BOOL SetImage2(
   CcImage * InputImage );
```

Include File C_RoiGauge.h

Description Specifies input image number 2.

Images are used only for color / grayscale pixel-averaging measurements. Any calibration object attached to an image is retrieved and used by the Measurement tool.

Parameters

Name: InputImage

Description: Pointer to a GLI/2 Image class.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcImage       *CImage;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;

//Fill the image somehow
. . . .
//Specify input image 2
bStatus=CRoiGauge.SetImage2(CImage
   );
```

SetImage3

Syntax

```
BOOL SetImage3(
   CcImage * InputImage );
```

Include File C_RoiGauge.h

Description Specifies input image number 3.

Images are used only for color / grayscale pixel-averaging measurements. Any calibration object attached to an image is retrieved and used by the Measurement tool.

Parameters

Name: InputImage

Description: Pointer to a GLI / 2 Image class.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
CcImage       *CImage;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;

//Fill the image somehow
. . . .
//Specify input image 3
bStatus=CRoiGauge.SetImage3(CImage
   );
```

SetAngle

Syntax

```
BOOL SetAngle(
   float Angle);
```

Include File C_RoiGauge.h

Description Specifies the angle used for directed and opposite measurements.

For more information, refer to the **DirectedDistance**, **MinDirectedDistance**, **MaxDirectedDistance**, **MinOppositeDistance**, and **MaxOppositeDistance** methods.

Parameters

Name: Angle

Description: The angle value in degrees. The value can range from 0 to 359.

Return Values

TRUE Input was valid.

FALSE Input was invalid.

Example The following is a sample code fragment:

```
float          Angle;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;

//Set the angle
Angle = 23.0;

//Specify the angle
bStatus=CRoiGauge.SetAngle(Angle);
```

MinDistance

Syntax

```
MinDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the minimum distance between two ROI objects.

The algorithm finds the two points, each belonging to a separate ROI, that are closest to each other. Freehand, ellipse, and point ROIs are supported.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float     Angle;
CcImage   *CImageIn1,*CImageIn2;
CcImage   *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOO        bStatus;
```

Example (cont.)

```
CcRoiGauge CRoiGauge;
stMResult  *pResult;
//holds result of a measurement

//Fill the images and ROIs with
   data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.MinDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

MaxDistance

Syntax

```
MaxDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the maximum distance between two ROI objects.

Description (cont.) The algorithm finds the two points, each belonging to a separate ROI, that are the farthest apart from each other. Freehand, ellipse, and point ROIs are supported.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float      Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
CcRoiGauge  CRoiGauge;
stMResult   *pResult;
//holds result of a measurement

//Fill the images and rois with
   data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.MaxDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

AvgDistance

Syntax

```
AvgDistance(
   );
```

Include File

C_RoiGauge.h

Description

Computes the average distance between two ROI objects.

Measurements between point and poly freehand ROIs and between point and freehand ROIs are supported.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
CcRoiGauge  CRoiGauge;
stMResult   *pResult;
//holds result of a measurement

//Fill the images and rois with
   data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
```

Example (cont.)

```
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.AvgDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

XCoordinate

Syntax

```
XCoordinate(
   );
```

Include File C_RoiGauge.h

Description Returns the x-axis coordinate value for a point ROI.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
CcRoiGauge  CRoiGauge;
stMResult   *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
```

Example (cont.)

```
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.XCoordinate();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

YCoordinate

Syntax

```
YCoordinate(
   );
```

Include File

C_RoiGauge.h

Description

Returns the y-axis coordinate value for a point ROI.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
```

Example (cont.)

```
CcRoiGauge  CRoiGauge;
stMResult  *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
//the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.YCoordinate();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


Width

Syntax

```
Width(
   );
```

Include File C_RoiGauge.h

Description For an ellipse or rectangle ROI object, returns the width (dimension with respect to the x-axis).

For a line, poly freehand, or freehand ROI object, returns the width of the boundary box that encompasses the ROI.

Parameters None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;

CcRoiGauge  CRoiGauge;
stMResult  *pResult;
//holds result of a measurement

//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.Width();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

Height

Syntax

```
Height(
   );
```

Include File

C_RoiGauge.h

Description

For an ellipse or rectangle ROI object, returns the height (dimension with respect to the y-axis).

For a line, poly freehand, or freehand ROI object, returns the height of the boundary box that encompasses the ROI.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
CcRoiGauge  CRoiGauge;
stMResult  *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
```

Example (cont.)

```
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.Height();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

AngleAtMiddlePoint

Syntax

```
AngleAtMiddlePoint(
   );
```

Include File

C_RoiGauge.h

Description

Computes the angle between two vectors formed by three point ROIs. The first vector points from the middle point ROI to the first point ROI; the second vector points from middle point ROI to the last point ROI. The angle is formed by going in a counterclockwise direction from the first vector to the second vector and can range from 0 to 360 degrees.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float        Angle;
CcImage     *CImageIn1,
             *CImageIn2;
CcImage     *CImageIn3;
```

Example (cont.)

```
CcRoiBase   *CRoi1, *CRoi2,
            *CRoi3;
BOOL         bStatus;
CcRoiGauge   CRoiGauge;
stMResult   *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.AngleAtMiddlePoint();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


AngleFromXaxis

Syntax

```
AngleFromXaxis(
   );
```

Include File C_RoiGauge.h

Description Computes the angle between the x-axis and a line ROI or between the x-axis and a line formed by two point ROIs.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class

CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.AngleFromXaxis();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

Area

Syntax

```
Area(
   );
```

Include File

C_RoiGauge.h

Description

Computes the area of a rectangle, ellipse, poly freehand, or freehand ROI.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL          bStatus;
CcRoiGauge    CRoiGauge;
stMResult    *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Invoke the measurement method
CRoiGauge.Area();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

Perimeter

Syntax

```
Perimeter(
   );
```

Include File C_RoiGauge.h

Description Computes the perimeter of a rectangle, ellipse, poly freehand, or freehand ROI.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
   data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
```

Example (cont.)

```
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.Perimeter();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

Distance

Syntax

```
Distance(
   );
```

Include File

C_RoiGauge.h

Description

Computes the distance between two point ROIs or between a point ROI and a line ROI. To compute the distance between a point ROI and a line ROI, the algorithm first creates a new line that passes through the point ROI and is perpendicular to the line ROI, extending the line ROI if necessary. The algorithm then calculates the distance between the point ROI and the intersection point between the line ROI and the new line.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
```

Example (cont.)

```
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult      *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.Distance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

DirectedDistance

Syntax

```
DirectedDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the directed distance between two point ROIs. To do this, the algorithm creates two parallel lines, both perpendicular to a line at the specified angle, and shifts the lines until each line crosses one of the point ROIs. The algorithm then creates a third line that is perpendicular to the two parallel lines. The directed distance is the distance between the two parallel lines.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float       Angle;
CcImage    *CImageIn1, *CImageIn2;
CcImage    *CImageIn3;
CcRoiBase  *CRoi1, *CRoi2, *CRoi3;
BOOL        bStatus;
CcRoiGauge  CRoiGauge;
stMResult  *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
   data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Invoke the measurement method
CRoiGauge.DirectedDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

LineLength

Syntax

```
LineLength(
   );
```

Include File

C_RoiGauge.h

Description

Computes the distance between the end-points of a line ROI.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
```

Example (cont.)

```
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.LineLength();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

IntersectionAngle

Syntax

```
IntersectionAngle(
   );
```

Include File C_RoiGauge.h

Description Computes the angle formed by two line ROIs.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
```

Example (cont.)

```
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.IntersectionAngle();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

MinDirectedDistance

Syntax

```
MinDirectedDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the minimum directed distance between either a point ROI and a freehand ROI or between two freehand ROIs. For more information, refer to the **DirectedDistance** method.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
```

Example (cont.)

```
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL           bStatus;
CcRoiGauge     CRoiGauge;
stMResult     *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.MinDirectedDistance();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


MaxDirectedDistance

Syntax

```
MaxDirectedDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the maximum directed distance between either a point ROI and a freehand ROI or between two freehand ROIs. For more information, refer to the **DirectedDistance** method.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.MaxDirectedDistance();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

MinOppositeDistance

Syntax

```
MinOppositeDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the minimum opposite distance between two points, each on a different freehand ROI, or between a point ROI and a point on a freehand ROI. To do this, the algorithm creates a series of lines that are parallel to the specified angle and that cross both ROIs. The algorithm measures the distance between the intersection points on each line and then returns the minimum distance.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL            bStatus;
CcRoiGauge      CRoiGauge;
stMResult      *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Invoke the measurement method
CRoiGauge.MinOppositeDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

MaxOppositeDistance

Syntax

```
MaxOppositeDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the maximum opposite distance between two points, each on a different freehand ROI, or between a point ROI and a point on a freehand ROI. To do this, the algorithm creates a series of lines that are parallel to the specified angle and that cross both ROIs. The algorithm measures the distance between the intersection points on each line and then returns the maximum distance.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
```

Example (cont.)

```
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.MaxOppositeDistance();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


MinPerpendicularDistance

Syntax

```
MinPerpendicularDistance(
   );
```

Include File C_RoiGauge.h

Description Computes the minimum perpendicular distance between a line ROI and an ellipse ROI or between a line ROI and a freehand ROI. To do this, the algorithm creates a series of lines that are perpendicular to the line ROI and that cross the ellipse or freehand ROI. The algorithm measures the distance between the intersection points on each line and then returns the minimum distance.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Invoke the measurement method
CRoiGauge.MinPerpendicularDistance
   ();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

MaxPerpendicularDistance

Syntax

```
MaxPerpendicularDistance(
   );
```

Include File

C_RoiGauge.h

Description

Computes the maximum perpendicular distance between a line ROI and an ellipse ROI or between a line ROI and a freehand ROI. To do this, the algorithm creates a series of lines that are perpendicular to the line ROI and that cross the ellipse or freehand ROI. The algorithm measures the distance between the intersection points on each line and then returns the maximum distance.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge     CRoiGauge;
stMResult      *pResult;
```

Example (cont.)

```
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.MaxPerpendicularDistance
   ();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

Roundness

Syntax

```
Roundness(
   );
```

Include File C_RoiGauge.h

Description Computes the degree of roundness of a rectangle, ellipse, poly freehand, or freehand ROI. The result of the measurement operation is less than or equal to 1, where a value of 1 indicates that the ROI is perfectly circular. The tool uses the following formula:

Roundness = (4*Pi*Area) / (Perimeter^2)

Parameters None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
//the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.Roundness();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

GreyAverage

Syntax

```
GreyAverage(
   );
```

Include File

C_RoiGauge.h

Description

Computes the average grayscale value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with any type of image.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
```

Example (cont.)

```
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.GreyAverage();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

RedAverage

Syntax

```
RedAverage(
   );
```

Include File

C_RoiGauge.h

Description

Computes the average red value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with RGB images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
```

Example (cont.)

```
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.RedAverage();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

GreenAverage

Syntax

```
GreenAverage(
   );
```

Include File C_RoiGauge.h

Description Computes the average green value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with RGB images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement

//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.GreenAverage();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


BlueAverage

Syntax

```
BlueAverage(
   );
```

Include File C_RoiGauge.h

Description Computes the average blue value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with RGB images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.BlueAverage();
```

Example (cont.)

```
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

HueAverage

Syntax

```
HueAverage(
   );
```

Include File

C_RoiGauge.h

Description

Computes the average hue value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with HSL images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
```

Example (cont.)

```
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.HueAverage();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

SatAverage

Syntax

```
SatAverage(
   );
```

Include File

C_RoiGauge.h

Description

Computes the average saturation value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with HSL images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
```

Example (cont.)

```
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the measurement method
CRoiGauge.SatAverage();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


LumAverage

Syntax

```
LumAverage(
   );
```

Include File C_RoiGauge.h

Description Computes the average luminance value of all pixels underneath a line, poly line, or freehand line ROI or within a rectangle, ellipse, poly freehand, or freehand ROI. Point ROIs are not supported. This method works with HSL images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .

// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the measurement method
CRoiGauge.LumAverage();
```

Example (cont.)

```
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

GrayValue

Syntax

```
GrayValue(
   );
```

Include File

C_RoiGauge.h

Description

Returns the gray value of the pixel underneath a point ROI. This method works with any type of image. If this method is used with an RGB image, the luminance value of the RGB pixel is returned.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
```

Example (cont.)

```
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.GrayValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

RedValue

Syntax

```
RedValue(
   );
```

Include File

C_RoiGauge.h

Description

Returns the red value of the pixel underneath a point ROI. This method works with RGB images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
```

Example (cont.)

```
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.RedValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


GreenValue

Syntax

```
GreenValue(
   );
```

Include File C_RoiGauge.h

Description Returns the green value of the pixel underneath a point ROI. This method works with RGB images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
```

Example (cont.)

```
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.GreenValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

BlueValue

Syntax

```
BlueValue(
   );
```

Include File C_RoiGauge.h

Description Returns the blue value of the pixel underneath a point ROI. This method works with RGB images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
   data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.BlueValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


HueValue

Syntax

```
HueValue(
   );
```

Include File C_RoiGauge.h

Description Returns the hue value of the pixel underneath a point ROI. This method works with HSL images.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.HueValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

SatValue

Syntax

```
SatValue(
   );
```

Include File

C_RoiGauge.h

Description

Returns the saturation value of the pixel underneath a point ROI. This method works with HSL images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
   data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Invoke the gauging method
CRoiGauge.SatValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

LumValue

Syntax

```
LumValue(
   );
```

Include File

C_RoiGauge.h

Description

Returns the luminance value of the pixel underneath a point ROI. This method works with HSL images.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
//CRoi1 must be a point ROI
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
```

Example (cont.)

```
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.LumValue();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

XIntersection

Syntax

```
XIntersection(
   );
```

Include File

C_RoiGauge.h

Description

Returns the X-coordinate of the intersection point between two line ROIs. This measurement is done at the subpixel level. This method works with any type of image.

Parameters

None

Return Values

Values are returned by the **GetResults** method, described on page 477.

Example

The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
```

Example (cont.)

```
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
//Should be a line ROI
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.XIntersection();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

YIntersection

Syntax

```
YIntersection(
   );
```

Include File C_RoiGauge.h

Description Returns the Y coordinate of the intersection point between two line ROIs. This measurement is done at the subpixel level. This method works with any type of image.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//will hold result of a
//measurement
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
//Should be a line ROI
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.YIntersection();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


GetResults

Syntax

```
StMResult * GetResults(
   );
```

Include File C_RoiGauge.h

Description Returns a pointer to the results structure. It can be invoked after executing one of the measurement methods.

Parameters None

Return Values

A pointer to the *stMResult* structure containing the measurement result.	Successful.
The *fresult* member of the *stMResult* structure will contain –1.	Unsuccessful.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
CcList *TheList;
CcGaugingMethod  *CMeasurement;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
```

Example (cont.)

```
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Get the list of measurement
// methods
TheList = CRoiGauge.GetMethodList
   ();
// Get the measurement object from
   the // list, based on the name
CMeasurement=(CcGaugingMethod *)
   TheList->GetViaName(“Min
   Distance”);
if (CMeasurement == NULL)
{
   Error("Can't get the
      measurement method!");
   return;
}
// Invoke the measurement method
(CRoiGauge.*CMeasurement->
       GaugingMethod)();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```


GetMethodList

Syntax

```
CcList * GetMethodList(
   );
```

Include File C_RoiGauge.h

Description Returns a pointer to the list of measurement method pointers. This list provides a way to associate text names of the measurement methods with pointers to these methods so that you can invoke the measurement methods based on their text names. The text names are defined at the top of the C_RoiGauge.h header file.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 477.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
CcList *TheList;
CcGaugingMethod  *CMeasurement;
stMResult *pResult;
//Holds result of a measurement
//Fill the images and ROIs with
//data
. . . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetImage2(CImageIn2);
CRoiGauge.SetImage3(CImageIn3);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
```

Example (cont.)

```
// Get the list of measurement
// methods
TheList = CRoiGauge.GetMethodList(
   );
// Get the measurement object from
// the list, based on the name
CMeasurement=(CcGaugingMethod *)
  TheList->GetViaName(“Min
   Distance”);

if (CMeasurement == NULL)
{
   Error("Can't get the
       measurement method!");
 return;
}
// Invoke the measurement method
(CRoiGauge.*CMeasurement->
      GaugingMethod)();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

14

Using the Morphology Tool API

Overview of the Morphology Tool API

The API for the Morphology tool has one object only: the CcMorphology class. This tool performs a morphological operation on a binary input image (derived from class CcImage), and places the result in an output image. The operation is performed with respect to the given ROI (derived from class CcRoiBase).

The CcMorphology object uses a standard constructor and destructor and the class methods listed in Table 30.

Table 30: CcMorphology Class Methods

Method Type	Method Name
Constructor & Destructor Methods	CcMorphology( );
	~ CcMorphology( );
CcMorphology Class Methods	int SetKernel(STMORPHKERNEL* stKer);
	int GetKernel(STMORPHKERNEL* stKer);
	int RestoreKernel(char* cFileName);
	int SaveKernel(char* cFileName);
	int OpenBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi, int iIterations);
	int CloseBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi, int iIterations);
	int ErodeBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi, int iIterations);
	int DilateBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi, int iIterations);

Table 30: CcMorphology Class Methods (cont.)

Method Type	Method Name
CcMorphology Class Methods (cont.)	int SkeletonBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi);
	int WatershedBinary(CcBinaryImage* CImageIn, CcBinaryImage* CImageOut,CcRoiBase* CRoi);
	int WaterShedDistance(CcBinaryImage* CImageIn, CcImage* CImageOut,CcRoiBase* Croi);

CcMorphology Methods

This section describes each method of the CcMorphology class in detail.

SetKernel

Syntax

```
int SetKernel(
   STMORPHKERNEL* stKer);
```

Include File C_Morph.h

Description Sets the kernel for the performed morphological operation.

Parameters

Name: stKer

Description: Pointer to a structure of type STMORPHKERNEL. This parameter holds information for the kernel.

Notes This method sets the kernel information that is used by the class when a morphological operation that uses a kernel is called. The kernel is of type STMORPHKERNEL and is defined as follows:

```
struct MKernelTag {
int iWidth;
int iHeight;
int iXCenterOffset;
int iYCenterOffset;
int Kernel[7][7];
};
typedef MKernelTag STMORPHKERNEL;
```

Notes (cont.) The entries for this structure are as follows:

- **iWidth** – The width of the kernel in pixels.
- **iHeight** – The height of the kernel in pixels.
- **iXCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the x-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **iYCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the y-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **Kernel[7][7]** – A 7 x 7 array of values to hold the coefficients of the kernel. Depending on the width and height of the kernel, not all of these values can be used.

Return Values

–1	Unsuccessful.
0	Successful.

GetKernel

Syntax

```
int GetKernel(
   STMORPHKERNEL* stKer);
```

Include File C_Morph.h

Description Returns the kernel for the performed morphological operation.

Parameters

Name: stKer

Description: Pointer to a a structure of type STMORPHKERNEL. This parameter holds information for the kernel.

Notes

This method gets the kernel information that is used by the class when a morphological operation that uses a kernel is called. The kernel is of type STMORPHKERNEL and is defined as follows:

```
struct MKernelTag {
int iWidth;
int iHeight;
int iXCenterOffset;
int iYCenterOffset;
int Kernel[7][7];
};
typedef MKernelTag STMORPHKERNEL;
```

The entries for this structure are as follows:

- **iWidth** – The width of the kernel in pixels.
- **iHeight** – The height of the kernel in pixels.
- **iXCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the x-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.
- **iYCenterOffset** – The offset from the lower-left corner (0,0) of the kernel to the y-location of the active pixel (usually thought of as the center pixel). For a 3 x 3 centered kernel, this value is 1.

Notes (cont.)

- **Kernel[7][7]** – A 7 x 7 array of values to hold the coefficients of the kernel. Depending on the width and height of the kernel, not all of these values can be used.

Return Values

–1	Unsuccessful.
0	Successful.

RestoreKernel

Syntax

```
int RestoreKernel(
   char* cFileName);
```

Include File C_Morph.h

Description Restores a kernel saved on disk.

Parameters

Name:	cFileName
Description:	Full path name of the file that contains the kernel you wish to restore.

Notes This method opens a kernel stored in the file *cFileName*. It restores all information for the kernel defined in the structure STMORPHKERNEL, not just the coefficients of the kernel.

Return Values

–1	Unsuccessful.
0	Successful.

SaveKernel

Syntax
```
int SaveKernel(char* cFileName);
```

Include File C_Morph.h

Description Saves the kernel to disk.

Parameters

cFileName	Full path name of the file that is created to hold the kernel information.
Name:	This method saves the kernel that is used by the CcMorphology class to disk. It saves all information given in the structure STMORPHKERNEL, not just the kernel coefficients. You can retrieve this information using **RestoreKernel( )**.
Description:	

Return Values

–1	Unsuccessful.
0	Successful.

OpenBinary

Syntax
```
int OpenBinary(
  CcBinaryImage* CImageIn,
  CcBinaryImage* CImageOut,
  CcRoiBase* CRoi,
  int iIterations);
```

Include File C_Morph.h

Description Performs the morphological opening operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: CImageOut

Description: Binary image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iIterations

Description: The number of openings to perform.

Notes This method performs the morphological opening operation. It uses information in the structure STMORPHKERNEL, which is set using the methods **SetKernel( )** and / or **RestoreKernel( )**. It performs the opening operation the number of times specified by *iIterations*.

Return Values

–1 Unsuccessful.

0 Successful.

CloseBinary

Syntax

```
int CloseBinary(
  CcBinaryImage* CImageIn,
  CcBinaryImage* CImageOut,
  CcRoiBase* CRoi,
  int iIterations);
```

Include File C_Morph.h

Description Performs the morphological closing operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: CImageOut

Description: Binary image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iIterations

Description: The number of closings to perform.

Notes This method performs the morphological closing operation. It uses information in the structure STMORPHKERNEL, which is set using **SetKernel( )** and / or **RestoreKernel( )**. It performs the closing operation the number of times specified by *iIterations*.

Return Values

–1 Unsuccessful.

0 Successful.

ErodeBinary

Syntax

```
int ErodeBinary(
  CcBinaryImage* CImageIn,
  CcBinaryImage* CImageOut,
  CcRoiBase* CRoi,
  int iIterations);
```

Include File C_Morph.h

Description Performs the morphological erosion operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: CImageOut

Description: Binary image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: iIterations

Description: The number of erosions to perform.

Notes This method performs the morphological erosion operation. It uses information in the structure STMORPHKERNEL, which is set using **SetKernel( )** and / or **RestoreKernel( )**. It performs the erosion operation the number of times specified by *iIterations*.

Return Values

–1	Unsuccessful.
0	Successful.

DilateBinary

Syntax

```
int DilateBinary(
  CcBinaryImage* CImageIn,
  CcBinaryImage* CImageOut,
  CcRoiBase* CRoi,
  int iIterations);
```

Include File C_Morph.h

Description Performs the morphological dilation operation.

Parameters

Name:	CImageIn
Description:	Binary image derived from the CcImage class. It is used as the input image.
Name:	CImageOut
Description:	Binary image derived from the CcImage class. It is used as the output image.
Name:	CRoi
Description:	ROI area in which to perform the operation.
Name:	iIterations
Description:	The number of dilations to perform.

Notes This method performs the morphological dilation operation. It uses information in the structure STMORPHKERNEL, which is set using **SetKernel( )** and / or **RestoreKernel( )**. It performs the dilation operation the number of times specified by **iIterations**.

Return Values

–1 Unsuccessful.

0 Successful.

SkeletonBinary

Syntax

```
int SkeletonBinary(
  CcBinaryImage* CImageIn,
  CcBinaryImage* CImageOut,
  cRoiBase* CRoi);
```

Include File C_Morph.h

Description Performs the morphological skeletonization operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: ImageOut

Description: Binary image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Notes This method performs the morphological skeletonization operation. It does not use a kernel to perform the operation.

Return Values

–1 Unsuccessful.

0 Successful.

WatershedBinary

Syntax

```
int WatershedBinary(
  cBinaryImage* CImageIn,
  cBinaryImage* CImageOut,
  cRoiBase* CRoi);
```

Include File C_Morph.h

Description Performs the morphological watershed operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: CImageOut

Description: Binary image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Notes This method performs the morphological watershed operation. It does not use a kernel to perform the operation. This operation calls the public class method **WaterShedDistance( )** as part of its calculation. You can view the distance calculation used by this operation by calling **WaterShedDistance( )**.

Return Values

–1 Unsuccessful.

0 Successful.

WaterShedDistance

Syntax

```
int WaterShedDistance(
  CcBinaryImage* CImageIn,
  CcImage* CImageOut,
  CcRoiBase* CRoi);
```

Include File C_Morph.h

Description Performs only the distance portion of the watershed operation.

Parameters

Name: CImageIn

Description: Binary image derived from the CcImage class. It is used as the input image.

Name: CImageOut

Description: Image derived from the CcImage class. It is used as the output image.

Name: CRoi

Description: ROI area in which to perform the operation.

Notes

This method performs the distance portion of the morphological watershed operation. It does not use a kernel to perform the operation. This public operation is called from **WatershedBinary( )** as part of its calculation. You can view the distance calculation used by the **WatershedBinary( )** operation by calling this method.

This method calculates the distance from each point in a foreground particle to its closest perimeter point. The distance calculated is stored in the pixel value for the given point. Thus, the output of this method is not a binary image. You can use an 8-bit or 32-bit grayscale image as the output image.

Return Values

–1	Unsuccessful.
0	Successful.

Example Program Using the Morphology Tool API

This example program performs an opening operation on an input image using the default 3 x 3 flat kernel. The operation is performed with respect to the given ROI. You could perform this operation to clean an image just before performing blob analysis on it.

Note: This example is made from code fragments with error checking removed. In an actual program, you should check return values and pointers.

```
int CleanFunction(CcBinaryImage* CImage,
     CcRoiBase* CRoi)
{
CcMorphology* CMorph;
//Object to perform morphological operation

//Create objects
CMorph=new CcMorphology( );

//Run the Opening morphological operation.
// We will put the result back into the same image.
// We will only perform 1 iteration.
CMorph->OpenBinary(CImage,CImage,CRoi,1);

//Free memory
delete CMorph;
return(0);
}
```

15

Using the Picture Tool API

Overview of the PictureTool API

The Picture tool is derived from the base picture class CcPictureTool. CcPictureTool allows you to acquire frames from a frame grabber board.

All methods of the Picture tool work in approximately the same way for all frame grabber boards. If your board does not support a particular operation, the method returns either a value less than 0 or FALSE.

The CcPictureTool class uses a standard constructor and destructor and the class methods listed in Table 31.

Table 31: CcPictureTool Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcPictureTool(void);
	~CcPictureTool(void);
CcPictureTool Class Methods	int Initialize();
	int GetPluginList(CcStringList *pPluginList);
	int OpenPlugin(LPCSTR szPluginName);
	int ClosePlugin();
	BOOL IsPluginOpen();
	int GetDeviceList(CcStringList *pDeviceList);
	int OpenDevice(LPCSTR szDeviceName);
	int CloseDevice();
	BOOL IsDeviceOpen();
	BOOL IsDeviceCapSupported(int iDeviceCap);
	int TakePicture(CcImage *pImage);

Table 31: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int CancelTakePicture();
	int GetInputChannelCount(int *piCount);
	int SetInputChannel(int iChannel);
	int GetInputChannel(int *piChannel);
	int GetCompatibleImage(CcImage **ppImage);
	int SetImageAverage(int iImageCount);
	int EnableTrigger();
	int DisableTrigger();
	BOOL IsTriggerEnabled();
	int SetTriggerTransition(int iTrigTrans);
	int GetTriggerTransition(int *piTrigTrans);
	int EnableTimeStamping();
	int DisableTimeStamping();
	BOOL IsTimeStampingEnabled();
	int SetTimeout(int iTimeOut);
	int GetTimeout(int *piTimeOut);
	int SetImageDimensions(int iWidth, int iHeight);
	int GetImageDimensions(int *piWidth, int *piHeight);
	int GetScaledImageDimensions(int *piWidth, int *piHeight);
	int SetImageWidth(int iWidth);
	int GetImageWidth(int *piWidth);
	int GetScaledImageWidth(int *piWidth);

Table 31: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int SetImageHeight(int iHeight);
	int GetImageHeight(int *piHeight);
	int GetScaledImageHeight(int *piHeight);
	int GetImageLimits(DT_IMAGELIMITS *pImageLimits);
	int SetImageType(int iImageType);
	int GetImageType(int *piImageType);
	BOOL IsImageTypeSupported(int iImageType);
	int SetFrameType(int iFrameType);
	int GetFrameType(int *piFrameType);
	BOOL IsFrameTypeSupported(int iImageType);
	int SetImageScale(int nHorzScale, int iVertScale);
	int GetImageScale(int *piHorzScale, int *piVertScale);
	int SetImageHorzScale(int iHorzScale);
	int GetImageHorzScale(int *piHorzScale);
	int SetImageVertScale(int iVertScale);
	int GetImageVertScale(int *piVertScale);
	int TimedPictureSaveToAVI(CcAVI pAVI,int iFrameCount, int iTimeDelay);
	int TimedPictureSaveToDisc(LPCSTR szDir, LPCSTR szBaseFileName, int iFrameCount, int iCountStart, int iTimeDelay);
	int TimedPictureSaveToMemory(LPCSTR szBaseImageName, int iFrameCount, int iCountStart, int iTimeDelay, CcList *pImageList);
	int CancelTimedSave();

Table 31: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int StartLiveVideo(HWND hParent);
	int StopLiveVideo();
	BOOL IsLiveVideoRunning();
	int EnablePassthruAcquire(BOOL bEnable);
	int StartPassthru(HWND hParent);
	int StopPassthru();
	BOOL IsPassthruRunning();
	int GetDeviceSettingsFileExt(LPSTR szFileExt);
	GetDeviceSettingsFileDesc(LPSTR szFileDesc);
	int LoadDeviceSettings(LPSTR szFileName);
	int SaveDeviceSettings(LPCSTR szFileName);
	int SetDeviceSettings(DEVICESETTINGS *pSettings);
	int GetDeviceSettings(DEVICESETTINGS *pSettings);
	int ShowDeviceSettingsDialog(HWND hParent);
	void GetErrorMessage(LPSTR szMsgBuff, int iBuffSize);

CcPictureTool Methods

This section describes each method of the CcPictureTool class in detail.

Initialize

Syntax

```
int Initialize(
   );
```

Include File C_PicTool.h

Description Initializes the CcPictureTool object. You must call this method before you call any other CcPictureTool methods.

Parameters None

Notes None

Return Values

< 0	The CcPictureTool object could not be initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
Result = PictureTool.Initialize();
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

GetPluginList

Syntax

```
int GetPluginList(
   CcStringList *pPluginList
   );
```

Include File C_PicTool.h

Description Returns the list of plug-ins that are installed in the system.

Parameters

Name: pPluginList

Description: A pointer to an object of type CcStringList.

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

This method replaces the contents of the CcStringList object with a list of strings that identify the plug-ins installed in the system. You can use these strings to identify plug-ins in subsequent calls to the **OpenPlugin** method.

Return Values

< 0	The plug-in list could not be obtained, the input argument is NULL, or the CcPictureTool object has not been initialized (by calling the **Initialize** method).
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// String list object.
CcStringList  PluginList;
int Result;
Result = PictureTool.GetPluginList
   (&PluginList );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

OpenPlugin

Syntax

```
int OpenPlugin(
   LPCSTR szPluginName
   );
```

Include File C_PicTool.h

Description Opens the specified plug-in.

Parameters

Name: szPluginName

Description: A zero-terminated constant string that identifies the plug-in that you want to open.

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Notes (cont.) You can retrieve a list of the plug-ins installed in your system by calling the **GetPluginList** method.

Return Values

< 0	The plug-in could not be opened, the input string is NULL, or the CcPictureTool object has not been initialized (by calling the **Initialize** method).
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
Result = PictureTool.OpenPlugin
   (“MyPlugin”);
if ( Result < 0 )
{
//Operation failed - handle error.
}
```


ClosePlugin

Syntax
```
int ClosePlugin(
   );
```

Include File C_PicTool.h

Description Closes the currently open plug-in.

Parameters None

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Return Values

< 0 No plug-in is currently open or the CcPictureTool object has not been initialized (by calling the **Initialize** method).

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
Result= PictureTool.ClosePlugin();
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

IsPluginOpen

Syntax

```
BOOL IsPluginOpen(
   );
```

Include File C_PicTool.h

Description Determines whether a plug-in is currently open on a CcPictureTool object.

Parameters None

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the "DT Picture Tool Plugins" category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Return Values

TRUE A plug-in is currently open on the CcPictureTool object.

FALSE No plug-in is open on the CcPictureTool object.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsOpen;
bIsOpen = PictureTool.IsPlugin
   Open();
if ( bIsOpen == TRUE )
{
// A plugin is currently open.
}
```


GetDeviceList

Syntax

```
int GetDeviceList(
   CcStringList *pDeviceList
   );
```

Include File C_PicTool.h

Description Returns a list of devices that are supported by the currently open plug-in.

Parameters

Name: pDeviceList

Description: A pointer to an object of type CcStringList.

Notes

A plug-in must support at least one device to be useful.

A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

This method replaces the contents of the CcStringList object with a list of strings that identify the devices supported by the currently open plug-in. You can use these strings to identify a particular device in subsequent calls to the **OpenDevice** method.

Return Values

< 0 The device list could not be obtained, a plug-in is not currently open, the input argument is NULL, or the CcPictureTool object has not been initialized (by calling the **Initialize** method).

0 Successful.

Example

The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// String list object.
CcStringList DeviceList;
int Result;
```

Example (cont.)

```
Result = PictureTool.GetDeviceList
   (&DeviceList);
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

OpenDevice

Syntax

```
int OpenDevice(
   LPCSTR szDeviceName
   );
```

Include File C_PicTool.h

Description Opens the specified device on the currently open plug-in.

Parameters

Name: szDeviceName

Description: A zero-terminated constant string that identifies the device that you want to open. You can retrieve a list of the devices supported by the currently open plug-in by calling the **GetDeviceList** method.

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Return Values

< 0	The device could not be opened, a plug-in is not currently open, the input string is NULL, or the CcPictureTool object has not been initialized (by calling the **Initialize** method).
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
Result = PictureTool.OpenDevice
   ("MyDevice");
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

CloseDevice

Syntax

```
int CloseDevice(
   );
```

Include File C_PicTool.h

Description Closes the currently open device on the currently open plug-in.

Parameters None

Notes A plugin is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Return Values

< 0	No device is currently open, no plug-in is currently open, or the CcPictureTool object has not been initialized (by calling the **Initialize** method).
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
Result= PictureTool.CloseDevice();
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

IsDeviceOpen

Syntax

```
BOOL IsDeviceOpen(
   );
```

Include File C_PicTool.h

Description Determines whether a device is open on the currently open plug-in.

Parameters None

Notes A plug-in is any COM object that implements the IPictureTool interface and registers itself under the “DT Picture Tool Plugins” category (the category has GUID=ABEB86A1-0BD5-11d4-8AE1-00A0C9A580D6).

Return Values

TRUE — A device is open on the currently open plug-in.

FALSE — A device is not open on the currently open plug-in.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsOpen;
bIsOpen = PictureTool.IsDevice
   Open();
if ( bIsOpen == TRUE )
{
// A device is currently open.
}
```

IsDeviceCapSupported

Syntax

```
BOOL IsDeviceCapSupported(
   int iDeviceCap
   );
```

Include File C_PicTool.h

Description Determines whether the specified capability is supported by the currently open device.

Parameters

Name: iDeviceCap

Description: The device capability. The value can be one of the following:

- DEVCAP_PASSTHRU = The device supports passthru.
- DEVCAP_PASSTHRUACQUIRE = The device supports the acquisition of frames while passthru is running.
- DEVCAP_SETTINGSDIALOG = The device is configurable through an settings dialog box.
- DEVCAP_TRIGGER = The device supports triggering.
- DEVCAP_TIMEOUT = The device timeout can be programmed.
- DEVCAP_FRAMETYPE = The device frame type can be programmed.
- DEVCAP_IMAGESCALE = The device horizontal and vertical scale factors can be programmed.
- DEVCAP_CHANNELSELECT = The device input channel can be programmed.
- DEVCAP_IMAGEDIMS = The device image dimensions can be programmed.
- DEVCAP_IMAGETYPE = The device image type can be programmed.

Notes None

Return Values

TRUE — The currently open device supports the specified capability.

FALSE — The currently open device does not support the specified capability.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsSupported;
// See if the open device supports
// passthru.
bIsSupported= PictureTool.IsDevice
   CapSupported(DEVCAP_PASSTHRU);
if ( bIsSupported == TRUE )
{
//The open device supports
//passthru.
}
```

TakePicture

Syntax

```
int TakePicture(
   CcImage *pImage
   );
```

Include File C_PicTool.h

Description Acquires a frame from the currently open device and returns it in the specified image object.

Parameters

Name: pImage

Description: A pointer to an object of a supported image type that will hold the acquired frame.

Notes To retrieve an image object that is compatible with the output format of the open device (appropriate image type, width, and height) use the **GetCompatibleImage** method.

Return Values

< 0 An image could not be acquired, a device and plug-in are not currently open, the input argument is NULL, or the CcPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool  PictureTool;
// Pointer to image object.
CcImage *pImage;
if (PictureTool.GetCompatibleImage
   (&pImage ) == 0)
{
// Acquire an image.
  if (PictureTool.TakePicture
     (pImage) < 0 )
     {
          // Operation failed - handle
      //error.
     }
 //Delete image object when done.
    delete pImage;
}
```

CancelTakePicture Method

Syntax

```
int CancelTakePicture(
   );
```

Include File C_PicTool.h

Description Cancels a **TakePicture** operation while the method is waiting for a trigger to occur.

Parameters None

Notes Calling **CancelTakePicture** when triggering is not enabled has no effect.

Return Values

< 0	A device and plug-in are not currently open or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture Tool object
CcPictureTool PictureTool;
int Result;

//Cancel a TakePicture operation
Result = PictureTool.CancelTake
   Picture( );
if (Result < 0)
{
//Operation failed - handle error.
}
```

GetInputChannelCount

Syntax

```
int GetInputChannelCount(
   int *piCount
   );
```

Include File C_PicTool.h

Description Returns the number of input channels (sources) that are supported by the currently open device.

Parameters

Name: piCount

Description: A pointer to an integer that receives the channel count.

Notes None

Return Values

< 0	The channel count could not be obtained, a device and plug-in are not currently open, the input argument is NULL, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result, iCount;
Result = PictureTool.GetInput
   ChannelCount (&iCount);
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

SetInputChannel

Syntax

```
int SetInputChannel(
   int iChannel
   );
```

Include File C_PicTool.h

Description Specifies the input channel on the currently open device that you want to use.

Parameters

Name: iChannel

Description: The input channel. The value can range from 0 to *n* – 1, where *n* is the number of input channels supported by the currently open device.

Notes The first input channel supported by a device is always channel zero (0). Therefore, acceptable channel values for a device that has four input channels are 0, 1, 2 and 3.

Return Values

< 0	The specified input channel is invalid, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
// Set the current input channel
// to 2.
```

Example (cont.)

```
Result = PictureTool.SetInput
   Channel( 2 );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

GetInputChannel

Syntax

```
int GetInputChannel(
   int *piChannel
   );
```

Include File C_PicTool.h

Description Returns the current input channel on the open device.

Parameters

Name: piChannel

Description: A pointer to an integer that receives the current input channel. The value can range from 0 to $n - 1$, where n is the number of input channels supported by the currently open device.

Notes The first input channel supported by a device is always zero (0). Therefore, possible return channel values for a device that has four input channels are 0, 1, 2 and 3.

Return Values

< 0	The input channel cannot be obtained, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int iResult, iChannel;
// Get the current input channel.
iResult = PictureTool.GetInput
   Channel ( &iChannel );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

GetCompatibleImage

Syntax

```
int GetCompatibleImage(
   CcImage **ppImage
   );
```

Include File C_PicTool.h

Description Returns an image object that is compatible with the output format of the open device (appropriate image type, width, and height).

Parameters

Name:	ppImage
Description:	A pointer to a pointer to a CcImage object that will hold the newly created image.

Notes

You can use the returned image object in subsequent calls to **TakePicture**.

Call the method **GetImageType** to determine the current image type used.

Make sure that you free all image objects obtained through calls to this method.

Return Values

< 0	An image object could not be retrieved, a device and plug-in are not currently open, the input argument is NULL, or the CcPictureTool object has not been initialized.
0	Successful.

Example

The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool  PictureTool;
//Pointer to receive image object.
CcImage *pImage;
Result = PictureTool.GetCompatible
   Image( &pImage );
if ( Result < 0 )
{
//Operation failed - handle error.
}
// Delete image object when done.
delete pImage;
```

SetImageAverage

Syntax

```
int SetImageAverage(
   int iImageCount
   );
```

Include File C_PicTool.h

Description Specifies the number of images that you want to acquire from the open device and average together when you call **TakePicture**. By default, a single image is acquired from the open device.

Parameters

Name: iImageCount

Description: The number of images that you want to acquire from the open device and average together.

Notes **TakePicture** always returns the average of images in a single image object.

Return Values

< 0 The specified image average could not be set, a device and plug-in are not currently open, the input argument is invalid, or the CcPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool  PictureTool;
//Average 4 images together on the
//next call to TakePicture().
Result = PictureTool.SetImage
   Average ( 4 );
```

Example (cont.)

```
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

EnableTrigger

Syntax `int EnableTrigger();`

Include File C_PicTool.h

Description Enables triggering on the open device.

Parameters None

Notes To determine whether the open device supports triggering, use the **IsDeviceCapSupported** method with the argument DEVCAP_TRIGGER.

Return Values

< 0	The open device does not support triggering, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsSupported;
// See if the open device supports
// triggering.
bIsSupported = PictureTool.Is
   DeviceCapSupported (DEVCAP_
   TRIGGER);
```

Example (cont.)

```
if ( bIsSupported == TRUE )
{
// Enable the trigger on the open
// device.
if ( PictureTool.EnableTrigger() <
   0 )
{
//Operation failed - handle error.
}
}
```

DisableTrigger

Syntax

```
int DisableTrigger();
```

Include File

C_PicTool.h

Description

Disables triggering on the open device.

Parameters

None

Notes

To determine whether the open device supports triggering, use the **IsDeviceCapSupported** method with the argument DEVCAP_TRIGGER.

Return Values

< 0	The open device does not support triggering, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;

// Disable the trigger on the open
// device.
if (PictureTool.DisableTrigger()
   < 0 )
{
//Operation failed - handle error.
}
```

IsTriggerEnabled

Syntax `BOOL IsTriggerEnabled();`

Include File C_PicTool.h

Description Determines whether the trigger is currently enabled on the open device.

Parameters None

Notes To determine whether the open device supports triggering, use the **IsDeviceCapSupported** method with the argument DEVCAP_TRIGGER.

Return Values

TRUE The trigger is currently enabled.

FALSE The trigger is not currently enabled.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
```

Example (cont.)

```
//BOOL to receive the result.
BOOL bIsEnabled;
// Determine whether the trigger
// is enabled on the open device.
bIsEnabled = PictureTool.IsTrigger
   Enabled();
if (bIsEnabled == TRUE)
{
//The trigger is currently
//enabled.
}
```

SetTriggerTransition

Syntax

```
int SetTriggerTransition(
   int iTrigTrans
   );
```

Include File C_PicTool.h

Description Specifies the current trigger transition on the open device.

Parameters

Name: iTrigTrans

Description: Specifies the trigger transition as either `TRIGTRANS_HITOLO` for a falling-edge trigger or `TRIGTRANS_LOTOHI` for a rising-edge trigger.

Notes Call **IsDeviceCapSupported** with the flag DEVCAP_TRIGGER to check whether triggering is supported.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool  PictureTool;
//Configure the trigger to fire on
//the rising edge of the trigger
//signal.
if (PictureTool.SetTrigger
   Transition(TRIGTRANS_LOTOHI)
   < 0 )
{
//Operation failed - handle error.
}
```

GetTriggerTransition

Syntax

```
int GetTriggerTransition(
   int *piTrigTrans
   );
```

Include File C_PicTool.h

Description Returns the current trigger transition on the open device.

Parameters

Name:	piTrigTrans
Description:	A pointer to an integer that receives the current trigger transition. The trigger transition is either `TRIGTRANS_HITOLO` for a falling-edge trigger or `TRIGTRANS_LOTOHI` for a rising-edge trigger.

Notes Call **IsDeviceCapSupported** with the flag DEVCAP_TRIGGER to check whether triggering is supported.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool  PictureTool;
// Variable to receive trigger
// transition.
int iTrigTrans;
if (PictureTool.GetTrigger
   Transition(&iTrigTrans)< 0 )
{
//Operation failed - handle error.
}
```

EnableTimeStamping

Syntax `int EnableTimeStamping();`

Include File C_PicTool.h

Description Enables time stamping on the open device.

Parameters None

Notes None

Return Values

< 0	A device and plug-in are not currently open or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Enable time stamping.
if (PictureTool.EnableTime
   Stamping() < 0 )
{
//Operation failed - handle error.
}
```

DisableTimeStamping

Syntax `int DisableTimeStamping();`

Include File C_PicTool.h

Description Disables time stamping.

Parameters None

Notes None

Return Values

< 0	A device and plug-in are not currently open or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Disable time stamping.
if (PictureTool.DisableTime
   Stamping() < 0 )
{
//Operation failed - handle error.
}
```

IsTimeStampingEnabled

Syntax `BOOL IsTimeStampingEnabled();`

Include File C_PicTool.h

Description Determines whether time stamping is currently enabled on the open device.

Parameters None

Notes None

Return Values

TRUE Time stamping is currently enabled.

FALSE Time stamping is not currently enabled.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//BOOL to receive the result.
BOOL bIsEnabled;
// Determine whether time stamping
// is enabled on the open device.
bIsEnabled = PictureTool.IsTime
   StampingEnabled();
```

Example (cont.)

```
if (bIsEnabled == TRUE)
{
//Time stamping is currently
//enabled.
}
```

SetTimeout

Syntax

```
int SetTimeout(
   int iTimeOut
   );
```

Include File C_PicTool.h

Description Specifies the timeout period (in seconds).

Parameters

Name: iTimeout

Description: The timeout period.

Notes To determine whether the open device supports a programmable timeout period, use the **IsDeviceCapSupported** method with the argument DEVCAP_TIMEOUT.

Return Values

< 0	The open device does not support a programmable timeout period, a device and plug-in are not currently open, the input argument is invalid, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsSupported;
// See if the open device supports
// a programmable timeout period.
bIsSupported = PictureTool.
   IsDeviceCapSupported(
   DEVCAP_TIMEOUT );
if ( bIsSupported == TRUE )
{
// Set the timeout on the open
// device to 10 seconds.
if (PictureTool.SetTimeout( 10 ) <
   0 )
   {
//Operation failed - handle error.
   }
}
```

GetTimeout

Syntax

```
int GetTimeout(
   int *piTimeOut
   );
```

Include File C_PicTool.h

Description Returns the current timeout period (in seconds).

Parameters

Name: piTimeout

Description: A pointer to an integer that receives the timeout value.

Notes To determine whether the open device supports a programmable timeout period, use the **IsDeviceCapSupported** method with the argument DEVCAP_TIMEOUT.

Return Values

< 0 The open device does not support a programmable timeout period, a device and plug-in are not currently open, the input argument is NULL, or the CcPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsSupported;
int iTimeout;
// See if the open device supports
// a programmable timeout period.
bIsSupported=PictureTool.IsDevice
   CapSupported(DEVCAP_TIMEOUT );
if ( bIsSupported == TRUE )
{
if ( PictureTool.GetTimeout
   ( &iTimeout ) < 0 )
```

Example (cont.)

```
        {
//Operation failed - handle error.
        }
}
```

SetImageDimensions

Syntax

```
int SetImageDimensions(
   int iWidth,
   int iHeight
   );
```

Include File C_PicTool.h

Description Specifies the current dimensions of the output image.

Parameters

Name: iWidth

Description: An integer that specifies the width of the image. Use **GetImageLimits** to determine the allowable range for the image width.

Name: iHeight

Description: An integer that specifies the height of the image. Use **GetImageLimits** to determine the allowable range for the image height.

Notes None

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Set the width and height of the
//current output image.
if (PictureTool.SetImageDimensions
   (640, 480)< 0 )
{
//Operation failed - handle error.
}
```

GetImageDimensions

Syntax

```
int GetImageDimensions(
   int *piWidth,
   int *piHeight
   );
```

Include File C_PicTool.h

Description Returns the current dimensions of the output image.

Parameters

Name: piWidth

Description: A pointer to an integer that receives the width of the image.

Name: piHeight

Description: Pointer to an integer that receives the height of the image.

Notes To determine whether the open device supports programmable image dimensions, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGEDIMS.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Holds image width and height.
int Result, iWidth, iHeight;
// Picture tool object.
CcPictureTool PictureTool;
Result = PictureTool.GetImage
   Dimensions(&iWidth, &iHeight );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

GetScaledImageDimensions

Syntax

```
int GetScaledImageDimensions(
   int *piWidth,
   int *piHeight
   );
```

Include File C_PicTool.h

Description Returns the dimensions of the output image after the current horizontal and vertical scale factors have been applied.

Parameters

Name:	piWidth
Description:	A pointer to the integer that receives the scaled width of the image.

Name: piHeight

Description: Pointer to the integer that receives the scaled height of the image.

Notes To determine whether the open device supports programmable image dimensions, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGEDIMS.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Get the scaled width and height
//of the current output image.
if (PictureTool.GetScaledImage
   Dimensions(&iWidth, &iHeight)
   < 0 )
{
//Operation failed - handle error.
}
```


SetImageWidth

Syntax

```
int SetImageWidth(
   int iWidth
   );
```

Include File C_PicTool.h

Description Sets the width of the output image.

Parameters

Name: iWidth

Description: An integer that specifies the image width. Use the method **GetImageLimits** to determine the allowable range for the image width.

Notes To determine whether the open device supports programmable image dimensions, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGEDIMS.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Set the width of the output
//image.
if (PictureTool.SetImageWidth(640)
   < 0)
{
//Operation failed - handle error.
}
```

GetImageWidth

Syntax

```
int GetImageWidth(
   int *piWidth
   );
```

Include File C_PicTool.h

Description Returns the current width of the output image.

Parameters

Name: piWidth

Description: A pointer to an integer that receives the current image width.

Notes To determine whether the open device supports programmable image dimensions, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGEDIMS.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Variable to receive the image
//width.
int iWidth;
//Get the current width of the
//output image.
if (PictureTool.GetImageWidth
   (&iWidth)< 0)
{
//Operation failed - handle error.
}
```

GetScaledImageWidth

Syntax

```
int GetScaledImageWidth(
   int *piWidth
   );
```

Include File C_PicTool.h

Description Returns the width of the image after the current horizontal scale factor has been applied.

Parameters

Name: piWidth

Description: A pointer to an integer that receives the scaled image width.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Variable to receive the scaled
//image width.
int iWidth;
//Get the current scaled width of
//the output image.
if (PictureTool.GetScaledImage
   Width(&iWidth)< 0)
```

Example (cont.)

```
{
//Operation failed - handle error.
}
```

SetImageHeight

Syntax

```
int SetImageHeight(
   int iHeight
   );
```

Include File C_PicTool.h

Description Sets the height of the output image.

Parameters

Name: iHeight

Description: An integer that specifies the image height. Use the method **GetImageLimits** to determine the allowable range for the image height.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Set the height of the output
//image.
```

Example (cont.)

```
if (PictureTool.SetImage
   Height(480)< 0)
{
//Operation failed - handle error.
}
```

GetImageHeight

Syntax

```
int GetImageHeight(
   int *piHeight
   );
```

Include File C_PicTool.h

Description Returns the current height of the output image.

Parameters

Name: piHeight

Description: A pointer to an integer that receives the current image height.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
```

Example (cont.)

```
//Variable to receive the image
//height.
int iHeight;
//Get the current height of the
//output image.
if (PictureTool.GetImageHeight
   (&iHeight)< 0)
{
//Operation failed - handle error.
}
```

GetScaledImageHeight

Syntax

```
int GetScaledImageHeight(
   int *piHeight
   );
```

Include File C_PicTool.h

Description Returns the height of the image after the current vertical scale factor has been applied.

Parameters

Name: piHeight

Description: A pointer to an integer that receives the scaled image height.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Variable to receive the scaled
//image height.
int iHeight;
//Get the current scaled height of
//the output image.
if (PictureTool.GetScaledImage
   Height(&iHeight)< 0)
{
//Operation failed - handle error.
}
```

GetImageLimits

Syntax

```
int GetImageLimits(
   DT_IMAGELIMITS *pImageLimits
   );
```

Include File C_PicTool.h

Description Returns the height, width, and type of the output image.

Parameters

Name: pImageLimits

Description: A pointer to the DT_IMAGELIMITS structure that receives the image height, width, and type.

Notes None

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
//Structure to receive the image
//limits.
DT_IMAGELIMITS ImageLimits;
//Get the width, height, and type
//of the current output image.
if (PictureTool.GetImageLimits
   (&ImageLimits) < 0 )
{
//Operation failed - handle error.
}
```

SetImageType

Syntax

```
int SetImageType(
   int iImageType
   );
```

Include File C_PicTool.h

Description Specifies the type of output image to use.

Parameters

Name: iImageType

Description: An integer that specifies the type of output image. The value can be one of the following:

- `IMAGE_TYPE_08BIT_GS` = An 8-bit grayscale image.
- `IMAGE_TYPE_16BIT_GS` = A 16-bit grayscale image.
- `IMAGE_TYPE_32BIT_GS` = A 32-bit grayscale image.
- `IMAGE_TYPE_FLOAT_GS` = A floating-point grayscale image.
- `IMAGE_TYPE_24BIT_RGB` = A 24-bit RGB color image.
- `IMAGE_TYPE_24BIT_HSL` = A 24-bit HSL color image.
- `IMAGE_TYPE_BINARY` = A binary image.

Notes

To determine whether the open device supports programmable image types, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGETYPE.

Use the method **GetImageLimits** to determine the valid image types before calling **SetImageType**.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Set the output image type to
// 8-bit grayscale.
if (PictureTool.SetImageType
   (IMAGE_TYPE_08BIT_GS) < 0 )
{
//Operation failed - handle error.

}
```

GetImageType

Syntax

```
int GetImageType(
   int *piImageType
   );
```

Include File C_PicTool.h

Description Returns the type of the image object that is returned on the next call to **GetCompatibleImage** or the type of image object that should be used on the next call to **TakePicture**.

Parameters

Name: piImageType

Description: A pointer to an integer that receives the image type. The value can be one of the following:

- `IMAGE_TYPE_08BIT_GS` = An 8-bit grayscale image.
- `IMAGE_TYPE_16BIT_GS` = A 16-bit grayscale image.

Description (cont.):

- `IMAGE_TYPE_32BIT_GS` = A 32-bit grayscale image.
- `IMAGE_TYPE_FLOAT_GS` = A floating-point grayscale image.
- `IMAGE_TYPE_24BIT_RGB` = A 24-bit RGB color image.
- `IMAGE_TYPE_24BIT_HSL` = A 24-bit HSL color image.
- `IMAGE_TYPE_BINARY` = A binary image.

Notes To determine whether the open device supports programmable image types, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGETYPE.

Return Values

< 0	The image type cannot be obtained, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result, ImageType;
// Get the current image type.
Result = PictureTool.GetImageType
   ( &ImageType );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

IsImageTypeSupported

Syntax

```
BOOL IsImageTypeSupported(
   int iImageType
   );
```

Include File C_PicTool.h

Description Determines which image types are supported by the open device.

Parameters

Name: iImageType

Description: An integer that specifies image type. The value can be one of the following:

- `IMAGE_TYPE_08BIT_GS` = An 8-bit grayscale image.
- `IMAGE_TYPE_16BIT_GS` = A 16-bit grayscale image.
- `IMAGE_TYPE_32BIT_GS` = A 32-bit grayscale image.
- `IMAGE_TYPE_FLOAT_GS` = A floating-point grayscale image.
- `IMAGE_TYPE_24BIT_RGB` = A 24-bit RGB color image.
- `IMAGE_TYPE_24BIT_HSL` = A 24-bit HSL color image.
- `IMAGE_TYPE_BINARY` = A binary image.

Notes To determine whether the open device supports programmable image types, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGETYPE.

Return Values

TRUE The specified image type is supported by the open device.

FALSE The specified image type is not supported by the open device.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Determine if the open device
// supports 8-bit grayscale images
if (PictureTool.IsImageType
   Supported(IMAGE_TYPE_08BIT_GS))
{
//8-bit grayscale images are
//supported.
}
```

SetFrameType

Syntax

```
int SetFrameType(
   int iFrameType
   );
```

Include File C_PicTool.h

Description Specifies the video frame type for devices that support this functionality.

Parameters

Name: iFrameType

Description: An integer that specifies the frame type. The value can be one of the following:

- `FRMTYPE_INTSTARTONEVEN` = Acquire an interlaced frame, starting with the next even field.
- `FRMTYPE_INTSTARTONODD` = Acquire an interlaced frame, starting with the next odd field.
- `FRMTYPE_INTSTARTONNEXT` = Acquire an interlaced frame, starting with the next field of either kind (odd or even).
- `FRMTYPE_EVENFIELD` = Acquire even fields from an interlaced frame, starting with the next even field.
- `FRMTYPE_ODDFIELD` = Acquire odd fields from an interlaced frame, starting with the next odd field.
- `FRMTYPE_NEXTFIELD` = Starting with the next field of either kind (odd or even) from an interlaced frame, acquire fields of that kind.
- `FRMTYPE_NONINTERLACED` = Acquire noninterlaced frame.

Notes To determine whether the open device supports programmable frame types, use the **IsDeviceCapSupported** method with the argument DEVCAP_FRAMETYPE.

Notes (cont.) Use the method **GetImageLimits** to determine the valid frame types before calling this method.

Return Values

< 0 The image type cannot be obtained, a device and plug-in are not currently open, or the CcPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Set the frame type to
// noninterlaced.
if (PictureTool.SetFrameType
   (FRMTYPE_NONINTERLACED) < 0 )
{
//Operation failed - handle error.
}
```

GetFrameType

Syntax

```
int GetFrameType(
   int *piFrameType
   );
```

Include File C_PicTool.h

Description Returns the video frame type for devices that support this functionality.

Parameters

Name: piFrameType

Description: A pointer to an integer in which the current frame type is returned. The value can be one of the following:

- `FRMTYPE_INTSTARTONEVEN` = Acquire an interlaced frame, starting with the next even field.
- `FRMTYPE_INTSTARTONODD` = Acquire an interlaced frame, starting with the next odd field.
- `FRMTYPE_INTSTARTONNEXT` = Acquire an interlaced frame, starting with the next field of either kind (odd or even).
- `FRMTYPE_EVENFIELD` = Acquire even fields from an interlaced frame, starting with the next even field.
- `FRMTYPE_ODDFIELD` = Acquire odd fields from an interlaced frame, starting with the next odd field.
- `FRMTYPE_NEXTFIELD` = Starting with the next field of either kind (odd or even) from an interlaced frame, acquire fields of that kind.
- `FRMTYPE_NONINTERLACED` = Acquire noninterlaced frame.

Notes To determine whether the open device supports programmable frame types, use the **IsDeviceCapSupported** method with the argument DEVCAP_FRAMETYPE.

Return Values

< 0	Operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Variable to receive the
// frame type.
int iFrameType;
// Get the current frame type for
// the open device.
if (PictureTool.GetFrameType
   (&iFrameType) < 0 )
{
//Operation failed - handle error.
}
```

IsFrameTypeSupported

Syntax

```
BOOL IsFrameTypeSupported(
   int iFrameType
   );
```

Include File C_PicTool.h

Description Determines which frame types are supported by the open device.

Parameters

Name: iFrameType

Description: An integer that specifies image type. The value can be one of the following:

- `FRMTYPE_INTSTARTONEVEN` = Acquire an interlaced frame, starting with the next even field.
- `FRMTYPE_INTSTARTONODD` = Acquire an interlaced frame, starting with the next odd field.
- `FRMTYPE_INTSTARTONNEXT` = Acquire an interlaced frame, starting with the next field of either kind (odd or even).
- `FRMTYPE_EVENFIELD` = Acquire even fields from an interlaced frame, starting with the next even field.
- `FRMTYPE_ODDFIELD` = Acquire odd fields from an interlaced frame, starting with the next odd field.
- `FRMTYPE_NEXTFIELD` = Starting with the next field of either kind (odd or even) from an interlaced frame, acquire fields of that kind.
- `FRMTYPE_NONINTERLACED` = Acquire noninterlaced frame.

Notes To determine whether the open device supports programmable frame types, use the **IsDeviceCapSupported** method with the argument DEVCAP_FRAMETYPE.

Return Values

TRUE The specified image type is supported by the open device.

FALSE The specified image type is not supported by the open device.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Determine if the open device
// supports noninterlaced frame
// types
if (PictureTool.IsFrameType
   Supported(FRMTYPE_NONINTERLACED
   ))
{
//noninterlaced frame types are
//supported.
}
```

SetImageScale

Syntax

```
int SetImageScale(
   int iHorzScale,
   int iVertScale
   );
```

Include File C_PicTool.h

Description Specifies the current horizontal and vertical scale factors of the output image.

Parameters

Name: iHorzScale

Description: An integer that specifies the horizontal scale factor. Value range from 0 to 100 percent.

Name: iVertScale

Description: An integer that specifies the vertical scale factor. Value range from 0 to 100 percent.

Notes

To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Use the method **GetImageLimits** to determine the valid frame types before calling this method.

Return Values

< 0	The operation failed.
0	Successful.

Example

The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Scale the image width and
// height by 50%.
if (PictureTool.SetImageScale
   (50,50) < 0 )
{
//Operation failed - handle error.
}
```

GetImageScale

Syntax

```
int GetImageScale(
   int *piHorzScale,
   int *piVertScale
   );
```

Include File C_PicTool.h

Description Returns the current horizontal and vertical scale factors of the output image.

Parameters

Name: piHorzScale

Description: A pointer to an integer that specifies the horizontal scale factor. Value range from 0 to 100 percent.

Name: piVertScale

Description: A pointer to an integer that specifies the vertical scale factor. Value range from 0 to 100 percent.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Variable to receive the
// horizontal scale factor.
```

Example

```
int iHorzScale;
// Variable to receive the
// vertical scale factor.
int iVertScale;
// Get the current horizontal and
// vertical scale factors.
if (PictureTool.GetImageScale
   (&iHorzScale, &iVertScale) < 0)
{
//Operation failed - handle error.
}
```

SetImageHorzScale

Syntax

```
int SetImageHorzScale(
   int iHorzScale
   );
```

Include File C_PicTool.h

Description Specifies the current horizontal scale factor of the output image.

Parameters

Name: iHorzScale

Description: An integer that specifies the horizontal scale factor. Value range from 0 to 100 percent.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Use the method **GetImageLimits** to determine the valid frame types before calling this method.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Scale the image width by 50%.
if (PictureTool.SetImageHorzScale
   (50) < 0 )
{
//Operation failed - handle error.
}
```

GetImageHorzScale

Syntax

```
int GetImageHorzScale(
   int *piHorzScale
   );
```

Include File C_PicTool.h

Description Returns the current horizontal scale factor of the output image.

Parameters

Name:	piHorzScale
Description:	A pointer to an integer that specifies the horizontal scale factor. Value range from 0 to 100 percent.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Variable to receive the
// horizontal scale factor.
int iHorzScale;
// Get the horizontal scale factor
if (PictureTool.GetImageHorzScale
   (&iHorzScale) < 0 )
{
//Operation failed - handle error.
}
```

SetImageVertScale

Syntax

```
int SetImageVertScale(
   int iVertScale
   );
```

Include File C_PicTool.h

Description Specifies the current vertical scale factor of the output image.

Parameters

Name:	iVertScale
Description:	An integer that specifies the vertical scale factor. Value range from 0 to 100 percent.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Use the method **GetImageLimits** to determine the valid frame types before calling this method.

Return Values

< 0	The operation failed.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Scale the image height by 50%.
if (PictureTool.SetImageVertScale
   (50) < 0 )
{
//Operation failed - handle error.
}
```

GetImageVertScale

Syntax

```
int GetImageVertScale(
   int *piVertScale
   );
```

Include File C_PicTool.h

Description Returns the current vertical scale factor of the output image.

Parameters

Name: piVertScale

Description: A pointer to an integer that specifies the vertical scale factor. Value range from 0 to 100 percent.

Notes To determine whether the open device supports programmable scale factors, use the **IsDeviceCapSupported** method with the argument DEVCAP_IMAGESCALE.

Return Values

< 0 The operation failed.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// Variable to receive the
// vertical scale factor.
int iVertScale;
// Get the vertical scale factor
if (PictureTool.GetImageVertScale
   (&iVertScale) < 0 )
{
//Operation failed - handle error.
}
```

TimedPictureSaveToAVI

Syntax

```
int TimedPictureSaveToAVI(
   CcAVI pAVI,
   int iFrameCount,
   int iTimeDelay
   );
```

Include File C_PicTool.h

Description Captures and saves one or more images to an .AVI file.

Parameters

Name: pAVI

Description: A pointer to an object of type CcAVI.

Name: iFrameCount

Description: The number of frames that you want to write to the .AVI file. The value must be greater than or equal to 1.

Name: iTimeDelay

Description: The delay time between frames that you want to use when generating the .AVI file, in milliseconds. The value must be greater than or equal to 0.

Notes Before calling this method, you must use **CcAVI::Create** to initialize a CcAVI object and create a new AVI file that is compatible with the current image output format of the CcPictureTool object (appropriate image type and dimensions). For more information, refer to Chapter 4, "Using the AVI Player Tool API."

Return Values

< 0	The format of the AVI file is not compatible with the current image output format of the CcPictureTool object, a device and plug-in are not currently open, *pAVI* is NULL, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// AVI object.
CcAVI AVI;
int iImageType;
int iWidth, iHeight;
// Get output format of picture
// tool.
if ( PictureTool.GetImageType
   ( &iImageType ) < 0 )
return -1;
// Get image dimensions.
if ( PictureTool.GetImage
   Dimensions(&iWidth,&iHeight)<0)
return -1;
// Create an AVI object that is
// compatible with the output
// format of the picture tool.
if ( AVI.Create("C:\\AviFile.avi",
   iImageType, iWidth, iHeight)<0)
// Create an AVI object that is
// compatible with the output
// format of the picture tool.
if ( AVI.Create("C:\\AviFile.avi",
   iImageType, iWidth, iHeight)<0)
return -1;
```

Example (cont.)

```
// Capture 40 frames (with a 100ms
// delay between each frame) and
// save them in the AVI file.
if ( TimedPictureSaveToAVI(&AVI,
   40, 100 ) < 0 )
return -1;
```

TimedPictureSaveToDisc

Syntax

```
int TimedPictureSaveToDisc(
   LPCSTR szDir,
   LPCSTR szBaseFileName,
   int iFrameCount,
   int iCountStart,
   int iTimeDelay
   );
```

Include File C_PicTool.h

Description Captures and saves one or more images to one or more bitmap files.

Parameters

Name: szDir

Description: A null terminated constant string that specifies the directory in which you want to place the bitmap file(s).

Name: szBaseFileName

Description: A null terminated constant string that specifies the base file name to use for all bitmap files.

Name: iFrameCount

Description: The number of frames that you want to write to disk. The value must be greater than or equal to 1.

Name: iCountStart

Description: The first counter number that you want to append to the base file name when you start generating the bitmap files.

Name: iTimeDelay

Description: The delay time between frames that you want to use when generating the bitmap files, in milliseconds. The value must be greater than or equal to 0.

Notes Bitmap files generated by this method have names of the form *BaseFileName*(*n*), where *BaseFileName* is the base file name for all bitmap files and *n* is a number that is appended to the end of the base file name to ensure uniqueness. For example, if *nCountStart* = 4, *nFrameCount* = 4, and *szBaseFileName* = "MyBitmap," the tool captures four images and saves them as MyBitmap4.bmp, MyBitmap5.bmp, MyBitmap6.bmp, and MyBitmap7.bmp.

Return Values

< 0	The device and plug-in are not currently open, one or more of the input argument is NULL or out of range, or the CcPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
// Capture 40 frames with a 100 ms
// delay between each frame.

Result = TimedPictureSaveToDisc(
   "C:\Temp\","MyBitmap", 40, 0,
   100)
if ( Result < 0)
{
   //Handle error
}
```

TimedPictureSaveToMemory

Syntax

```
int TimedPictureSaveToMemory(
   LPCSTR szBaseImageName,
   int iFrameCount,
   int iCountStart,
   int iTimeDelay,
   CcList *pImageList
   );
```

Include File C_PicTool.h

Description Captures and saves one or more images to images in memory.

Parameters

Name: szBaseImageName

Description: A null terminated constant string that specifies the base name for all images generated by this method. The value must not be NULL.

Name: iFrameCount

Description: The number of frames that you want to acquire from the open device. The value must be greater than or equal to one (1). The maximum value is determined by the amount of memory available in your system.

Name: iCountStart

Description: The first counter number that you want to append to the base file name when you start generating the bitmap files. The value must be greater than or equal to zero (0).

Name: iTimeDelay

Description: The delay time between frames that you want to use when generating the bitmap files, in milliseconds. The value must be greater than or equal to 0.

Name: pImageList

Description: A pointer to a CcList object that will receive the image objects generated by this method. Any existing objects in the specified list are deleted. The value cannot be NULL.

Notes Images generated by this method have names of the form *BaseFileName*(*n*), where *BaseFileName* is the base file name for all bitmap files and *n* is a number that is appended to the end of the base file name to ensure uniqueness. For example, if *nCountStart* = 4, *nFrameCount* = 4, and *szBaseFileName* = "MyImage", the tool captures four images and saves them as MyImage4, MyImage5, MyImage6, and MyImage7.

Return Values

< 0	One of the input arguments is out of range, one of the input arguments is equal to NULL, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
CcList ImageList;
int iFrameCount = 10;
int iCountStart = 0;
int iTimeDelay  = 100;
// Acquire 10 frames at 100ms
// intervals. Images will be named
// Image0, Image1, Image2, ...,
// Image9.
Result =
   PictureTool.TimedPictureSaveTo
   Memory("Image", iFrameCount,
   iCountStart, iTimeDelay,
   &ImageList );
```

Example (cont.)

```
if ( Result < 0 )
    {
// Handle error.
    }
```

CancelTimedSave Method

Syntax

```
int CancelTimedSave(
   );
```

Include File C_PicTool.h

Description Cancels a call to the **TimedPictureSaveToAVI**, **TimedPictureSaveToDisc**, or **TimedPictureSaveToMemory** method.

Parameters None

Notes None

Return Values

< 0	A device and plug-in are not currently open or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture Tool object
CcPictureTool PictureTool;
int Result;

//Cancel any Timed Save operation
Result=PictureTool.CancelTimed
   Save();
```

Example (cont.)

```
if (Result < 0)
{
//Operation failed - handle error.
}
```

StartLiveVideo

Syntax

```
int StartLiveVideo(
   HWND hParent
   );
```

Include File C_PicTool.h

Description Starts a simulated passthru (live video) operation in the specified window.

Parameters

Name: hParent

Description: A handle to the window in which to display the live video. The value cannot be NULL.

Notes Live video is a low frame-rate form of passthru in which images are continually acquired through calls to **TakePicture** and displayed in the specified window.

Return Values

< 0	A device and plug-in are not currently open, the input argument is NULL, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
HWND hWindow;
// Fill in hWindow with a valid
// window handle.
hWindow = <some window handle>;
// Start live video.
if ( PictureTool.StartLiveVideo
   ( hWindow ) < 0 )
{
//Operation failed - handle error.
}
```

StopLiveVideo

Syntax

```
int StopLiveVideo(
   );
```

Include File C_PicTool.h

Description Stops a simulated passthru (live video) operation.

Parameters None

Notes None

Return Values

< 0	A device and plug-in are not currently open, live video is not currently running, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// See if live video is currently
// running.
if (PictureTool.IsLiveVideo
   Running() )
{
// Stop live video.
if (PictureTool.StopLiveVideo()
   < 0 )
{
//Operation failed - handle error.
    }
}
```

IsLiveVideoRunning

Syntax
```
BOOL IsLiveVideoRunning(
   );
```

Include File C_PicTool.h

Description Determines whether a simulated passthru (live video) operation is currently running.

Parameters None

Notes It is recommended that you call this method before you call **StartLiveVideo** or **StopLiveVideo**.

Return Values

TRUE Live video is currently running.

FALSE Live video is not currently running.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsRunning;
bIsRunning=PictureTool.IsLiveVideo
   Running();
if ( bIsRunning == TRUE )
{
//Live video is currently running.
}
```

EnablePassthruAcquire

Syntax

```
int EnablePassthruAcquire(
   BOOL bEnable
   );
```

Include File C_PicTool.h

Description Enables passthru acquire mode. This allows you to capture images while passthru is running.

Parameters

Name: bEnable

Description: Specifies whether passthru acquire mode is enabled. If *bEnable* = TRUE, passthru acquire mode is enabled. If *bEnable* = FALSE, passthru acquire mode is disabled.

Notes To determine whether the open device supports passthru acquire mode, use the **IsDeviceCapSupported** method with the argument DEVCAP_PASSTHRUACQUIRE.

For some devices, passthru acquire mode reduces the amount of overhead associated with acquiring a frame and reduces the acquisition time.

Return Values

< 0 A device and plug-in are not currently open, the open device does not support passthru acquire, the input argument is out of range, or the CCPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
BOOL bIsSupported;
// See if the open device supports
// passthru capture.
bIsSupported=PictureTool.IsDevice
   CapSupported(DEVCAP_PASSTHRU
   ACQUIRE);
if ( bIsSupported == TRUE )
{
// Enable passthru acquire mode.
    if
   (PictureTool.EnablePassthru
   Acquire( TRUE ) < 0 )
    {
//Operation failed - handle error.
    }
```

StartPassthru

Syntax

```
int StartPassthru(
   HWND hParent
   );
```

Include File C_PicTool.h

Description Starts a true passthru operation in the specified window.

Parameters

Name: hParent

Description: A handle to the window in which to display the true passthru operation. The value cannot be NULL.

Notes None

Return Values

< 0	A device and plug-in are not currently open, the input argument is NULL, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsSupported;
HWND hWindow;
// Fill in the window handle.
hWindow = <some window handle>;
// See if the open device supports
// passthru.
bIsSupported=PictureTool.IsDevice
   CapSupported (DEVCAP_PASSTHRU);
```

Example (cont.)

```
if ( bIsSupported == TRUE )
{
// Start passthru.
    if ( PictureTool.StartPassthru
   ( hWindow ) < 0 )
    {
//Operation failed - handle error.
    }
}
```

StopPassthru

Syntax

```
int StopPassthru(
   );
```

Include File C_PicTool.h

Description Stops a true passthru operation.

Parameters None

Notes None

Return Values

< 0	A device and plug-in are not currently open, passthru is not currently running, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
BOOL bIsSupported;
// See if the open device supports
// passthru.
bIsSupported=PictureTool.IsDevice
   CapSupported(DEVCAP_PASSTHRU);
```

Example (cont.)

```
if ( bIsSupported == TRUE )
{
// See if live video is currently
// running.
  if
   (PictureTool.IsPassthruRunning(
   ) )
  {
// Stop passthru.
    if (PictureTool.StopPassthru()
   < 0 )
    {
//Operation failed - handle error.
    }
  }
}
```

IsPassthruRunning

Syntax

```
BOOL IsPassthruRunning(
   );
```

Include File C_PicTool.h

Description Determines whether a true passthru operation is currently running.

Parameters None

Notes It is recommended that you call this method before you call **StartPassthru** or **StopPassthru**.

Return Values

TRUE A true passthru operation is currently running.

FALSE A true passthru operation is not currently running.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
// BOOL to receive result.
BOOL bIsRunning;
bIsRunning=PictureTool.IsPassthru
   Running();
if ( bIsRunning == TRUE )
{
// Passthru is currently running.
}
```

GetDeviceSettingsFileExt

Syntax

```
int GetDeviceSettingsFileExt(
   LPSTR szFileExt
   );
```

Include File C_PicTool.h

Description Returns the 3-character extension of the settings file for the currently open device.

Parameters

Name: szFileExt

Description: A string that contains the 3-character extension (and the NULL termination character).

Notes This method is useful if you want to create files with unique file names to store the settings for your various devices. For example, if the open device is a DT3152 frame grabber board, this method always returns `052`. You can append this extension to a base file name (such as Settings) to create a unique file name (Settings.O52) for the DT3152 settings file.

Return Values

< 0	A device and plug-in are not currently open or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
// Picture tool object.
CcPictureTool PictureTool;
int Result;
// String to receive extension.
char szFileExt[10];
Result =
   PictureTool.GetDevice
   Settings FileExt( szFileExt );
if ( Result < 0 )
{
//Operation failed - handle error.
}
```

GetDeviceSettingsFileDesc

Syntax

```
GetDeviceSettingsFileDesc(LPSTR
   szFileDesc
   );
```

Include File C_PicTool.h

Description Returns a brief description of the settings file for the open device.

Parameters

Name: szFileDesc

Description: A string that is large enough to hold at least _MAX_PATH characters, including the NULL terminator.

Notes The returned string is primarily used in Open / Save file dialog boxes. For example, a call to this method might return the characters "DT3152 Settings Files" when the open device is a DT3152 board.

_MAX_PATH is defined in stdlib.h

Return Values

< 0	A device and plug-in are not currently open or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture tool object
ccPictureTool PictureTool;
//String to receive the
//description
```

Example (cont.)

```
char szFileDesc[_MAX_PATH+1];
int Result;

Result = PictureTool.GetDevice
   SettingsFileDesc( szFileDesc);
if Result <0)
{
//Operation failed - handle error
}
```

LoadDeviceSettings

Syntax

```
int LoadDeviceSettings(
   LPSTR szFileName
   );
```

Include File C_PicTool.h

Description Loads the settings for the open device from a file.

Parameters

Name: szFileName

Description: A NULL-terminated string that identifies the settings file to load.

Notes None

Return Values

< 0	A device and plug-in are not currently open, the device settings file could not be loaded, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture Tool object
CcPictureTool PictureTool;
int Result;
Result = PictureTool.LoadDevice
   Settings (c:\MySettings.052);
if (Result < 0)
{
//Operation failed - handle error
}
```

SaveDeviceSettings

Syntax

```
int SaveDeviceSettings(
   LPCSTR szFileName
   );
```

Include File C_PicTool.h

Description Saves the settings for the open device to a file.

Parameters

Name: szFileName

Description: A NULL-terminated string that identifies the file in which the device settings for the open device are saved.

Notes Call **GetDeviceSettingsFileExt** to retrieve the three-character settings file extension for the open device. Then, append this extension to the base name of any settings files that were created for this device.

Using this naming convention, you can easily distinguish settings files for each device.

Return Values

< 0	A device and plug-in are not currently open, the device settings file could not be created, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture tool object.
CcPictureTool  PictureTool;
int Result;

Result = PictureTool.SaveDevice
   Settings( "C:\MySettings.052");
If (Result < 0 )
{
//Operation failed - handle error
}
```

SetDeviceSettings

Syntax

```
int SetDeviceSettings(
   DEVICESETTINGS *pSettings
   );
```

Include File C_PicTool.h

Description Programs the open device with the device settings that are contained in the supplied DEVICESETTINGS structure.

Parameters

Name:	pSettings
Description:	A pointer to a DEVICESETTINGS structure that contains the settings for the open device.

Notes None

Return Values

< 0	A device and plug-in are not currently open, the device settings in the supplied DEVICESETTINGS are not compatible with the open device or the input argument is NULL, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture tool object;
CcPictureTool PictureTool;
DEVICESETTINGS Settings;
int Result;

//Initialize the device settings
//structure from a memory location

Result = PictureTool.SetDevice
   Settings (&Settings);
if (Result < 0)
{
//Operation failed - handle error
}
```

GetDeviceSettings

Syntax

```
int GetDeviceSettings(
   DEVICESETTINGS *pSettings
   );
```

Include File C_PicTool.h

Description Returns the device settings from the open device in the supplied DEVICESETTINGS structure.

Parameters

Name: pSettings

Description: A pointer to a DEVICESETTINGS structure that will receive the settings for the open device.

Notes None

Return Values

< 0 A device and plug-in are not currently open, the input argument is NULL, or the CCPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
//Picture tool object;
CcPictureTool PictureTool;
DEVICESETTINGS Settings;
int Result;

Result = PictureTool.GetDevice
   Settings (&Settings);


if (Result < 0)
{
//Operation failed - handle error
}
```

ShowDeviceSettingsDialog

Syntax

```
int ShowDeviceSettingsDialog(
   HWND hParent
   );
```

Include File C_PicTool.h

Description Shows the settings dialog box for the open device.

Parameters

Name: hParent

Description: A handle to the window that serves as the parent of the settings dialog box.

Notes The settings dialog box allow you to modify the settings for the open device. This dialog box is device-specific; therefore, different device or plug-in combinations may have different settings dialog boxes.

Call **IsDeviceCapSupported** with a flag of DEVCAP_SETTINGSDIALOG before using **ShowDeviceSettingsDialog** to determine whether the open device provides a settings dialog.

Return Values

< 0	A device and plug-in are not currently open, an invalid parent window handle is provided, or the CCPictureTool object has not been initialized.
0	Successful.

Example The following is a sample code fragment:

```
//Picture Tool object
ccPictureTool PictureTool;
HWND hParent;

hParent = <some window handle>

if (PictureTool.IsDeviceCap
   Supported(DEVCAP_SETTINGSDIALOG
   ))
{
  //Show the settings dialog box.
  if
   (PictureTool.ShowDeviceSettings
   Dialog( hParent) < 0)
{
//Operation failed - handle error
}
}
```


GetErrorMessage

Syntax

```
void GetErrorMessage(
   LPSTR szMsgBuff,
   int iBuffSize
   );
```

Include File C_PicTool.h

Description Returns a description of the last error that occurred.

Parameters

Name: szMsgBuff

Description: A character buffer that is capable of holding at least ERROR_TEXT_BUF_SIZE characters, including the NULL terminator.

Name: iBuffSize

Description: The size of the supplied character buffer.

This size must be greater than or equal to the value of ERROR_TEXT_BUF_SIZE.

Notes None

Return Values

< 0 A device and plug-in are not currently open, one or more of the input arguments is out of range, or the CCPictureTool object has not been initialized.

0 Successful.

Example The following is a sample code fragment:

```
//Picture Tool object
CcPictureTool PictureTool;
char szMsgBuff[ERROR_TEXT_BUF_
   SIZE];

//Initialize the Picture Tool
//object
if (PictureTool.Initialize() < 0)
{
   //Get error message.
   GetErrorMessage(szMsgBuff,
    ERROR_TEXT_BUF_SIZE);
}
```

16

Using the Pixel Change Tool API

Overview of the Pixel Change Tool API

The API for the Pixel Change tool has one object only: the CcChange class. This tool sets all pixels inside the given ROI (CcRoiBase GLI/2 object) to the specified value for the given image (CcImage GLI/2 object).

Note: Currently, this tool does not support 24-bit HSL color images.

The CcChange class uses a standard constructor and destructor and the class methods listed in Table 32.

Table 32: CcChange Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcChange(void);
	~CcChange(void);
CcChange Class Methods	int Change (CcImage* CImage,CcRoiBase* CRoi, float fNewValue);
	int ChangeRGB(Cc24BitRGBImage* CImage, CcRoiBase* CRoi,BYTE bRed,BYTE bGreen,BYTE bBlue);
	int ChangeOverlay(CcImage* CImage,CcRoiBase* CRoi, BYTE bNewValue);

CcChange Methods

This section describes each method of the CcChange class in detail.

Change

Syntax

```
int Change(
  CcImage* CImage,
  CcRoiBase* CRoi,
  float fNewValue);
```

Include File C_Change.h

Description Changes all the pixels inside the ROI to the specified value in the given image.

Parameters

Name: CImage

Description: Image derived from the CcImage class.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: fNewValue

Description: All pixels inside the ROI are set to this value.

Return Values

–1 Unsuccessful.

0 Successful.

ChangeRGB

Syntax

```
int ChangeRGB(
  Cc24BitRGBImage* CImage,
  CcRoiBase* CRoi,
  BYTE bRed,
  BYTE bGreen,
  BYTE bBlue);
```

Include File C_Change.h

Description Changes all the pixels inside the ROI to the specified value in the given image.

Parameters

Name: CImage

Description: 24- bit color image.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: bRed

Description: Sets all pixels inside the specified ROI for the red plane of the color image to this value.

Name: bGreen

Description: Sets all pixels inside the specified ROI for the green plane of the color image to this value.

Name: bBlue

Description: Sets all pixels inside the specified ROI for the blue plane of the color image to this value.

Return Values

–1 Unsuccessful.

0 Successful.

ChangeOverlay

Syntax

```
int ChangeOverlay(
 CcImage* CImage,
 CcRoiBase* CRoi,
 BYTE bNewValue);
```

Include File C_Change.h

Description Changes all the pixels inside the ROI to the specified value in the given image's overlay.

Parameters

Name: CImage

Description: Image derived from the CcImage class.

Name: CRoi

Description: ROI area in which to perform the operation.

Name: bNewValue

Description: Sets all pixels inside the specified ROI to one of the following values:

- OVERLAY_CLEAR – Clears the overlay.
- OVERLAY_RED – Sets the overlay to a transparent red.
- OVERLAY_GREEN – Sets the overlay to a transparent green.
- OVERLAY_BLUE – Sets the overlay to a transparent blue.
- OVERLAY_YELLOW – Sets the overlay to a transparent yellow.
- OVERLAY_VIOLET – Sets the overlay to a transparent violet.

Description (cont.):

- OVERLAY_CYAN – Sets the overlay to a transparent cyan.
- OVERLAY_WHITE – Sets the overlay to a transparent white.

Return Values

–1	Unsuccessful.
0	Successful.

Example Program Using the Pixel Change Tool API

This example opens a stored image named image1.bmp from disk as a 32-bit image, changes a rectangular portion of the image to the value 55, and then stores the image to disk with the name output.bmp.

Note: This example is made from code fragments with error checking removed. In an actual program, you should check return values and pointers.

A complete tool demo of how to implement your own change tool API and example change tool is located under C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL. Note that this example does not include point & click scripting. If you want to include point & click scripting in your custom tool, refer to the Visual C++ project for the Change tool, located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImageInt32* C32BitImage;

//32-bit grayscale Image
CcRoiRect* CRectRoi;

//Where operation will take place
CcChangeCChange;
//Object to perform operation
/*End of Dec Section*/
```

```
//Allocate memory for objects
C32BitImage = new CcGrayImageInt32();
CRectRoi = new CcRoiRect();
//Initialize ROI
RECT stROI;
stROI.bottom = 50;
stROI.top = 150;
stROI.left = 50;
stROI.right = 150;
CRectRoi->SetRoiImageCord((VOID*)&stROI);
//Open images from disk (or get image data from
//frame grabber)
 C32BitImage->OpenBMPFile(“image1.bmp”);
//Perform change
CChange.Change(C32BitImage,CRectRoi,55);
//Save output to disk
C32BitImage->SaveBMPFile(“output.bmp”);
//Free memory
delete C32BitImage; delete CRectRoi;
}
```

17

Using the Serial I/O Tool API

Overview of the Serial I/O Tool API

The API for the Serial I/O tool has one object only: the CcSerialIO class. This tool reads data from and writes data to the COM ports. You can use all COM ports at the same time and read from or write to these ports in synchronous or asynchronous mode.

The CcSerialIO class uses a standard constructor and destructor and the class methods listed in Table 33.

Table 33: CcSerialIO Object Methods

Method Type	Method Name
CcSerialIO Constructor and Destructor	CcSerialIO();
	~CcSerialIO();
CcSerialIO Class Methods	BOOL IsComPortAvailable(int iComPort);
	int SetComPortNumber(int iComPort);
	int GetComPortNumber();
	int InitializeComPort(BOOL bAsync = FALSE, BOOL bPurge = FALSE);
	int FreeComPort(void);
	int WriteComPort(char* cPrefix,char* cText,char* cSuffix);
	int WriteComPort(char* cPrefix,CcString* CString, char* cSuffix);
	int WriteComPort(char* cPrefix,CcNumber* CNumber, char* cSuffix);
	char* ReadComPort(char* cPrefix,char* cSuffix);
	int ReadComPort(char* cPrefix,CcString* CString, char* cSuffix);
	int ReadComPort(char* cPrefix,CcNumber* CNumber, char* cSuffix);

Table 33: CcSerialIO Object Methods (cont.)

Method Type	Method Name
CcSerialIO Class Methods (cont).	int SetTimeOut(int iTimeOutRead,int iTimeOutWrite);
	int GetTimeOut(int* iTimeOutRead,int* iTimeOutWrite);
	int SetNumberFormat(int iBefore,int iAfter);
	int GetNumberFormat(int* iBefore,int* iAfter);
	int SetComOptions(HWND hWnd);
	int GetAllComOptions(STALLCOMOPT* stAllOptions);
	int SetAllComOptions(STALLCOMOPT* stAllOptions);
	int Save(char* cFileName);
	int Restore(char* cFileName);
	BOOL IsAsync(void);

CcSerialIO Methods

This section describes each method of the CcSerialIO class in detail.

FreeComPort

Syntax

```
int FreeComPort(void);
```

Include File C_Serial.h

Description Frees up the COM port for use by other applications.

Notes While you are using a COM port, no other applications in the system can use the COM port. After using it, call this method to free the COM port for use by other applications.

Return Values

–1	Unsuccessful.
The value of the active COM port.	Successful.

GetAllComOptions

Syntax

```
int GetAllComOptions(
   STALLCOMOPT* stAllOptions);
```

Include File C_Serial.h

Description Retrieves all options for the active COM port.

Parameters

Name:	stAllOptions
Description:	Pointer to a STALLCOMOPT structure that defines the COM port options.

Notes The STALLCOMOPT structure is defined as follows:

```
struct STComAllOptions {
int iBefore,iAfter;
COMMCONFIG stComConfig;
COMMTIMEOUTS stComTimeouts;
};
typedef struct STComAllOptions
   STALLCOMOPT;
```

The parameters are described as follows:

- *iBefore* – Is the number of decimal places before the decimal point for number formatting.
- *iAfter* – Is the number of decimal places after the decimal point for number formatting.
- *stComConfig* – Is a standard Windows COMMCONFIG structure. For more information, refer to the Windows SDK documentation.
- *stComTimeouts* – A standard Windows COMMTIMEOUTS structure. For more information, refer to the Windows SDK documentation.

This method rarely needs to be used and is for advanced users only. The easiest way to use this method is to first call **GetAllComOptions( )** to fill in the structure. Then, change only what is needed before calling **SetAllComOptions( )** to make the needed changes to the COM port options.

Return Values

–1	Unsuccessful.
0	Successful.

GetComPortNumber

Syntax

```
int GetComPortNumber(void);
```

Include File C_Serial.h

Description Returns the number of the COM port being used.

Notes The serial I/O class works with one COM port at a time. You can use this method to determine which COM port is the active COM port.

Return Values

–1	Unsuccessful.
The value of the active COM port.	Successful.

GetNumberFormat

Syntax

```
int GetNumberFormat(
  int* iBefore,
  int* iAfter);
```

Include File C_Serial.h

Description Retrieves the number format (number of integers before and after the decimal point) for the active COM port.

Parameters

Name: iBefore

Description: The number of integers before the decimal point.

Name: iAfter

Description: The number of integers after the decimal point.

Notes When a Number object is read from a COM port, the object is formatted for the desired decimal places before and after the decimal point. Use this method to retrieve this formatting.

Return Values

–1 Unsuccessful.

0 Successful.

GetTimeOut

Syntax

```
int GetTimeOut(
  int* iTimeOutRead,
  int* iTimeOutWrite);
```

Include File C_Serial.h

Description Retrieves the timeouts for read and write operations for the active COM port.

Parameters

Name: iTimeOutRead

Description: Timeout for read operations, in milliseconds. To disable the timeout, enter 0.

Name: iTimeOutWrite

Description: Timeout for write operations, in milliseconds. To disable the timeout, enter 0.

Notes Each COM port has separate timeouts for reading and writing.

Return Values

–1 Unsuccessful.

0 Successful.

InitializeComPort

Syntax

```
int InitializeComPort(
  BOOL bAsync = FALSE,
  BOOL bPurge = FALSE);
```

Include File C_Serial.h

Description Initializes the active COM port.

Parameters

Name: bAsync

Description: Set this parameter to TRUE if you want to perform asynchronous reads / writes on the COM port. Set this parameter to FALSE to perform synchronous reads / writes on the COM port.

Name: bPurge

Description: Set this parameter to TRUE to clear / purge the associated buffer for the COM port for both the read and write operations.

Notes You can use each COM port for asynchronous or synchronous communication. Each COM port also has a 1K write buffer and a 1K read buffer associated with it. You can use the *bPurge* parameter to clear these buffers when initializing a COM port. The COM port must be initialized before calling any read / write operation.

Return Values

–1	Unsuccessful.
The value of the active COM port.	Successful.

IsAsync

Syntax `BOOL IsAsync(void);`

Include File C_Serial.h

Description Queries the Serial I/O tool to determine if data is being transferred asynchronously for the active COM port.

Notes You can use this method to query the active COM port to determine whether it is set up for asynchronous or synchronous communication.

Return Values

False	COM port is using asynchronous communication.
True	COM port is using asynchronous communication.

IsComPortAvailable

Syntax

```
BOOL IsComPortAvailable(
   int iComPort);
```

Include File C_Serial.h

Description Queries the computer to determine whether the COM port is available.

Parameters

Name: iComPort

Description: Number of the desired COM port.

Notes Use this method before using a COM port to determine whether it is available for use, and that it is working properly on the system.

Return Values

False The COM port is not available.

True The COM port is available.

ReadComPort

Syntax

```
char* ReadComPort(
  char* cPrefix,
  char* cSuffix);
```

or

```
int ReadComPort(
  char* cPrefix,
  CcString* CString,
  char* cSuffix);
```

or

```
int ReadComPort(
  char* cPrefix,
  CcNumber* CNumber,
  char* cSuffix);
```

Include File C_Serial.h

Description Reads the active COM port that is returning the data and discards the given prefix and suffix.

Parameters

Name: cPrefix

Description: Prefix of the data to wait for before reading the data that is entering the COM port.

Name: CString

Description: GLI/2 String object that is receiving the data that is entering the COM port.

Name: CNumber

Description: GLI/2 Number object that is receiving the data that is entering the COM port.

Name:	cSuffix
Description:	Suffix of the data to wait for before reading the data that is entering the COM port.
Notes	This method has three forms. Not all parameters may be required. You can use the contents of a GLI/2 String or Number object to receive the data, or have the method return a simple string pointer. If you use a Number object, the class automatically formats the Number objects to the desired decimal places according to the options set up for this COM port. In all cases, only the data for the read transmission is returned. The prefix and suffix information is discarded.

Return Values

–1	Unsuccessful.
0	Successful.
NULL	Unsuccessful.
A pointer to string (char*) containing read.	Successful.

Restore

Syntax	`int Restore(char* cFileName);`
Include File	C_Serial.h
Description	Restores all Serial I/O tool settings from disk.

Parameters

Name: cFileName

Description: Full path name of the file used to restore the serial I/O options.

Notes Use these setting to restore all COM port settings from disk.

Return Values

–1 Unsuccessful.

0 Successful.

Save

Syntax `int Save(char* cFileName);`

Include File C_Serial.h

Description Saves all Serial I/O tool settings to disk.

Parameters

Name: cFileName

Description: Full path name of the file that is used to save serial I/O options.

Notes Use these setting to save all COM port setting to disk.

Return Values

–1 Unsuccessful.

0 Successful.

SetAllComOptions

Syntax

```
int SetAllComOptions(
   STALLCOMOPT* stAllOptions);
```

Include File C_Serial.h

Description Sets all options for the active COM port.

Parameters

Name: stAllOptions

Description: Pointer to a STALLCOMOPT structure that is used to define the COM port options.

Notes The STALLCOMOPT structure is defined as follows:

```
struct STComAllOptions {
int iBefore,iAfter;
COMMCONFIG stComConfig;
COMMTIMEOUTS stComTimeouts;
};
typedef struct STComAllOptions
   STALLCOMOPT;
```

The parameters are defined as follows:

- *iBefore* – Is the number of decimal places before the decimal point for number formatting.
- *iAfter* – Is the number of decimal places after the decimal point for number formatting.
- *stComConfig* – Is a standard Windows COMMCONFIG structure. For more information, refer to the Windows SDK documentation.

Notes (cont.)

- *stComTimeouts* – Is a standard Windows COMMTIMEOUTS structure. For more information, refer to the Windows SDK documentation.

This method rarely needs to be used and is for advanced users only. The easiest way to use this method is to first call **GetAllComOptions( )** to fill in the structure. Then, change only what is needed before calling **SetAllComOptions( )** to make the needed changes to the COM port options.

Return Values

–1	Unsuccessful.
0	Successful.

SetComOptions

Syntax `int SetComOptions(HWND hWnd);`

Include File C_Serial.h

Description Sets the COM port options using the operating system-supplied dialog box.

Parameters

Name:	hWnd
Description:	Handle of the window to become the parent window for the system dialog box.

Notes The Windows operating system supplies a common dialog box for setting COM port settings such as baud rate, parity, stop bits, and so on. You can set these COM port options for the active COM port using this method. The dialog box should have a parent window to attach to. You supply the handle to this window (a window in your application / tool) in the *hWnd* parameter.

Return Values

–1 Unsuccessful.

0 Successful.

SetComPortNumber

Syntax

```
int SetComPortNumber(
   int iComPort);
```

Include File C_Serial.h

Description Sets the COM port number (1 to 15).

Parameters

Name: iComPort

Description: The number of the COM port to become the active COM port for the class.

Notes The serial I / O class works with one COM port at a time. Use this method to activate the desired COM port. Other methods then operate on this COM port.

Return Values

–1	Unsuccessful.
0	Successful.

SetNumberFormat

Syntax

```
int SetNumberFormat(
  int iBefore,
  int iAfter);
```

Include File C_Serial.h

Description Sets the number format (number of integers before and after the decimal point) for the active COM port.

Parameters

Name: iBefore

Description: The number of integers before the decimal point.

Name: iAfter

Description: The number of integers after the decimal point.

Notes When a Number object is written to a COM port, the object is formatted for the desired decimal places before and after the decimal point. Use this method to set this formatting.

Return Values

–1	Unsuccessful.
0	Successful.

SetTimeOut

Syntax

```
int SetTimeOut(
  int iTimeOutRead,
  int iTimeOutWrite);
```

Include File C_Serial.h

Description Sets the read and write timeouts for the active COM port.

Parameters

Name: iTimeOutRead

Description: Timeout for read operations, in milliseconds. To disable the timeout, enter 0.

Name: iTimeOutWrite

Description: Timeout for write operations, in milliseconds. To disable the timeout, enter 0.

Notes Each COM port has separate timeouts for reading and writing.

Return Values

–1	Unsuccessful.
0	Successful.

WriteComPort

Syntax

```
int WriteComPort(
  char* cPrefix,
  char* cText,
  char* cSuffix);
```

or

```
int WriteComPort(
  char* cPrefix,
  CcString* CString,
  char* cSuffix);
```

or

```
int WriteComPort(
  char* cPrefix,
  CcNumber* CNumber,
  char* cSuffix);
```

Include File C_Serial.h

Description Writes the prefix, data, and suffix out the active COM port.

Parameters

Name: cPrefix

Description: Pointer to a string that contains the prefix to send out the COM port.

Name: cText

Description: Pointer to a string that contains the data to send out the COM port.

Name: CString

Description: Pointer to a GLI/2 String object that contains the data to send out the COM port.

Name: CNumber

Description: Pointer to a GLI/2 Number object that contains the data to send out the COM port.

Name: cSuffix

Description: Pointer to a string that contains the suffix to send out the COM port.

Notes This method has three forms. You can use the contents of a GLI/2 String or Number object as the data along with normal strings. The class converts these objects to text and sends it out the active COM port. It also automatically formats the Number objects to the desired decimal places according to the options set up for this COM port.

Return Values

−1 Unsuccessful.

0 Successful.

Example Program Using the Serial I/O Tool API

This program demonstrates the use of the serial I/O API.

Note: This example is made from code fragments from the Serial I/O tool with error checking removed. For an actual program, you should check return values and pointers.

```
int SomeFunction(void)
{
   CcSerialIO CCom;

   //Set class to use COM1
   CCom.SetComPortNumber (1);

   //See if COM port is ok to use
   if( CCom.IsComPortAvailable() != TRUE)
return(-1);

   //Initialize COM Port
   CCom.InitializeComPort();

   //Write Text out Port
   CCom.WriteComPort (“Prefix”, “Data”, “Suffix”);

   //Return OK
   return(0);
}
```

18

Using the Sound Tool API

Overview of the Sound Tool API

The API for the Sound tool uses GLI/2 API objects. CcWAV uses a standard constructor and destructor and the class methods listed in Table 34.

Table 34: CcWAV Object Methods

Method Type	Method Name
CcWAV Class Methods	void SetWAVFile(char *pszWAVFile);
	void SetSyncMode(int iMode);
	int GetSyncMode( );
	int PlayWAVFile(int iLoopMode);
	int PlayWAVFile(char *pszWAVFile, int iLoopMode);
	void CancelWAVPlay( );

CcWAV Methods

This section describes each method of the CcWAV class in detail.

CancelWAVPlay

Syntax `void CancelWAVPlay(void);`

Include File C_Wav.h

Description Terminates any .WAV file playback that is in progress.

Notes This method stops the playback of any .WAV audio file, including looped and asynchronous playback. Since it is not an error to cancel playback when there is no sound, this method does not return a status value.

Return Values

–1	Unsuccessful.
0	Successful.

GetSyncMode

Syntax `int GetSyncMode (void);`

Include File C_Wav.h

Description Returns the current synchronous / asynchronous playback mode value.

Notes A CcWAV object defaults to synchronous playback.

Return Values

1 Playback is synchronous; a call to **PlayWAVFile()** waits until completion before returning.

0 Playback is asynchronous; a call to **PlayWAVFile()** returns immediately after starting.

PlayWAVFile

Syntax `int PlayWAVFile(int iLoopMode);`

Include File C_Wav.h

Description Plays a .WAV file in single-play or looped-play mode.

Parameters

Name: iLoopMode

Description: Specifies the play mode. If *iLoopMode* is 0, the .WAV file is played once. Otherwise, the .WAV file is played continuously.

Notes You must call **SetWAVFile( )** to specify the .WAV file to play. You can stop looped-play mode either by calling **PlayWAVFile( )** with a new .WAV audio file or by calling **CancelWAVPlay(**).

Return Values

–1 Unsuccessful.

0 Successful.

PlayWAVFile

Syntax

```
int PlayWAVFile(
  char * pszWAVFile,
  int iLoopMode);
```

Include File C_Wav.h

Description Plays a .WAV file in single-play or looped-play mode.

Parameters

Name: pszWAVFile

Description: Full path name of the .WAV audio file.

Name: iLoopMode

Description: Specifies the play mode. If *iLoopMode* is 0, the .WAV file is played once. Otherwise, the .WAV file is played continuously.

Notes The path name that is specified as the first parameter overrides any previous path name that was specified by a call to **SetWAVFile( )**. You can stop looped-play mode either by calling **PlayWAVFile( )** with a new .WAV audio file or by calling **CancelWAVPlay( )**.

Return Values

–1	Unsuccessful.
0	Successful.

SetSyncMode

Syntax

```
void SetSyncMode (int iMode);
```

Include File C_Wav.h

Description Sets either synchronous or asynchronous playback mode.

Parameters

Name: iMode

Description: Specifies the playback mode. If *iMode* is 0, playback is asynchronous; a call to **PlayWAVFile( )** returns immediately after starting. If *iMode* is 1, playback is synchronous; a call to **PlayWAVFile( )** waits until completion before returning.

Notes A CcWAV object defaults to synchronous playback.

Return Values

–1 Unsuccessful.

0 Successful.

SetWAVFile

Syntax `void SetWAVFile(char *pszWAVFile);`

Include File C_Wav.h

Description Specifies the .WAV file to play.

Parameters

Name: pszWAVFile

Description: Full path name of the .WAV audio file.

Notes This method must be called prior to using a **PlayWAVFile( )** method that does not take a .WAV file path name as its first parameter.

Return Values

–1	Unsuccessful.
0	Successful.

Example Program Using the Sound Tool API

This example code creates a CcWAV object. The Sound tool object is used to play a specified .WAV audio file.

Note: This example is made from code fragments from the Sound tool with error checking removed. In an actual program, you should check return values and pointers.

```
int SomeFunction(char *pszWAVPathname)
{
   CcWAV* CWAVPlayer;
   int iReturn;

//Create a new Sound tool object.
//It initializes in synchronous play mode.
//Call SetSyncMode() to change it to asynchronous
//play mode, if desired.

   CWAVPlayer = new CcWAV();

//Play the sound file once. To play in a continuous
//loop, call the method with iLoopMode = 1.

   iReturn = CWAVPlayer->PlayWAVFile(
      pszWAVPathname, 0);

   //Free the memory

   delete CWAVPlayer;
   //Return status
   return(iReturn);
}
```

19

Using the Text Tool API

Overview of the Text Tool API

The API for the Text tool has one object only: the CcTextRoiRect class. This tool places text in an image or its overlay. It is derived from a rectangle ROI class.

The CcTextRoiRect class uses the class methods listed in Table 35.

Table 35: CcTextRoiRect Object Methods

Method Type	Method Name
Constructor and Destructor Methods	CcTextRoiRect( );
	~ CcTextRoiRect( );
CcTextRoiRect Class Methods	int RestoreOrigImageData(CcImage* CImage);
	int CopyTextToImage(HWND hChildWindow, CcImage* CImage);
	int SetPosition(POINT* stPosition);
	int GetPosition(POINT* stPosition);
	int ClearAllLinesOfText(void);
	int AddLineOfText(char* cLineOfText);
	char* GetLineOfText(int iLineNumber);
	int GetNumberOfLinesOfText(void);
	int SetDrawTo(int iDrawToFlag);
	int GetDrawTo(void);
	int SetColors(float fForeground,float fBackground);
	int GetColors(float* fForeground,float* fBackground);
	int SelectFont(LOGFONT* pLogFont);

CcTextRoiRect Methods

This section describes each method of the CcTextRoiRect class in detail.

RestoreOrigImageData

Syntax

```
int RestoreOrigImageData(
   CcImage* CImage);
```

Include File C_Text.h

Description Restores the given image to its original state, clearing the text placed earlier on the image by this object.

Parameters

Name: CImage

Description: Image to be restored.

Notes When a Text object writes text to an image or its overlay, the object copies that portion of the image so that it can be restored. If you need to restore an image to its original state, call this method. You place text on an image by calling **CopyTextToImage( )**.

Because a Text object is derived from a rectangular ROI object, you do not need to call any methods when moving a Text object around on an image with the mouse. The object does this for you, by default.

Return Values

–1 Unsuccessful.

0 Successful.

CopyTextToImage

Syntax

```
int CopyTextToImage(
  HWND hChildWindow,
  CcImage* CImage);
```

Include File C_Text.h

Description Copies the text owned by the Text object to the given image.

Parameters

Name: hChildWindow

Description: Handle to the window in which the given image is displayed.

Name: CImage

Description: Image in which you want to place the text.

Notes Calling this method places the text owned by the Text object into the image or its overlay. The text owned by the Text object is initialized or set by calling **AddLineOfText( )**. You can remove the text from the image by calling **RestoreOrigImageData( )**.

Because a Text object is derived from a rectangular ROI object, you need not call any methods when moving a Text object around on an image with the mouse. The object does this for you, by default.

Return Values

–1 Unsuccessful.

0 Successful.

SetPosition

Syntax

```
int SetPosition(
   POINT* stPosition);
```

Include File C_Text.h

Description Sets the location of the Text object on the image with respect to pixel coordinates.

Parameters

Name: stPosition

Description: Pointer to a Windows POINT structure.

Notes The POINT structure (*stPosition*) describes the placement of the Text object on the image. The point designates the lower-left corner of the Text object. The x,y coordinates must be given in pixel coordinates.

The Windows POINT structure is defined as follows:

```
{
LONG x;
LONG y;
};
```

x specifies the x-coordinate of the point;

y specifies the y-coordinate of the point.

For more detail on this structure, see the Microsoft Win32 SDK.

Because a Text object is derived from a rectangular ROI object, you need not call any methods when moving a Text object around on an image with the mouse. The object does this for you, by default.

Return Values

–1	Unsuccessful.
0	Successful.

GetPosition

Syntax

```
int GetPosition(
   POINT* stPosition);
```

Include File C_Text.h

Description Returns the location of the Text object on the image with respect to pixel coordinates.

Parameters

Name: stPosition

Description: A pointer to a Windows POINT structure.

Notes The POINT structure (*stPosition*) describes the placement of the Text object on the image. The point designates the lower-left corner of the Text object. The x,y coordinates must be given in pixel coordinates.

The Windows POINT structure is defined as follows:

```
{
LONG x;
LONG y;
};
```

x specifies the x-coordinate of the point;

y specifies the y-coordinate of the point.

Notes (cont.) For more detail on this structure, see the Microsoft Win32 SDK.

Because a Text object is derived from a rectangular ROI object, you need not call any methods when moving a Text object around on an image with the mouse. The object does this for you, by default.

Return Values

–1 Unsuccessful.

0 Successful.

ClearAllLinesOfText

Syntax `int ClearAllLinesOfText(void);`

Include File C_Text.h

Description Clears all lines of text owned by the Text object.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can clear all of the text owned by the Text object by calling this method.

A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

–1 Unsuccessful.

0 Successful.

AddLineOfText

Syntax int AddLineOfText(char* cLineOfText);

Include File C_Text.h

Description Adds the given line of text to the Text object.

Parameters

Name: cLineOfText

Description: Line of text to add to the Text object.

Notes The Text object keeps the text you add to it by calling this method internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. It adds the lines of text sequentially starting with line 0.

A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

–1 Unsuccessful.

0 Successful.

GetLineOfText

Syntax

```
char* GetLineOfText(
   int iLineNumber);
```

Include File C_Text.h

Description Returns the desired line of text owned by the Text object.

Parameters

Name:	iLineNumber
Description:	Line of text that you want to retrieve. The first line of text in the Text object is 0.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can retrieve a specific line of text by calling this method.

A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

NULL	Unsuccessful.
The desired line of text.	Successful.

GetNumberOfLinesOfText

Syntax `int GetNumberOfLinesOfText(void);`

Include File C_Text.h

Description Returns the current number of lines of text that are used by the Text object.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can retrieve the number of lines of text currently being used by the Text object by calling this method.

Notes (cont.) A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

–1	Unsuccessful.
The number of lines of text being used by the Text object.	Successful.

SetDrawTo

Syntax
```
int SetDrawTo(int iDrawToFlag);
```

Include File C_Text.h

Description Sets the drawing mode of the object.

Parameters

Name: iDrawToFlag

Description: Flag to set the drawing mode of the object, which can be one of the following values:

- SET_ACCESS_TO_IMAGE_DATA – Places text in the image.
- SET_ACCESS_TO_OVERLAY_DATA – Places text in the image's overlay.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can specify where the text is placed in the image by calling this method. Text can be placed either directly in the image or in the image's transparent overlay.

Notes (cont.) This is not the x,y location where the text is placed. For the location use **SetPosition( )**.

A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

–1	Unsuccessful.
0	Successful.

GetDrawTo

Syntax `int GetDrawTo(void);`

Include File C_Text.h

Description Returns the drawing mode of the object.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can determine where the text is placed in the image by calling this method. Text can be placed either directly in the image or in the image's transparent overlay.

This is not the x,y location where the text is placed. For the location use **GetPosition( )**.

A Text object can hold up to 10 lines of code (lines 0 to 9). Each line can be up to 100 characters.

Return Values

–1	Unsuccessful.
SET_ACCESS_TO_IMAGE_DATA	Places text in image.
SET_ACCESS_TO_OVERLAY_DATA	Places text in image's overlay.

SetColors

Syntax

```
int SetColors(
  float fForeground,
  float fBackground);
```

Include File C_Text.h

Description Sets the foreground (text color) and background color in which the text is displayed.

Parameters

Name:	fForeground
Description:	The color of the text.
Name:	fBackground
Description:	The color of the background behind the text.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can specify in what color the text is shown by calling this method. It differs if you are showing the text in the image or in its overlay.

Notes (cont.) Use one of the following foreground background values:

- OVERLAY_RED – Transparent red.
- OVERLAY_GREEN – Transparent green.
- OVERLAY_BLUE – Transparent blue.
- OVERLAY_WHITE – Transparent white.
- OVERLAY_YELLOW – Transparent yellow.
- OVERLAY_VIOLET – Transparent violet.
- OVERLAY_CYAN – Transparent cyan.
- OVERLAY_CLEAR – Clear, nothing is shown.
- BLACK_TEXT – Solid black.
- BLACK_SEMI_TEXT – Solid semi-black.
- GRAY_DARK_TEXT – Solid dark gray.
- GRAY_TEXT – Solid gray.
- GRAY_LIGHT_TEXT – Solid light gray.
- WHITE_SEMI_TEXT – Solid semi-white.
- WHITE_TEXT – Solid white.
- CLEAR_TEXT – Clear, nothing is shown.

You can also use custom values if you desire.

Return Values

–1	Unsuccessful.
0	Successful.

GetColors

Syntax

```
int GetColors(
  float* fForeground,
  float* fBackground);
```

Include File C_Text.h

Description Returns the foreground (text color) and background color in which the text is displayed.

Parameters

Name: fForeground

Description: The color of the text.

Name: fBackground

Description: The color of the background behind the text.

Notes The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can determine what color the text is shown in by calling this method. It differs if you are showing the text in the image or in its overlay. It returns one of the following values:

- OVERLAY_RED – Transparent red.
- OVERLAY_GREEN – Transparent green.
- OVERLAY_BLUE – Transparent blue.
- OVERLAY_WHITE – Transparent white.
- OVERLAY_YELLOW – Transparent yellow.
- OVERLAY_VIOLET – Transparent violet.

Notes (cont.)

- OVERLAY_CYAN – Transparent cyan.
- OVERLAY_CLEAR – Clear, nothing is shown.
- BLACK_TEXT – Solid black.
- BLACK_SEMI_TEXT – Solid semi-black.
- GRAY_DARK_TEXT – Solid dark gray.
- GRAY_TEXT – Solid gray.
- GRAY_LIGHT_TEXT – Solid light gray.
- WHITE_SEMI_TEXT – Solid semi-white.
- WHITE_TEXT – Solid white.
- CLEAR_TEXT – Clear, nothing is shown.

This method returns a custom value if you entered a custom value using **SetColors( )**.

Return Values

–1	Unsuccessful.
0	Successful.

SelectFont

Syntax `int SelectFont(LOGFONT* pLogFont);`

Include File C_Text.h

Description Sets the font in which the text is displayed.

Parameters

Name:	pLogFont
Description:	A pointer to a LOGFONT structure that describes the desired font.

Notes

The Text object keeps the text you add to it by calling **AddLineOfText( )** internally. It then places this text in an image or its overlay by calling **CopyTextToImage( )**. You can specify in what font the text is shown by calling this method. If you leave the *pLogFont* parameter NULL, a font selection dialog box is displayed, which allows you to choose the desired font. The font you enter here is global to all Text objects in the system.

The LOGFONT is a Windows SDK structure. For more information on it, see the Win32 SDK.

Return Values

–1	Unsuccessful.
0	Successful.

20

Using the Threshold Tool API

Overview of the Threshold Tool API

The API for the Threshold tool has one object only: the CcThreshold class. This tool thresholds an input image (derived from class CcImage) into a binary output image.

The CcThreshold class uses the class methods listed in Table 36.

Table 36: CcThreshold Object Methods

Method Type	Method Name
Constructor and Destructor Methods	CcThreshold(void);
	~ CcThreshold(void);
CcThreshold Class Methods	int Threshold(CcImage* CImageIn, CcBinaryImage* CImageOut,float fMin,float fMax);
	int ThresholdRGB(Cc24BitRGBImage* CImageIn, CcBinaryImage* CImageOut,int iRedMin,int iRedMax, int iGreenMin,int iGreenMax,int iBlueMin,int iBlueMax);
	int ThresholdHSL(Cc24BitRGBImage* CImageIn, CcBinaryImage* CImageOut,int iHueMin,int iHueMax, int iSatMin,int iSatMax,int iLumMin,int iLumMax);
	int ThresholdMulti(CcImage* CImageIn, CcBinaryImage* CImageOut, STTHRESHOLD* stThreshold, int iNumberOfRegions);
	int InvertOutput(BOOL bInvert);

CcThreshold Methods

This section describes each method of the CcThreshold class in detail.

Threshold

Syntax

```
int Threshold(
  CcImage* CImageIn,
  CcBinaryImage* CImageOut,
  float fMin,
  float fMax);
```

Include File C_Thresh.h

Description Thresholds the given input image into a binary output image.

Parameters

Name: CImageIn

Description: Image that was derived from the CcImage class.

Name: CImageOut

Description: Binary image that was derived from the CcImage class.

Name: fMin

Description: Low threshold limit.

Name: fMax

Description: High threshold limit.

Notes This method thresholds the entire image; it does not use an ROI. The input image must either be a binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, or RGB color image. The input and output images must be the same size.

Return Values

–1 Unsuccessful.

0 Successful.

ThresholdRGB

Syntax

```
int ThresholdRGB(
  Cc24BitRGBImage* CImageIn,
  CcBinaryImage* CImageOut,
  int iRedMin,
  int iRedMax,
  int iGreenMin,
  int iGreenMax,
  int iBlueMin,
  int iBlueMax);
```

Include File C_Thresh.h

Description Thresholds the given color input image into a binary output image with respect to all three color planes of an RGB image.

Parameters

Name: CImageIn

Description: RGB color image that was derived from the CcImage class.

Name: CImageOut

Description: Binary image that was derived from the CcImage class.

Name: iRedMin

Description: Low threshold limit for the red color plane of the color image.

Name: iRedMax

Description: High threshold limit for the red color plane of the color image.

Name: iGreenMin

Description: Low threshold limit for the green color plane of the color image.

Name: iGreenMax

Description: High threshold limit for the green color plane of the color image.

Name: iBlueMin

Description: Low threshold limit for the blue color plane of the color image.

Name: iBlueMax

Description: High threshold limit for the blue color plane of the color image.

Notes This method thresholds the entire image; it does not use an ROI. The input image must be a 24-bit RGB color image. The input and output images must be the same size. The minimum and maximum threshold limits for each color plane are AND'ed together so that you can threshold on a very specific color. Thus, the foreground values in the output image are the locations where any color pixel in the input image was within the threshold limit for all three color planes.

Return Values

–1 Unsuccessful.

0 Successful.

ThresholdHSL

Syntax

```
int ThresholdRGB(
  Cc24BitRGBImage* CImageIn,
  CcBinaryImage* CImageOut,
  int iHueMin,
  int iHueMax,
  int iSatMin,
  int iSatMax,
  int iLumMin,
  int iLumMax);
```

Include File C_Thresh.h

Description Thresholds the given color input image into a binary output image with respect to all three color planes of an HSL image.

Parameters

Name: CImageIn

Description: HSL color image that was derived from the CcImage class.

Name: CImageOut

Description: Binary image that was derived from the CcImage class.

Name: iHueMin

Description: Low threshold limit for the hue color plane of the color image.

Name: iHueMax

Description: High threshold limit for the hue color plane of the color image.

Name: iSatMin

Description: Low threshold limit for the saturation color plane of the color image.

Name: iSatMax

Description: High threshold limit for the saturation color plane of the color image.

Name: iLumMin

Description: Low threshold limit for the luminance color plane of the color image.

Name: iLumMax

Description: High threshold limit for the luminance color plane of the color image.

Notes This method thresholds the entire image; it does not use an ROI. The input image must be a 24-bit HSL color image. The input and output images must be the same size. The minimum and maximum threshold limits for each color plane are AND'ed together so that you can threshold on a very specific color plane. Thus, the foreground values in the output image are the locations where any color pixel in the input image was within the threshold limit for all three color planes.

Return Values

–1 Unsuccessful.

0 Successful.

ThresholdMulti

Syntax

```
int ThresholdMulti(
  CcImage* CImageIn,
  CcBinaryImage* CImageOut,
  STTHRESHOLD* stThreshold,
  int iNumberOfRegions);
```

Include File C_Thresh.h

Description Thresholds the given input image into a binary output image with respect to the given threshold structure.

Parameters

Name: CImageIn

Description: Image that was derived from the CcImage class.

Name: CImageOut

Description: Binary image that was derived from the CcImage class.

Name: stThreshold

Description: Pointer to an array of multiple thresholding structures.

Name: iNumberOfRegions

Description: Size of an array of multiple thresholding structures.

Notes This method thresholds the entire image; it does not use an ROI. The input image must be either an 8-bit grayscale, 32-bit grayscale, floating-point grayscale, or RGB color image. The input and output images must be the same size.

The low and high threshold limits for each region are OR'ed together. Thus, the foreground values in the output image are the locations where any pixel in the input image was within the threshold limits for any region. The structure's *iRe*d, *iGreen* and *iBlue* elements are not used in this method. The multiple thresholding structure (STTHRESHOLD) is as follows:

```
struct STTHRESHOLD {
float fLOThresholdValue;
//High Limit for thresholding
float fHIThresholdValue;
//Low Limit for thresholding
int iRed;
```

Notes (cont.)

```
//Color of this region
int iGreen;
int iBlue;
};
```

Return Values

–1 Unsuccessful.

0 Successful.

InvertOutput

Syntax

```
int InvertOutput(BOOL bInvert);
```

Include File C_Thresh.h

Description Determines whether the output image is inverted.

Parameters

Name: bInvert

Description: Flag for inverting the output. It can contain one of the following values:

- TRUE – Output image is inverted.
- FALSE – Output image is not inverted.

Notes This method determines whether the output image is inverted the next time **Threshold( )** is called. By default, the output image is not inverted.

Return Values

–1 Unsuccessful.

0 Successful.

Example Program Using the Threshold Tool API

This example program opens an 8-bit image from disk, thresholds it between the values of 10 and 50, and saves the image to disk:

```
void SomeFunction(void)
{
/*Start of Dec Section*/
CcGrayImage256 * C8BitImage;
//8-bit grayscale image
CcThreshold* CThresh;
//Thresholding object
/*End of Dec Section*/

//Allocate memory for objects
 C8BitImage = new CcGrayImage256( );
 CThresh = new CcThreshold( );

//Open image from disk (or get image data from
//frame grabber)
 C8BitImage->OpenBMPFile(“image1.bmp”);

//Perform thresholding (do not invert output)
 CThresh->Threshold(C8BitImage,C8BitImage,10,50);

//Save output to disk
 C8BitImage->SaveBMPFile(“output.bmp”);

//Free memory
 delete C8BitImage; delete CThresh;
}
```

21

Creating GLI/2 Tools

Introduction

This chapter describes the operation of the GLI/2 tools, how they communicate with the main application, and how to create your own custom tools. For further information on the main application and how it uses tools, refer to the *GLOBAL LAB Image/2 User's Manual*.

What is a Tool?

The GLI/2 main application does not provide any analysis, modification, segmentation, or computational functionality of any type. This functionality is brought to the imaging application by tools. Tools are independent units that perform specific operations, such as creating histograms, creating line profiles, thresholding, filtering, opening and saving various types of file formats, communicating with imaging hardware, controlling machinery, accessing databases, and so on.

In programming terms, a tool is a modeless dialog box procedure wrapped inside of a DLL (dynamically linked library). This dialog box procedure is simply a user interface to an underlying C/C++ method or set of methods (such as a histogram method).

How a Tool Communicates with the Main Application

The main application communicates with tools by sending and receiving standard Windows messages.

When a significant event happens in the main application, such as a ROI being moved, the main application sends a message to notify all tools of this event. The tool can then process this message and do something about the ROI being moved, if desired.

Guidelines for Creating a Tool

If you create your own tool, it is recommended that you follow these guidelines:

- Keep the user interface of the tool and the functionality of the tool in separate modules.

  This is accomplished by placing the code that performs the actual operation in a separate class. The dialog box procedure then calls the class methods to perform the operation. For example, the GLI/2 package includes an example change tool (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default) that contains a user interface to a lower-level change class that provides the actual functionality of the tool. By keeping the user interface separate from the code that performs the operation, you can reuse the same class in your own imaging application and/or other tools.

- Use only GLI/2 messages to communicate with the main application.

  Experienced Windows programmers may have a desire to use faster or more direct approaches to communicate with the main application. However, using nonstandard approaches may lead to unpredictable results if you include other tools that were created by someone else.

- Keep the important controls of the tool in the upper-left corner of the tool.

  All tools are resizable. By keeping the most important controls in the upper-left corner of the tool, you can shrink the tool and still use it.

GLI/2 Messages

All tools communicate with the GLI/2 main application using a standard set of GLI/2 messages. The following messages are provided:

- Request messages,
- Notification messages,
- Command messages, and
- Point and click script messages.

When it initializes a tool, the main application creates a handle to itself and places it in the member variable *m_hMainApplication*. This handle is a member variable for all tools. You can use this handle to query the main application to find its active viewport using the GLI/2 message HL_GET_ACTIVE_VIEWPORT. The handle that this message returns is the handle of the active viewport; it is used in all future request and command messages from the tool to the main application.

Communication from the tools to the main application is always with respect to one of the main application's viewports, usually the active viewport. A tool can communicate with any viewport using a valid handle, even if it is not the active viewport. This is the case for tools that communicate with more than one viewport at a time, such as those that have an input image and an output image.

This section describes the GLI/2 messages in detail.

Note: All messages are defined in the DT_MGS.H header file, which is located in C:\GLI\GLI\INCLUDE, by default. Structures used by these messages are located in the DT_STR.H header file also found in the INCLUDE directory.

Request Messages

Request messages are sent from a tool to the main application to request some type of information. For further information on the returned information, see Chapter 2 starting on page 11.

Before using any request or command message, you must obtain a valid handle to a viewport in the main application. To do this, query the main application for its active viewport. Then, place the returned handle in the provided member variable *m_hActiveViewport* or in one of your own variables, as shown in the following example:


```
//Get Handle to Active Viewport
m_hActiveViewport = (HWND)::SendMessage(
      m_hMainApplication,HL_GET_ACTIVE_VIEWPORT,
      0,0L);
```

All future request messages can then use this or another valid handle. If you need to communicate with more than one viewport, you must first obtain a handle to each of the viewports (while each is the active viewport), and then store these handles in your own variables. Tools that have an input and output image require this type of storage.

All messages are sent to the main application using the standard Windows function **SendMessage**. For more information on the **SendMessage( )** function, see the Windows SDK API documentation.

A request message has the following form:

```
SendMessage(hViewport,HL_REQUEST, requested
   message, 0);
```

The parameters of the **SendMessage( )** function are as follows:

- *hViewport* – Handle to the desired viewport (*m_hActiveViewport*) from which you are requesting information.
- *HL_REQUEST* – The request message. This must be HL_REQUEST for all request messages.

- *Requested message* – One of the request messages described in detail in this section.

Note: All GLI/2 request messages start with the prefix: HLR_.

Request messages are sent from a tool to the main application to request some type of information. They are never sent from the main application to a tool or between tools.

The request messages are briefly described in Table 37.

Table 37: Request Messages

Request Message	Returned Information	Return Type
HLR_SUPPLY_IMAGE_OBJECT	The image associated with the given viewport.	CcImage*
HLR_SUPPLY_IMAGE_OBJECT_LIST	The entire list of images in memory.	CcList*
HLR_SUPPLY_ACTIVE_ROI_OBJECT	The active ROI in the given viewport.	CcRoiBase*
HLR_SUPPLY_ROI_OBJECT_LIST	The list of ROIs for the given viewport.	CcList*
HLR_SUPPLY_ROI_TYPE	ROI creation value type.	int
HLR_SUPPLY_VIEWPORTS_INSTANCE	The instance of the viewport.	int
HLR_SUPPLY_VIEWPORT_VIA_INSTANCE	The handle to the desired viewport.	HWND
HLR_SUPPLY_VIEWPORT_VIA_IMAGE	The handle to the desired viewport.	HWND
HLR_SUPPLY_NEW_VIEWPORT	A new viewport handle.	HWND

Table 37: Request Messages

Request Message	Returned Information	Return Type
HLR_SUPPLY_CALIBRATION_ OBJECT_LIST	The list of Calibration objects in the system.	CcList*
HLR_SUPPLY_DEFAULT_ CALIBRATION_OBJECT	The default Calibration object.	CcCalibration*
HLR_SUPPLY_VIEWPORT_ ARRAY	An array of handles for all viewports.	HWND*
HLR_SUPPLY_LIST_BY_NAME	A pointer to a specified list.	CcList*
HLR_IS_SCRIPT_RUNNING	A value of 1 if the script is running; a value of 0 if the script is not running.	int

The request messages are described in detail in the remainder of this section.

HLR_SUPPLY_IMAGE_OBJECT

Syntax

```
CImage =(CcImage*) ::
   SendMessage(
   hViewport,HL_REQUEST,
   HLR_SUPPLY_IMAGE_OBJECT,0);
```

Include File DT_Msg.h

Description Obtains the Image object that is associated with the given viewport.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name:	HL_REQUEST
Description:	Required for all request messages.
Name:	HLR_SUPPLY_IMAGE_OBJECT
Description:	Specific type of request message.
Name:	0
Description:	This request message does not require any parameter for *lParam*.

Notes

This message is used to obtain a pointer to the image that is associated (displayed) in the given viewport. You must cast the return value into CcImage* before you can use the object. This object can be any type of Image object derived from the CcImage* object base class. These include binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, 24-bit RGB color, and user-defined images. For more information on these types of image objects, see Chapter 2 on page 11.

If the viewport is not displaying an image, this message returns NULL.

Return Values

NULL	Unsuccessful.
CImage – a pointer to a GLI/2 Image object derived from CcImage.	Successful.

HLR_SUPPLY_IMAGE_OBJECT_LIST

Syntax

```
CList =(CcList*) ::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_IMAGE_OBJECT_LIST,0);
```

Include File DT_Msg.h

Description Obtains the GLI/2 main application's list of Image objects.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_IMAGE_OBJECT_LIST

Description: Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes The GLI/2 main application keeps a list of all Image objects in memory. You can obtain a pointer to this list by sending any viewport this message. All viewports return the same list; there is only one list in the system. You can then use this list to examine all the Image objects. Do not add your own created Image objects to the list or delete images from the list directly. Instead, use the command messages HLC_ADD_IMAGE_OBJECT_TO_LIST and HLC_DEL_IMAGE_OBJECT_FR_LIST. These messages notify other tools in the system of these events.

The Memory Image tool uses this message to obtain the list of images so that it can display the list.

Return Values

NULL	Unsuccessful.
A pointer to a GLI/2 CcList object.	Successful.

HLR_SUPPLY_ACTIVE_ROI_OBJECT

Syntax

```
CRoi =(CcRoiBase*) ::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_ACTIVE_ROI_OBJECT,
   0);
```

Include File DT_Msg.h

Description Obtains the given viewport's active ROI.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_ACTIVE_ROI_OBJECT

Description: Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes

Each viewport in the GLI/2 main application can have many ROIs associated with it. Only one of these ROIs can be active at a time. This message returns a pointer to the active ROI object for the given viewport. If the viewport has no active ROI, this method returns NULL.

There are two modes of operation in the main application with respect to ROIs. The ROIs can be attached to the viewport or to the image itself. In either case, only one active ROI can be associated with a viewport. This message always returns the active ROI and is transparent to which mode of operation the main application is in.

Notes (cont.) The ROI returned is derived from an GLI/2 CcRoiBase* base class (point, rectangular, elliptical, line, poly line, freehand line, poly freehand, or freehand ROI). For more information on these objects, see Chapter 2 on page 11.

Return Values

NULL	Unsuccessful.
A pointer to a GLI/2 ROI object.	Successful.

HLR_SUPPLY_ROI_OBJECT_LIST

Syntax

```
CRoiList =(CcList*) ::
   SendMessage(
   hViewport,HL_REQUEST,
   HLR_SUPPLY_ROI_OBJECT_LIST,0);
```

Include File DT_Msg.h

Description Obtains the given viewport's list of ROI objects.

Parameters

Name:	hViewport
Description:	Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.
Name:	HL_REQUEST
Description:	Required for all request messages.
Name:	HLR_SUPPLY_ROI_OBJECT_LIST
Description:	Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes

Each viewport in the GLI/2 main application contains a list of ROIs. You can obtain a pointer to this list so that you can use and analyze the ROIs in this list. You can also add and delete ROI objects from this list, and add and delete ROI objects to/from the viewport's list using the command messages HLC_ROI_ADD and HLC_ROI_DELETE.

If you need to add or delete many ROIs from this list, use the methods of the CcList object. Make sure that the last ROI added or deleted from the list using the command messages; these messages update all tools and viewports. If you do not add or delete the last ROI in this manner, the tools and the viewports are not updated.

You can add and delete all ROIs to/from the list using the command messages, but this is slower than doing it directly. Thus, if you want to add ten new ROI objects to the list, add the first nine directly, and add the tenth ROI using the command message HLC_ROI_ADD. This is how the Blob Analysis tool adds and deletes ROIs.

Notes (cont.)

There are two modes of operation in the main application with respect to ROIs. The ROIs can be attached to the viewport or to the image itself. In either case, only one ROI list can be associated with a viewport at any given time. This message always returns the correct ROI list and is transparent to which mode of operation the main application is in.

The ROI list returned contains all ROIs associated with the given viewport. It can contain any combination of point, rectangular, elliptical, line, poly line, freehand line, poly freehand, and freehand ROIs. For more information on these objects, see Chapter 2 on page 11.

Return Values

NULL	Unsuccessful.
A pointer to a GLI/2 CcList object.	Successful.

HLR_SUPPLY_ROI_TYPE

Syntax

```
iRoiType = (int) ::
   SendMessage(
   hViewport,HL_REQUEST,
   HLR_SUPPLY_ROI_TYPE,0);
```

Include File DT_Msg.h

Description Obtains the GLI/2 main application's ROI creation type.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_ROI_TYPE

Description: Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes

You must first set the type of ROI to be created using the menu item Option | ROI Type or the ROI tool. You can query the main application to find out the creation type for the ROIs by using this message.

An ROI type can be one of the following values:

- ROI_POINT – Point.
- ROI_LINE – Line.
- ROI_PLINE – Poly Line.
- ROI_FLINE – Freehand Line.
- ROI_RECT – Rectangle.
- ROI_ELLIPSE – Ellipse.
- ROI_FREEHAND – Freehand.
- ROI_PFREEHAND – Poly Freehand.

Notes (cont.) The ROI tool uses this value to query the main application for the ROI creation type. You can then set the ROI's creation type using the command message HLC_SET_ROI_TYPE_TO.

Return Values

–1	Unsuccessful.
The main application's ROI creation type.	Successful.

HLR_SUPPLY_VIEWPORTS_INSTANCE

Syntax

```
iViewNumber = (int)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_VIEWPORTS_INSTANCE,
   0);
```

Include File DT_Msg.h

Description Obtains the instance of the given view.

Parameters

Name:	hViewport
Description:	Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.
Name:	HL_REQUEST
Description:	Required for all request messages.
Name:	HLR_SUPPLY_VIEWPORTS_INSTANCE
Description:	Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes All viewports in the system have an associated instance or viewport number. If you want to know the number of the viewport you can use this message.

Return Values

–1 Unsuccessful.

The given viewport's instance. Successful.

HLR_SUPPLY_VIEWPORT_VIA_INSTANCE

Syntax

```
hViewport = (HWND)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_VIEWPORT_VIA_
     INSTANCE,(LPARAM)iInstance);
```

Include File DT_Msg.h

Description Obtains the viewport with the given instance.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_VIEWPORT_VIA_ INSTANCE

Description: Specific type of request message.

Name:	iInstance
Description:	The instance of the viewport for which you are searching. This is an integer variable.
Notes	All viewports in the system have an associated instance or viewport number. If you want to know the viewport associated with a certain instance, you can use this message.

Return Values

NULL	Unsuccessful.
The handle to the viewport for the given instance.	Successful.

HLR_SUPPLY_VIEWPORT_VIA_IMAGE

Syntax

```
hViewport = (HWND)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_VIEWPORT_VIA_IMAGE,
   (LPARAM)CImage);
```

Include File DT_Msg.h

Description Obtains the viewport showing the given image.

Parameters

Name:	hViewport
Description:	Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.
Name:	HL_REQUEST
Description:	Required for all request messages.

Name: HLR_SUPPLY_VIEWPORT_VIA_IMAGE

Description: Specific type of request message.

Name: CImage

Description: Pointer to the image who's viewport you are requesting.

Notes All viewports in the system can have an associated image that they are displaying, or they can be blank. If the image you are searching for is not being displayed by any viewport, this message returns NULL. If two or more viewports are showing the given image, only one viewport is returned.

Return Values

NULL Unsuccessful.

The handle to the viewport. Successful.

HLR_SUPPLY_NEW_VIEWPORT

Syntax

```
hViewport = (HWND)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_NEW_VIEWPORT,0);
```

Include File DT_Msg.h

Description Creates a new viewport and returns the handle to it.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name:	HL_REQUEST
Description:	Required for all request messages.
Name:	HLR_SUPPLY_NEW_VIEWPORT
Description:	Specific type of request message.
Name:	0
Description:	This request message does not require any parameter for *lParam*.
Notes	This message opens up a new blank viewport. You can then place an image in it using the command message HLC_SET_IMAGE_OBJECT.

Return Values

NULL	Unsuccessful.
The handle to the new viewport.	Successful.

HLR_SUPPLY_CALIBRATION_OBJECT_LIST

Syntax

```
CList = (CcList*)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_CALIBRATION_OBJECT_
      LIST,0);
```

Include File DT_Msg.h

Description Obtains the GLI/2 main application's list of Calibration objects.

Parameters

Name:	hViewport
Description:	Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.
Name:	HL_REQUEST
Description:	Required for all request messages.
Name:	HLR_SUPPLY_CALIBRATION_OBJECT_LIST
Description:	Specific type of request message.
Name:	0
Description:	This request message does not require any parameter for *lParam*.

Notes All Calibration objects in the system are held in a single list. Use this message to obtain a pointer to this list. The list is a CcList object. For more information see Chapter 2 on page 11.

Return Values

NULL	Unsuccessful.
A pointer to the Calibration object list.	Successful.

HLR_SUPPLY_DEFAULT_CALIBRATION_OBJECT

Syntax

```
CCal = (CcCalibration*)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_DEFAULT_CALIBRATION_
     OBJECT,0);
```

Include File DT_Msg.h

Description Obtains the default Calibration object for the system from the main application.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_DEFAULT_CALIBRATION_OBJECT

Description: Specific type of request message.

Name: 0

Description: This request message does not require any parameter for *lParam*.

Notes All files opened from disk use the default Calibration object for converting pixel measurements to real-world measurements. To get a pointer to this default Calibration object, use this message.

Return Values

NULL Unsuccessful.

A pointer to the default Calibration object. Successful.

HLR_SUPPLY_VIEWPORT_ARRAY

Syntax

```
phViewportArray = (HWND*)::
   SendMessage(hViewport,
   HL_REQUEST,
   HLR_SUPPLY_VIEWPORT_ARRAY,
   LPARAM)&iNumOfViewports);
```

Include File DT_Msg.h

Description Obtains a list of all open viewports from the main application.

Parameters

Name: hViewport

Description: Viewport from which you are requesting the information. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_VIEWPORT_ARRAY

Description: Specific type of request message.

Name: iNumOfViewports

Description: Pointer to a user-defined integer to hold the number of viewports returned in the array.

Notes It may be desirable to work on all viewports at one time. You can obtain a list of all open viewports at one time using this method.

Return Values

NULL	Unsuccessful.
A pointer to an array of viewports.	Successful.

Example The following is example gets the list of all open viewports and sends each a message to restore them.

```
//EXAMPLE OF USING THE VIEWPORT
//ARRAY
void CcTool::OnRestoreAll( )
{
int x,iNumOfViewports;
HWND* phViewportArray;
//Get Viewport Array
phViewportArray = (HWND*)::
   SendMessage(m_hActiveViewport,
   HL_REQUEST,
   HLR_SUPPLY_VIEWPORT_ARRAY,
   (LPARAM)&iNumOfViewports);

if(phViewportArray == NULL)
   return;
//Restore All Viewports


for(x=0; x<iNumOfViewports; x++)
    {
::SendMessage(
   phViewportArray[x],HL_COMMAND,
   HLC_MANAGE_VIEWPORT,
   (LPARAM)SW_RESTORE);
    }
}
```

HLR_SUPPLY_LIST_BY_NAME

Syntax

```
CcList *pCList = (CcList*)::
   SendMessage(m_hActiveViewport,
   HL_REQUEST,
   HLR_SUPPLY_LIST_BY_NAME,
   LPARAM)cString);
```

Include File DT_Msg.h

Description Retrieves a pointer to a specified list based on its name.

Parameters

Name: m_hActiveViewport

Description: The active viewport from which you are requesting the information.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_SUPPLY_LIST_BY_NAME

Description: Specific type of request message.

Name: cString

Description: The name of the specified list (such as number or string).

Notes None

Return Values

pCList A pointer to a list.

HLR_IS_SCRIPT_RUNNING

Syntax

```
int iIsRunning = (int)::
   SendMessage(m_hActiveViewport,
   HL_REQUEST,
   HLR_IS_SCRIPT_RUNNING, 0);
```

Include File DT_Msg.h

Description Determines whether or not the specified script is running.

Parameters

Name: m_hActiveViewport

Description: The active viewport from which you are requesting the information.

Name: HL_REQUEST

Description: Required for all request messages.

Name: HLR_IS_SCRIPT_RUNNING

Description: Specific type of request message.

Name: 0

Description: No information is needed for this message.

Notes None

Return Values

iIsRunning A value of 1 if the script is running; a value of 0 if the script is not running.

Notification Messages

Notification messages are sent from the main application to all open tools when a significant event happens in the main application; notification messages notify the tools of the event. They are never sent from the tools to the main application or between tools. You do not need to return anything to the main application. A tool does not have to process any of these messages. Your tool should process only the messages that make sense for its operation.

All notification messages are sent from the main application to all open tools using the standard Windows **SendMessage** function. The syntax is as follows:

```
SendMessage(hTool, HL_NOTIFY,specific notification
   message, message specific information)
```

Information about the event is often contained in the *lParam* parameter of the message. For further information on the contained information, see Chapter 2 on page 11.

All notification messages are processed in a tool by processing the HL_NOTIFY message sent by the main application. This message map is already set up to map to the **HLNotify( )** message handler in the example change tool that is included in the GLI/2 package (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default). Thus, all notification messages should be processed in the switch statement of the **HLNotify( )** message handler. You can process none, all, or some of the notification messages in the switch statement. Which notification messages you process is determined by the desired functionality of your tool.

The following example shows starting code for this event handler and the notification messages HLN_NEW_IMAGE_OBJECT and HLN_VIEWPORT_ACTIVATED:

```
//*************** H L  N O T I F Y **************//
LRESULT CcTool::HLNotify(WPARAM wParam,
   LPARAM lParam)
{
/*Start of Dec Section*/
/*End  of Dec Section*/

switch(wParam)
{
   case HLN_NEW_IMAGE_OBJECT:
   (process this message here)
   return(TRUE);
   break;

   case HLN_VIEWPORT_ACTIVATED:
   (process this message here)
   return(TRUE);
   break;
}
   return(TRUE);
}
//************* H L  N O T I F Y ***************//
```

Note: All GLI/2 notification messages start with the prefix HLN_.

The notification messages are briefly described in Table 38.

Table 38: Notification Messages

Specific Notification Message	Description of Message
HLN_NEW_IMAGE_OBJECT	A new image has been added to the main application's image list. A pointer to the image that was added is given in *lParam*.
HLN_DELETED_IMAGE_OBJECT	An image has been deleted from the main application's image list. A pointer to the image that was deleted is given in *lParam*.
HLN_DELETING_IMAGE_OBJECT	An image is about to be deleted from the main application's image list. A pointer to the image to be deleted is given in *lParam*.
HLN_ROI_TYPE_CHANGE	The ROI creation type has changed in the main application. The new ROI type is given in *lParam*.
HLN_ROI_CREATED	An ROI has been created in the main application. A pointer to the ROI object is given in *lParam*.
HLN_DELETED_ROI_OBJECT	An ROI was deleted in the main application. A pointer to the deleted ROI object is given in *lParam*.
HLN_DELETING_ROI_OBJECT	An ROI is about to be deleted in the main application. A pointer to the ROI to be deleted is given in *lParam*.
HLN_ROI_ACTIVATED	An ROI has become activated in the main application. A pointer to the activated ROI object is given in *lParam*.
HLN_ROI_COPIED	An ROI has been created (or is being created) by copying another ROI. A pointer to the newly created (copied) ROI is given in *lParam*.
HLN_ROI_MOVED	An ROI is being moved in the main application. A pointer to the ROI is given in *lParam*.

Table 38: Notification Messages (cont.)

Specific Notification Message	Description of Message
HLN_ROI_RESIZED	An ROI has been drawn with the mouse (created) in the main application. The user is currently resizing the ROI. A pointer to the ROI is given in *lParam*.
HLN_MOUSEMOVE	The mouse is moving in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_LBUTTONDOWN	The user has depressed the left mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_LBUTTONUP	The user has released the left mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_RBUTTONDOWN	The user has depressed the right mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_RBUTTONUP	The user has released the right mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_LBUTTONDBLCLK	The user has double-clicked the left mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.
HLN_RBUTTONDBLCLK	The user has double-clicked the right mouse button in the main application. A pointer to a structure describing the mouse is given in *lParam*.

Table 38: Notification Messages (cont.)

Specific Notification Message	Description of Message
HLN_VIEWPORTS_IMAGE_ CHANGED	An image in a viewport has been redrawn (usually as a result of a tool changing the image's data). A pointer to the image is given in *lParam*.
HLN_VIEWPORT_ACTIVATED	A different viewport has become the active viewport. A handle to the activated viewport is given in *lParam*.
HLN_VIEWPORT_DEACTIVATED	The active viewport has been deactivated (because another viewport is now the active viewport). A handle to the deactivated viewport is given in *lParam*.
HLN_OBJECT_NAME_CHANGED	An object has had its name changed. A pointer to the object is given in *lParam*.
HLN_NEW_CALIBRATION_OBJECT	A new Calibration object has been added to the system. A pointer to the new object is given in *lParam*.
HLN_DELETED_CALIBRATION_ OBJECT	A Calibration object has been deleted. A pointer to the object that has been deleted is given in *lParam*.
HLN_DELETING_CALIBRATION_ OBJECT	A Calibration object is about to be deleted. A pointer to the object to be deleted is given in *lParam*.
HLN_DEFAULT_CALIBRATION_ OBJECT_CHANGED	The default Calibration object has changed. A pointer to the new default Calibration object is given in *lParam*.
HLN_SCRIPT_RUNNING	A point and click script has been activated or run.
HLN_LIST_CHANGED	One of the internal lists (such as number, string, or roi) has been updated.

The notification messages are described in detail in the remainder of this section.

HLN_NEW_IMAGE_OBJECT

Syntax

```
//******* H L  N O T I F Y ******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_NEW_IMAGE_OBJECT:
CcImage* CImage =
   (CcImage*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that a new image has been added to the main application's image list.

Parameters

Name: CcImage *

Description: A pointer to the image that was added to the main application's image list is contained in the lParam of the message.

Notes When a new image is created (from taking a picture using a picture tool or from opening an image from disk) and is added to the main application's image list, this message is sent. The image that was added to the list is contained in the *lParam* parameter of the message. You can obtain and use this pointer by casting *lParam* to a CcImage* pointer.

Notes (cont.) This message is always sent after an image is added to the image list using the command message HLC_ADD_IMAGE_OBJECT_TO_LIST.

HLN_DELETED_IMAGE_OBJECT

Syntax

```
//******** H L  N O T I F Y *****//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_DELETED_IMAGE_OBJECT:
CcImage* CImage = (
   CcImage*)lParam;
(stop using this image, DO NOT USE
   THE POINTER TO THE IMAGE!)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that an image has been deleted from the main application's image list.

Parameters

Name: CcImage *

Description: A pointer to the image that was deleted from the main application's image list is contained in the *lParam* parameter of the message.

Notes When an image is deleted, the image is removed from the main application's image list. A tool may be using this image using its pointer. All tools that store pointers to images for usage should check this message to make sure that the images they are using have not been deleted. When a tool receives this message, it can check its images against the deleted image pointer given in *lParam*; it must not use the deleted image using its pointer (because the image has already been deleted when the tool gets this message).

This message is always sent after an image has been deleted from the image list using the command message HLC_DEL_IMAGE_OBJECT_FR_LIST. Never directly delete an image that is contained in the main application's image list; use this message so that other tools know that the image has been deleted. If your tool was responsible for creating an image (and the image was never attached to the main application's image list), you can delete the image directly (this is because no other tools should be using the image).

HLN_DELETING_IMAGE_OBJECT

Syntax

```
//******** H L  N O T I F Y *****//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_DELETING_IMAGE_OBJECT:
CcImage* CImage = (
   CcImage*)lParam;
(stop using this image, YOU CAN
   USE THE POINTER TO THE IMAGE!)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that an image will be deleted from the main application's image list.

Parameters

Name: CcImage *

Description: A pointer to the image that is being deleted from the main application's image list is contained in the *lParam* parameter of the message.

Notes When an image is deleted, the image is removed from the main application's image list. A tool may be using this same image using its pointer. All tools that store pointers to images for usage should check this message to make sure that the images they are using have not been deleted. When it receives this message, a tool can do any clean up what it needs to using the image given in *lParam*.

This message is always sent out before an image has been deleted from the image list using the command message HLC_DEL_IMAGE_OBJECT_FR_LIST. Never directly delete an image that is contained in the main application's image list; use this message to inform other tools that the image has been deleted. If your tool was responsible for creating an image (and the image was never attached to the main application's image list), you can delete the image directly (this is because no other tools should be using the image).

HLN_ROI_TYPE_CHANGE

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_TYPE_CHANGE:
int iType = (int)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that the ROI creation type has changed.

Parameters

Name: int

Description: A new ROI creation type variable is given in *lParam*.

Notes When you create an ROI in the main application, you must first set the type of ROI to be created. An integer variable describing the new type is given in *lParam*.

This message is generated when you change the ROI creation type using the main application's menu item Option | ROI Type, the ROI bar, the ROI tool, or the command message HLC_SET_ROI_TYPE_TO.

Notes (cont.) This type can be one of the following values:

- ROI_POINT
- ROI_LINE
- ROI_FLINE
- ROI_PLINE
- ROI_RECT
- ROI_ELLIPSE
- ROI_FREEHAND
- ROI_PFREEHAND

HLN_ROI_CREATED

Syntax

```
//******** H L  N O T I F Y*****//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_CREATED:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that an ROI has been created.

Parameters

Name: CcRoiBase *

Description: A pointer to the created ROI that is given in *lParam*.

Notes When you create an ROI in the main application, the main application sends out this message. This message is also generated when a tool creates an ROI and adds the ROI to a viewport's list of ROIs using the command message HLC_ROI_ADD. A pointer to the newly created ROI is given in *lParam*.

HLN_DELETED_ROI_OBJECT

Syntax

```
//*****H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_DELETED_ROI_OBJECT:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message accordingly...,
   DO NOT USE THE POINTER TO THE
   ROI!)
return(TRUE);
}
return(TRUE);
}
//**** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that an ROI has been deleted.

Parameters

Name: CcRoiBase *

Description: Pointer to the deleted ROI that is given in *lParam*.

Notes When you delete an ROI in the main application, the main application sends out this message. This message is also generated when a tool deletes a ROI from a viewport's list of ROIs using the command message HLC_ROI_DELETE. A pointer to the deleted ROI is given in *lParam*. At this point the ROI object has already been deleted; you cannot use the pointer to the deleted object.

HLN_DELETING_ROI_OBJECT

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_DELETING_ROI_OBJECT:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message
   accordingly...,YOU CAN USE THE
   POINTER TO THE ROI!)
return(TRUE);
}
return(TRUE);
}
//****** H L  N O T I F Y *******//
```

Include File DT_Msg.h

Description Notifies a tool that a ROI is being deleted.

Parameters

Name: CcRoiBase *

Description: A pointer to the ROI to be deleted is given in *lParam*.

Notes When you delete an ROI in the main application, the main application sends this message. This message is also generated when a tool deletes an ROI from a viewport's list of ROIs using the command message HLC_ROI_DELETE. A pointer to the deleted ROI is given in *lParam*. At this point, the ROI object can still be used because its object has not yet been deleted.

HLN_ROI_ACTIVATED

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_ACTIVATED:
CcRoiBase* CRoi =
   (CcRoiBase*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that an ROI has been activated.

Parameters

Name: CcRoiBase *

Description: A pointer to the activated ROI is given in *lParam*.

Notes When a user activates an ROI in the main application, the main application sends this message. A pointer to the activated ROI is given in *lParam*.

HLN_ROI_COPIED

Syntax

```
/**** H L  N O T I F Y *********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_COPIED:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that a new ROI has been created by copying an existing ROI.

Parameters

Name: CcRoiBase *

Description: A pointer to the new ROI is given in *lParam*.

Notes When a user copies an ROI in the main application, the main application sends this message. A pointer to the new ROI is given in *lParam*.

HLN_ROI_MOVED

Syntax

```
//******* H L  N O T I F Y ******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_MOVED:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that an ROI is being moved and/or resized.

Parameters

Name: CcRoiBase *

Description: A pointer to the ROI that is being moved/resized is given in *lParam*.

Notes When you move or resize an ROI in the main application, the main application sends this message. A pointer to the ROI is given in *lParam*.

HLN_ROI_RESIZED

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_ROI_RESIZED:
CcRoiBase* CRoi = (
   CcRoiBase*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y ********//
```

Include File Include DT_Msg.h

Description Notifies a tool that an ROI is being resized.

Parameters

Name: CcRoiBase *

Description: A pointer to the ROI that is being resized is given in *lParam*.

Notes While an ROI is being created using the mouse in the main application, the main application sends this message every time the you resize the ROI. This is sent only during the creation stage of an ROI. If you want to know when an ROI is being resized after it has been created, use the HLN_ROI_MOVED notification message.

HLN_MOUSEMOVE

Syntax

```
//******* H L  N O T I F Y ******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_MOUSEMOVE:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);

}
return(TRUE);
}
//******** H L  N O T I F Y *****//
```

Include File DT_Str.h

DT_Msg.h

Description Notifies a tool that the mouse is being moved within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_LBUTTON – Set if the left mouse button is down.
- MK_RBUTTON – Set if the right mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes When the mouse is moved within a viewport (does not have to be the active viewport), the main application sends this message. If you choose to process this message, do so quickly. If you take too long to process this message, a jerky response is added to the overall application.

The Pixel Analysis tool uses this message. Its processing time is short and performs its functionality only when the left button of the mouse is depressed. It is a good idea to perform your functionality only if some type of key-mouse button combination is activated instead of processing on every mouse move. This allows you to keep your tool open, but activate the tool only when a specific key-mouse button combination is activated.

HLN_LBUTTONDOWN

Syntax

```
//*** H L  N O T I F Y **********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_LBUTTONDOWN:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the left mouse button has been depressed within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_RBUTTON – Set if the right mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT– Set if the SHIFT key is down.

Name: vpCImage

Description: Pointer to the image associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you depress the left mouse button in an open viewport in the main application.

HLN_LBUTTONUP

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_LBUTTONUP:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the left mouse button has been released within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_RBUTTON – Set if the right mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image that is associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you release the left mouse button within a viewport in the main application.

HLN_RBUTTONDOWN

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_RBUTTONDOWN:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the right mouse button has been depressed within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_LBUTTON – Set if the left mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you depress the right mouse button within a viewport in the main application.

HLN_RBUTTONUP

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_RBUTTONUP:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y *******//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the right mouse button has been released within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_LBUTTON – Set if the left mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image that is associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you release the right mouse button within a viewport in the main application.

HLN_LBUTTONDBLCLK

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_LBUTTONDBLCLK:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//*** H L  N O T I F Y **********//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the left mouse button has been double-clicked within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_RBUTTON – Set if the right mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you double-click the left mouse button within a viewport in the main application.

HLN_RBUTTONDBLCLK

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_RBUTTONDBLCLK:
STMOUSEMOVE* stMouse = (
   STMOUSEMOVE*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File

DT_Str.h

DT_Msg.h

Description

Notifies a tool that the right mouse button has been double-clicked within a viewport in the main application.

Parameters

Name: STMOUSEMOVE *

Description: A pointer to a structure describing the mouse information is given in *lParam*.

Name: stMousePoint

Description: A POINT structure describing the x,y-location of the mouse cursor in the viewport (in image coordinates).

Name: stSubMousePoint

Description: An STPOINTs structure describing the x,y-location of the mouse cursor in the viewport (in sub-pixel image coordinates).

Name: nFlags

Description: Windows SDK flags given with the WM_MOUSE_MOVE message. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

- MK_CONTROL – Set if the CTRL key is down.
- MK_LBUTTON – Set if the left mouse button is down.
- MK_MBUTTON – Set if the middle mouse button is down.
- MK_SHIFT – Set if the SHIFT key is down.

Name: vpCImage

Description: A pointer to the image associated with the viewport that the mouse is in.

Name: hWnd

Description: Handle to the viewport that the mouse is in.

Notes This message is sent when you double-click the right mouse button within a viewport in the main application.

HLN_VIEWPORTS_IMAGE_CHANGED

Syntax

```
//** H L  N O T I F Y **********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_VIEWPORTS_IMAGE_CHANGED:
CcImage* CImage = (
   CcImage*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******** H L  N O T I F Y *****//
```

Include File DT_Msg.h

Description Notifies a tool that an image has been changed.

Parameters

Name: CcImage*

Description: A pointer to the image that has been changed is given in *lParam*.

Notes When a tool changes an image (such as the Filter tool), the tool commands the main application to redraw the image to reflect the change using the command message HLC_REDRAW_VIEW. When this happens, this message is sent. There is no viewport associated with this message because a single image can be displayed in multiple viewports.

HLN_VIEWPORT_ACTIVATED

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_VIEWPORT_ACTIVATED:
HWND hViewport = (HWND)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that a viewport has become activated.

Parameters

Name: HWND

Description: A handle to the activated viewport is given in *lParam*.

Notes This message is sent when a viewport becomes activated by clicking in it with the left mouse button. If an active viewport already exists, the viewport is deactivated and the HLN_VIEWPORT_DEACTIVATED notification message is sent.

HLN_VIEWPORT_DEACTIVATED

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_VIEWPORT_DEACTIVATED:
HWND hViewport = (HWND)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that a viewport has become deactivated.

Parameters

Name: HWND

Description: A handle to the deactivated viewport is given in *lParam*.

Notes When a viewport becomes activated by clicking in it with the left mouse button, the previous active viewport becomes deactivated (it is no longer the active viewport). When this happens, this message is sent. The newly activated viewport sends a HLN_VIEWPORT_ACTIVATED message letting the tools know which viewport is the active viewport.

HLN_OBJECT_NAME_CHANGED

Syntax

```
//****** H L  N O T I F Y *******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_OBJECT_NAME_CHANGED:
CcHLObject* CHLObject = (
   CcHLObject*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******** H L  N O T I F Y *****//
```

Include File DT_Msg.h

Description Notifies a tool that an object's name has been changed.

Parameters

Name: CcHLObject*

Description: A pointer to the object whose name has changed is given in *lParam*.

Notes When a tool (such as the Memory Images tool) changes an object's name, the tool must command the main application to notify the tools using the command message HLC_SEND_NAME_CHANGE_NOTIFICATION. When this happens, this message is sent. A pointer to the object that has had its name changed is given in *lParam*.

HLN_NEW_CALIBRATION_OBJECT

Syntax

```
//******* H L  N O T I F Y ******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_NEW_CALIBRATION_OBJECT:
CcCalibration* CCal = (
   CcCalibration*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//**** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that a new Calibration object has been added to the main application's Calibration object list.

Parameters

Name: CcCalibration*

Description: A pointer to the new Calibration object is given in *lParam*.

Notes When a tool (such as the Calibration tool) adds a new Calibration object to the system it does so by using the command message HLC_ADD_CALIBRATION_OBJECT_TO_LIST; then, this message is sent. A pointer to the object that has had its name changed is given in *lParam*.

HLN_DELETED_CALIBRATION_OBJECT

Syntax

```
//**** H L  N O T I F Y *********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case
   HLN_DELETED_CALIBRATION_OBJECT:
CcCalibration* CCal = (
   CcCalibration*)lParam;
(process message accordingly..,DO
   NOT USE THE POINTER TO THE
   OBJECT!)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y ********//
```

Include File DT_Msg.h

Description Notifies a tool that a Calibration object has been deleted from the main application's Calibration object list.

Parameters

Name: CcCalibration*

Description: A pointer to the Calibration object that has been deleted.

Notes When a tool (such as the Calibration tool) deletes a Calibration object from the system it does so by using the command message HLC_DEL_CALIBRATION_OBJECT_FR_ LIS; then, this message is sent. A pointer to the object that has been deleted is given in *lParam*. You cannot use the pointer to the Calibration object because it has already been deleted.

HLN_DELETING_CALIBRATION_OBJECT

Syntax

```
//**** H L  N O T I F Y *********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case
   HLN_DELETING_CALIBRATION_OBJECT
   :
CcCalibration* CCal = (
   CcCalibration*)lParam;
(process message accordingly, YOU
   CAN USE THE POINTER TO THE
   OBJECT!)
return(TRUE);
}
return(TRUE);
}
//***** H L  N O T I F Y *******//
```

Include File DT_Msg.h

Description Notifies a tool that a Calibration object will be deleted from the main application's Calibration object list.

Parameters

Name: CcCalibration*

Description: A pointer to the Calibration object to be deleted.

Notes When a tool (such as the Calibration tool) deletes a Calibration object from the system, it does so using the command message HLC_DEL_CALIBRATION_OBJECT_FR_LIST; then, this message is sent. A pointer to the object to be deleted is given in *lParam*. You can use the pointer to the Calibration object because it has not been deleted yet.

HLN_DEFAULT_CALIBRATION_OBJECT_CHANGED

Syntax

```
//******* H L  N O T I F Y ******//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case
   HLN_DEFAULT_CALIBRATION_OBJECT_
   CHANGED:
CcCalibration* CCal = (
   CcCalibration*)lParam;
(process message accordingly...
return(TRUE);
}
return(TRUE);
}
//******* H L  N O T I F Y ******//
```

Include File DT_Msg.h

Description Notifies a tool that the default Calibration object has changed to a new Calibration object.

Parameters

Name: CcCalibration*

Description: A pointer to the new default Calibration object.

Notes When an image is opened from disk and is the correct size, the image uses the default Calibration object to calculate all of its measurements. If this default Calibration object changes as a result of a tool using the command message HLC_SET_DEFAULT_CALIBRATION_OBJECT, this message is sent.

HLN_SCRIPT_RUNNING

Syntax

```
//** H L  N O T I F Y ***********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_SCRIPT_RUNNING:
BOOL bRunning = (BOOL)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******** H L  N O T I F Y *****//
```

Include File DT_Msg.h

Description Notifies every tool that is referenced by the Point and Click Script tool that a point and click script has been activated or run.

Parameters

Name: BOOL

Description: Boolean variable. If TRUE, the script is running; if FALSE, the script is not running.

Notes None

HLN_LIST_CHANGED

Syntax

```
//** H L  N O T I F Y **********//
LRESULT CcTool::HLNotify(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLN_LIST_CHANGED:
char* pListName = (char*)lParam;
(process message accordingly...)
return(TRUE);
}
return(TRUE);
}
//******** H L  N O T I F Y *****//
```

Include File DT_Msg.h

Description Notifies all active tools that one of the internal lists (such as the number, string, or roi) has been updated.

Parameters

Name: char*

Description: A pointer to a string which holds the name of the list that was modified.

Notes The tools can synchronize their GUIs with the changes to the internal lists once they receive this notification message.

Command Messages

Command messages are sent from a tool to the main application to instruct it to perform some type of action. They are never sent from the main application to a tool or between tools. For further information on the command parameter information, see Chapter 2 on page 11. Command messages are sent to the main application the same way as request messages.

Before using any request or command message, obtain a valid handle to a viewport in the main application by querying the main application for its active viewport. Then, place the returned handle in the provided member variable *m_hActiveViewport* or in one of your own variables, as shown in the following code:

```
//Get Handle to Active Viewport
m_hActiveViewport = (HWND)::SendMessage(
   m_hMainApplication,HL_GET_ACTIVE_VIEWPORT,0,0L);
```

All future command messages then use this or another valid handle. If you need to communicate with more than one viewport, you must first obtain a handle to each viewport (while each is the active viewport), and then store these handles in your own variables. Tools that have an input and output image require this type of storage.

All command messages are sent to the main application using the SDK **SendMessage( )** function. For more information on **SendMessage( )**, see the Windows SDK API documentation. All command messages must have the message parameter of the Windows SDK **SendMessage( )** set to HL_COMMAND. Command messages have no return value.

A command message has the following form:

```
SendMessage(hViewport,HL_COMMAND, command message,
   command information);
```


The parameters of the **SendMessage( )** function are as follows:

- *hViewport* – Handle to the desired viewport (*m_hActiveViewport*) to which you are sending the command.
- *HL_COMMAND* – The command message.
- *Command message* – One of the command messages described in detail in this section.
- *Command information* – Needed information so that the main application can perform the command.

Note: All GLI/2 command messages start with the prefix: HLC_.

The command messages are briefly described in Table 39.

Table 39: Command Messages

Command Message	Description of Message
HLC_FILE_OPEN	Opens the given BMP file and places the full path name of the BMP file to open in *lParam*.
HLC_FILE_SAVE	Saves the image in the given viewport as the full path name given in *lParam*.
HLC_SIZE_IMAGE_TO_WINDOW	Shows the image in the given viewport by stretching the image to fit within the viewport without changing the size of the viewport. Note that this message does not keep the aspect ratio of the image.
HLC_SIZE_IMAGE_AS_ACTUAL	Shows the image in the given viewport in its actual size. If the image is too big to fit in the viewport, scrollbars are added to the viewport. Note that this message does not change the size of the viewport, but does keep the aspect ratio of the image.
HLC_SIZE_WINDOW_TO_IMAGE	Shows the image in the given viewport in its actual size. If the image is larger than the current viewport, the viewport is resized to fit the entire size of the image. The aspect ratio of the image is kept.
HLC_ADD_IMAGE_OBJECT_TO_LIST	Adds the Image object given in *lParam* to the main application's image list.
HLC_DEL_IMAGE_OBJECT_FR_LIST	Deletes the Image object given in *lParam* from the main application's image list. Note that this message deletes the Image object for you. You do not need to delete the Image object again.
HLC_SET_IMAGE_OBJECT	Associates the Image object given in *lParam* with the given viewport. The viewport then displays the given image.

Table 39: Command Messages (cont.)

Command Message	Description of Message
HLC_CLEAR_IMAGE_OBJECT	Clears all viewports from their associated image if they are using the image given in *lParam*. The viewport no longer has an image associated with it. This message is usually called when an image is deleted from memory.
HLC_REDRAW_IMAGE_ OVERLAY	Redraws the image overlay being shown in the given viewport without redrawing the image.
HLC_REDRAW_VIEW	Redraws the image associated with the given viewport. This message is usually called after a tool changes the image shown in a viewport (the output image of a tool).
HLC_SET_LOGICAL_PALETTE_ TO	Redraws the image in the given viewport using the given color palette in *lParam*.
HLC_SHOW_PIXEL_GROUPING	Shows a group of pixels given in *lParam* in the given viewport. The pixels are described in a PIXELGROUPING structure. This is a nondestructive method of drawing on the viewport's associated image in color.
HLC_ADD_CALIBRATION_ OBJECT_TO_LIST	Adds a Calibration object to the list of Calibration objects in the system.
HLC_DEL_CALIBRATION_ OBJECT_FR_LIST	Deletes a Calibration object from the list of Calibration objects in the system.
HLC_SET_DEFAULT_ CALIBRATION_OBJECT	Sets the given Calibration object as the default Calibration object.
HLC_ACTIVATE_ROI	Activates the given ROI in the given viewport.

Table 39: Command Messages (cont.)

Command Message	Description of Message
HLC_ROI_DELETE_ALL	Deletes all the ROIs in the given viewport.
HLC_SET_ROI_TYPE_TO	Sets the ROI creation type in the main application.
HLC_SET_ROI_MODE_TO	Sets the ROI drawing mode in the main application.
HLC_ROI_ADD	Adds the ROI given in *lParam* to the given viewport.
HLC_ROI_DELETE	Deletes the ROI given in *lParam* from the given viewport.
HLC_SEND_NAME_CHANGE_ NOTIFICATION	Instructs the main application to send a notification name change message to all open tools. The object whose name has changed is placed in *lParam*.
HLC_SHOW_TOOL_BOX	Hides or shows the floating tool box.
HLC_MANAGE_VIEWPORT	Controls a viewport's restore, minimize, and maximize functionality.
HLC_MANAGE_MAINAPP	Controls the main application's restore, minimize, and maximize functionality.
HLC_POSITION_VIEWPORT	Positions and sizes a viewport.
HLC_POSITION_MAINAPP	Positions and sizes the main application.
HLC_ARRANGE_VIEWPORTS	Arranges all viewports with respect to tile vertical, tile horizontal, cascade, and arrange icons.
HLC_CLOSE_VIEWPORT	Closes the given viewport.
HLC_ACTIVATE_VIEWPORT	Activates the given viewport.

Table 39: Command Messages (cont.)

Command Message	Description of Message
HLC_ADD_LIST_TO_MAIN_LIST	Adds a user-defined list to the main object list.
HLC_SEND_LIST_CHANGE_ NOTIFICATION	Notifies the tools about a change in one of the lists that is managed by the main application.
HLC_ADD_TO_SCRIPT_TOOLS	Places a tool in the Point and Click Script tool.

The command messages are described in detail in the remainder of this section.

HLC_FILE_OPEN

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_FILE_OPEN,(LPARAM)
   cFileName);
```

Include File DT_Msg.h

Description Opens the specified *cFileName* from disk.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_FILE_OPEN

Description: Specific type of command message.

Name: cFileName

Description: Char string specifying the full path name of the bitmap file to open.

Notes Use this command to open a bitmap file (*.BMP) from disk. You need to specify the full path name to the image in the lParam.

GLI/2 supports five different image types: binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit RGB color. The file is opened using the current image type. You can set the image type to open using the menu item Option | Image Type.

After the main application opens the file, the application adds the image to its image list. It then notifies all tools of the new image using the notification message HLN_NEW_IMAGE_OBJECT.

You can also open the image directly, then add the image to the main application's image list using the command message HLC_ADD_IMAGE_OBJECT_TO_LIST.

HLC_FILE_SAVE

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_FILE_SAVE,(LPARAM)
   cFileName);
```

Include File DT_Msg.h

Description Saves the image in the given viewport to disk and gives the file the name given in *cFileName*.

Parameters

Name: hViewport

Description: Viewport to which are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_FILE_SAVE

Description: Specific type of command message.

Name: cFileName

Description: A char string specifying the full path name for the saved image.

Notes Use this command to save a bitmap file (*.BMP) to disk. You need to specify the full path name for the image in *lParam*.

GLI/2 supports five different image types: binary, 8-bit grayscale, 32-bit grayscale, floating-point grayscale, and 24-bit RGB color. The file is saved with its correct image type automatically.

HLC_SIZE_IMAGE_TO_WINDOW

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SIZE_IMAGE_TO_WINDOW,0);
```

Include File DT_Msg.h

Description Sets the given viewport's display mode to stretches its associated image so that the entire image is displayed in the viewport without changing the size of the viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SIZE_IMAGE_TO_WINDOW

Description: Specific type of command message.

Name: 0

Description: This message has no associated information.

Notes This message changes the display mode for the given viewport. Any image placed in this viewport is displayed so that the entire image is displayed in the viewport without resizing the viewport. This does not keep the aspect ratio of the image.

HLC_SIZE_IMAGE_AS_ACTUAL

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SIZE_IMAGE_AS_ACTUAL,0);
```

Include File DT_Msg.h

Description Sets the given viewport's display mode so that it shows its associated image in its actual size without changing the size of the viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SIZE_IMAGE_AS_ACTUAL

Description: Specific type of command message.

Name: 0

Description: This message has no associated information.

Notes This message changes the display mode for the given viewport. Any image placed in this viewport is displayed in its actual size without resizing the viewport. If the image is larger than the viewport, scrollbars are added to the viewport. If the image is smaller than the viewport, the viewport shrinks to fit the image. This keeps the aspect ratio of the image.

HLC_SIZE_WINDOW_TO_IMAGE

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SIZE_WINDOW_TO_IMAGE,0);
```

Include File DT_Msg.h

Description Sizes the given viewport to the same size as the image so that it shows the entire associated image in its actual size.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SIZE_WINDOW_TO_IMAGE

Description: Specific type of command message.

Name: 0

Description: This message has no associated information.

Notes This message changes the size of the viewport to fit the size of the image. It sets the mode of the viewport to HLC_SIZE_IMAGE_AS_ACTUAL. If the viewport is then resized using the mouse, scrollbars are added and the image is displayed in its actual size.

HLC_ADD_IMAGE_OBJECT_TO_LIST

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_ADD_IMAGE_OBJECT_TO_LIST,
   (LPARAM)CImage);
```

Include File DT_Msg.h

Description Adds an image to the main application's image list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ADD_IMAGE_OBJECT_TO_LIST

Description: Specific type of command message.

Name: CImage

Description: Pointer to a CcImage derived Image object that you want added to the main application's image list.

Notes Any tool can create an GLI/2 CcImage derived Image object or a custom CcImage derived Image object and then share this image with other tools by adding the image to the main application's image list. Send a pointer to the image in *lParam* when using this message. Once you add an image to the image list, you should not delete the image directly because another tool may be using it. If you wish to delete an image from the image list, use the HLC_DEL_IMAGE_OBJECT_FR_LIST command message.

Notes (cont.) After the main application adds the image to its image list, the application notifies all tools of the new image via the notification message HLN_NEW_IMAGE_OBJECT.

HLC_DEL_IMAGE_OBJECT_FR_LIST

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_DEL_IMAGE_OBJECT_FR_LIST,
   (LPARAM)CImage);
```

Include File DT_Msg.h

Description Deletes an image from the main application's image list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_DEL_IMAGE_OBJECT_FR_LIST

Description: Specific type of command message.

Name: CImage

Description: Pointer to a CcImage derived Image object that you want deleted from the main application's image list.

Notes If an image is in the main application's image list, you should not delete it directly because another tool may be using it. To delete such an image, use this message specifying the image you want deleted in *lParam*. When the main application deletes an image due to this message, the application notifies all tools using the notification message HLN_DEL_IMAGE_OBJECT. If you created an image and have not added it to the main application's image list, you need to delete it directly.

This message deletes the Image object as it removes it from the list. Do not delete the Image object yourself when using this message.

HLC_SET_IMAGE_OBJECT

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SET_IMAGE_OBJECT,
   LPARAM)CImage);
```

Include File DT_Msg.h

Description Associates the given image with the given viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SET_IMAGE_OBJECT

Description: Specific type of command message.

Name: CImage

Description: Pointer to the image you want to associate with the specified viewport.

Notes An image is displayed in a viewport by associating the image with the viewport. A single image can be associated with multiple viewports. A tool can create an image and then display this image in a viewport by associating the image with the viewport using this message. The Memory Images tool selects images into viewports using this message. Before you associate a newly created image (an image created by your tool) with a viewport, the image should be added to the main application's image list using the command message HLC_ADD_IMAGE_OBJECT_TO_LIST.

HLC_CLEAR_IMAGE_OBJECT

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_CLEAR_IMAGE_OBJECT,
   (LPARAM)CImage);
```

Include File DT_Msg.h

Description Unassociates the given Image object from all viewports without deleting the image object.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_CLEAR_IMAGE_OBJECT

Description: Specific type of command message.

Name: CImage

Description: Pointer to an Image object derived from a CcImage object.

Notes

If you want a tool to clear an image from all viewports but not delete the image or remove it from the main application's image list, you can use this message. Place a pointer to the image in the *lParam* parameter of this message. You do not have to send this message to each viewport associated with this image to clear them all; the main application does that for you.

If you want a tool to delete an Image object and remove this object from the main application's image list and from all viewports, use the command message HLC_DEL_IMAGE_OBJECT_FR_LIST.

HLC_REDRAW_IMAGE_OVERLAY

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_REDRAW_IMAGE_OVERLAY,0);
```

Include File DT_Msg.h

Description Redraws the image's overlay for the image in the given viewport without redrawing the image.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Parameter is required for all command messages.

Name: HLC_REDRAW_IMAGE_OVERLAY

Description: Specific type of command message.

Name: 0

Description: No information is needed for this message.

Notes If you add something to an image's overlay you can call this method to only redraw the image's overlay and not redraw the image. If you change the overlay (not just add to it) you should redraw both the image and its overlay using the message HLC_REDRAW_VIEW.

HLC_REDRAW_VIEW

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND, HLC_REDRAW_VIEW,
   (LPARAM)0);
```

Include File DT_Msg.h

Description Redraws the image that is associated with the given viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_REDRAW_VIEW

Description: Specific type of command message.

Name: 0

Description: No information is needed for this message.

Notes After a tool changes an image (such as the Filter tool), the tool must redraw the image so that you can see the change in the image. The main application redraws the image that is associated with the given viewport and all other viewports that are displaying this image. It also redraws the image's overlay if the image has one. For example, if you change an image and it is being displayed in five viewports, you need only send this message to the viewport where you obtained the image. The main application automatically changes the image in the other viewports.

When a viewport redraws its associated image due to this message, it notifies all tools using the notification message HLN_VIEWPORTS_IMAGE_CHANGED.

HLC_SET_LOGICAL_PALETTE_TO

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SET_LOGICAL_PALETTE_TO,
   LPARAM)colorpalette);
```

Include File DT_Msg.h

Description Sets the display mode of the viewport to the given type of color palette.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SET_LOGICAL_PALETTE_TO

Description: Specific type of command message.

Name: colorpalette

Description: Specifies the type of color palette with which to display its associated image. Can be one of the following values:

- CTABLE_TO_ORIG_RGB – Displays the image with its original color table. Only files opened from disk have an original color table.
- CTABLE_TO_LINR_RGB – Displays the image as an RGB image. You can use this color table to view an RGB, HSL, or grayscale image with false coloring.
- CTABLE_TO_INDEXED256 – Displays the image using 256 shades of gray.
- CTABLE_TO_INDEXED128 – Displays the image using 128 shades of gray.
- *CTABLE_TO_INDEXED064 – Displays the image using 64 shades of gray. This is the default.
- CTABLE_TO_RINDEXED256 – Displays the image using 256 colors that can be grayscale or RGB colors. This type of color table is used for thresholding using palette animation.

Description (cont.)

- CTABLE_TO_RINDEXED128 – Displays the image using 128 colors that can be grayscale or RGB colors. This type of color table is used for thresholding using palette animation.
- CTABLE_TO_RINDEXED064 – Displays the image using 64 colors that can be grayscale or RGB colors. This type of color table is used for thresholding using palette animation.

Notes

GLI/2 provides eight different ways to display the same image. You can display the same image in multiple viewports, each using a different color table.

The grayscale tables are used only for binary, 8-bit, 32-bit, and floating-point grayscale images.

GLI/2 supports many color tables because of the vast differences in hardware that you may be using or to which you may be porting your algorithm. The default color table is the CTABLE_TO_INDEXED064.

You can change the color table that is used by a viewport by activating the viewport, and then selecting your choice from the GLI/2 main application's menu item View | Color Table or by using a tool that is using this message.

The color table and the output LUT described in the main application's documentation are synonymous.

You can use the GLI/2 Display tool to display images with various color tables.

HLC_SHOW_PIXEL_GROUPING

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SHOW_PIXEL_GROUPING,
   (LPARAM)pPixels);
```

Include File DT_Msg.h

DT_Str.h

Description Shows a graphic consisting of a group of pixels in the given viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SHOW_PIXEL_GROUPING

Description: Specific type of command message.

Name: pPixels

Description: Pointer to a PIXELGROUPING structure.

Notes It is sometimes useful to display graphics in different colors on an image in a viewport. The Line Profile tool does this to mark the spot on an image that corresponds to a specific point on the line profile. To use this command, fill in a PIXELGROUPING structure and then send a pointer to the structure with this message to the viewport in which you want the graphics displayed.

```
PIXELGROUPING description:
struct PixelGroupTag {
int iRed,iGreen,iBlue;
int iNumOfPoints;
POINT* stPOINTS;
HGLOBAL hstPOINTS;
};
typedef struct PixelGroupTag
   PIXELGROUPING;
```

The *iRed*, *iGreen* and *iBlue* variables describe the color in which the graphic is displayed.

You need to set *iNumOfPoints* to the number of points in the graphic.

You need to allocate the memory for the points that make up the graphic using the SDK function **GlobalAlloc( )**. The returned global handle is placed in the *hstPOINTS* variable. Once allocated, you must **GlobalLock** the memory and cast the pointer into the variable *stPOINTS*. You need to free the memory later by calling **GlobalUnlock** and **GlobalFree**.

Example The following is an example of how to draw a red diagonal line made up of 100 points in a viewport:

```
{
PIXELGROUPING Pixels;

Pixels.iRed= 255;
Pixels.iGreen= 0;
Pixels.iBlue= 0;
Pixels.hstPOINTS=
   GlobalAlloc(GHND, 100
   *sizeof(POINT));
Pixels.stPOINTS= (POINT*)
   GlobalLock(Pixels.hstPOINTS);
for(int x=0; x<100; x++)
{
Pixels.stPOINTS.x = x;
Pixels.stPOINTS.y = x;
}
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SHOW_PIXEL_GROUPING,
   (LPARAM)&Pixels)
GlobalUnlock(Pixels.hstPOINTS);
GlobalFree(Pixels.hstPOINTS);
}
```

Do not forget to free the memory once you no longer need it (using **GlobalUnlock( )** and **GlobalFree( )**).

The line is not displayed in the viewport if the viewport has no image associated with it.

HLC_ADD_CALIBRATION_OBJECT_TO_LIST

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_ADD_CALIBRATION_OBJECT_TO_
     LIST,(LPARAM)CCalibration);
```

Include File DT_Msg.h

Description Adds the given Calibration object to the main application's Calibration object list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ADD_CALIBRATION_OBJECT_TO_ LIST

Description: Specific type of command message.

Name: CCalibration

Description: Pointer to a Calibration object to add to the list (class CcCalibration).

Notes Tools use Calibration objects to calculate their measurements in calibrated units. The main application keeps a list of all Calibration objects in the system. You can add a new Calibration object to this list by using this message. After the Calibration object is added to the list, the main application notifies all tools using the notification message HLN_NEW_CALIBRATION_OBJECT.

HLC_DEL_CALIBRATION_OBJECT_FR_LIST

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_DEL_CALIBRATION_OBJECT_FR_
     LIST,(LPARAM)CCalibration);
```

Include File DT_Msg.h

Description Deletes the given Calibration object from the main application's Calibration object list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_DEL_CALIBRATION_OBJECT_FR_LIST

Description: Specific type of command message.

Name: CCalibration

Description: Pointer to a Calibration object to be deleted from the list (class CcCalibration).

Notes Tools use Calibration objects to calculate their measurements in calibrated units. The main application keeps a list of all Calibration objects in the system. You can remove a Calibration object from this list by using this message. After the Calibration object is removed from the list, the main application notifies all tools using the notification message HLN_DELETING_CALIBRATION_OBJECT and the message HLN_DELETED_CALIBRATION_OBJECT.

Any Image objects using the deleted Calibration object is disassociated from it automatically.

HLC_SET_DEFAULT_CALIBRATION_OBJECT

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SET_DEFAULT_CALIBRATION_
   OBJECT, (LPARAM)CCalibration);
```

Include File DT_Msg.h

Description Sets the given Calibration object as the default Calibration object within the system.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SET_DEFAULT_CALIBRATION_OBJECT

Description: Specific type of command message.

Name: Ccalibration

Description: A pointer to a Calibration object that becomes the default Calibration object (class CcCalibration).

Notes Tools use Calibration objects to calculate their measurements in calibrated units. The main application keeps a list of all Calibration objects in the system. When a file is opened from disk and it is the correct size, the application uses the default Calibration object to calculate its measurements. You can set the default Calibration object using this message. The main application notifies all tools using the notification message HLN_DEFAULT_CALIBRATION_OBJECT_CHANGED.

HLC_ACTIVE_ROI

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_ACTIVATE_ROI,(LPARAM)CRoi);
```

Include File DT_Msg.h

Description Makes the given ROI the active ROI within the given viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ACTIVATE_ROI

Description: Specific type of command message.

Name: CRoi

Description: Pointer to a ROI object derived from a CcRoiBase object.

Notes Most tools work on the active ROI when they perform their calculations. You can set the active ROI by using this message. The main application notifies the tools of this using the notification message HLN_ROI_ACTIVATED.

HLC_ROI_DELETE_ALL

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_ROI_DELETE_ALL,0);
```

Include File DT_Msg.h

Description Deletes all the ROIs in the given viewport.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ROI_DELETE_ALL

Description: Specific type of command message.

Name: 0

Description: No information is needed for this message

Notes After adding several ROIs to a viewport you may need to delete the ROIs. You can delete all ROIs in the given viewport by using this message. The main application notifies the tools for each ROI that it is deleted using the notification message HLN_DELETING_ROI_OBJECT and the message HLN_DELETED_ROI_OBJECT.

HLC_SET_ROI_TYPE_TO

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SET_ROI_TYPE_TO,
   (LPARAM)iType);
```

Include File DT_Msg.h

Description Sets the ROI type creation in the main application to the desired type.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SET_ROI_TYPE_TO

Description: Specific type of command message.

Name: iType

Description: Type of ROI creation desired. It can be one of the following:

- ROI Type – Description.
- ROI_POINT – Point.
- ROI_RECT – Rectangular.
- ROI_LINE – Line.
- ROI_FLINE – Freehand Line.
- ROI_PLINE – Poly Freehand Line.

Description (cont.):

- ROI_ELLIPSE – Elliptical.
- ROI_FREEHAND – Freehand.
- ROI_PFREEHAND – Poly Freehand.

Notes Instead of having to select the ROI type manually from the main application or ROI tool, you can use this message to set the ROI creation type. The main application notifies the tools of this using the notification message HLN_ROI_TYPE_CHANGE.

HLC_SET_ROI_MODE_TO

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SET_ROI_MODE_TO,
   (LPARAM)iMode);
```

Include File DT_Msg.h

Description Sets the ROI mode of action in the main application to the desired type.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SET_ROI_MODE_TO

Description: Specific type of command message.

Name: iMode

Description: Mode of ROI action desired. It can be one of the following:

- HLROI_MODE_OFF – No default action occurs.
- HLROI_MODE_DRAW – Current ROI type is created.
- HLROI_MODE_MOVE – Active ROI is moved / resized.
- HLROI_MODE_COPY – Active ROI is copied.
- HLROI_MODE_DELETE – Active ROI is deleted.
- HLROI_MODE_ACTIVATE – Any ROI is activated.

Notes The ROI mode of action is the action that results when you perform mouse operations in a viewport. If the ROI mode is set to HLROI_MODE_DRAW, an ROI is created. If the ROI mode is set to HLROI_MODE_OFF, no default action occurs.

HLC_ROI_ADD

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_ROI_ADD,(LPARAM)CRoi);
```

Include File DT_Msg.h

Description Adds the given ROI to the given viewport's ROI list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ROI_ADD

Description: Specific type of command message.

Name: CRoi

Description: Pointer to an ROI object derived from a CcRoiBase object.

Notes

A tool can create a ROI and then add this ROI to a viewport. The Blob tool uses this message to add ROIs to viewports. To add a newly created ROI to a viewport, send this message to the desired viewport with a pointer to the ROI in the *lParam* parameter of the message.

After a viewport adds the ROI to its list, the viewport notifies the tools using the notification message HLN_ROI_CREATED.

Notes (cont.)

Each viewport in the GLI/2 main application contains a list of ROIs. When you add an ROI to a viewport, you are adding the ROI to the viewport's list of ROIs. If you need to add many ROIs to this list, add the ROIs directly using the methods of the CcList object. Make sure that the last ROI is added to the list using this command message; this updates all tools and viewports. If you do not add the last ROI in this manner, the tools and the viewports are not updated. You can add all ROIs to the viewport's list using this command message, but this is slower than doing it directly. Thus, if you have ten new ROI objects to add to the list, add the first nine directly, and add the tenth ROI using this command message. This is how the Blob Analysis tool adds ROIs.

There are two modes of operation in the main application with respect to ROIs: the ROIs can be attached to the viewport or to the image itself. In either case, only one ROI list can be associated with a viewport at any given time. This message always adds the ROI to the correct ROI list and is transparent to which mode of operation the main application is in.

HLC_ROI_DELETE

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_ROI_DELETE,(LPARAM)CRoi);
```

Include File

DT_Msg.h

Description

Deletes the given ROI from the given viewport's ROI list.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ROI_DELETE

Description: Specific type of command message.

Name: CRoi

Description: Pointer to a ROI object derived from a CcRoiBase object.

Notes

A tool can create a ROI and then add this ROI to a viewport. Later, you may want the tool to delete this ROI from the viewport. It can do so using this message.

After a viewport deletes the ROI from its list, it notifies the tools using the notification message HLN_DELETING_ROI_OBJECT and the message HLN_DELETED_ROI_OBJECT.

Each viewport in the GLI/2 main application contains a list of ROIs. When you delete an ROI from a viewport, you are deleting the ROI from the viewport's list of ROIs. If you need to delete many ROIs from this list, do it directly using the methods of the CcList object.

Notes (cont.) Make sure that you delete the last ROI from the list using this command message; this updates all tools and viewports. If you do not delete the last ROI in this manner, the tools and the viewports are not updated. You can delete all ROIs from the viewport's list using this command message, but this is slower than doing it directly. Thus, if you have ten ROI objects to delete from the list, delete the first nine directly, and delete the tenth ROI by using this command message. This is how the Blob Analysis tool deletes ROIs.

There are two modes of operation in the main application with respect to ROIs: the ROIs can be attached to the viewport or to the image itself. In either case, only one ROI list can be associated with a viewport at any given time. This message always deletes the ROI from the correct ROI list and is transparent to which mode of operation the main application is in.

HLC_SEND_NAME_CHANGE_NOTIFICATION

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_SEND_NAME_CHANGE_
     NOTIFICATION,(LPARAM)CObject);
```

Include File DT_Msg.h

Description Instructs the main application to notify all tools that the name of the given object has changed.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SEND_NAME_CHANGE_ NOTIFICATION

Description: Specific type of command message.

Name: CObject

Description: Pointer to any GLI/2 derived object.

Notes If a tool changes the name of an object (such as the Memory Images tool), the tool must let the main application and other tools know about it. You do this by sending the main application this message with a pointer to the object whose name has changed, given in the *lParam* parameter of the message. The main application then notifies all tools of this using the notification message HLN_OBJECT_NAME_CHANGED.

HLC_SHOW_TOOL_BOX

Syntax

```
::SendMessage(
   hViewport,HL_COMMAND,
   HLC_SHOW_TOOL_BOX,(LPARAM)bFlag
   );
```

Include File DT_Msg.h

Description Show or hide the floating tool box.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SHOW_TOOL_BOX

Description: Specific type of command message.

Name: bFlag

Description: TRUE to show the tool box, FALSE to hide the tool box.

Notes In custom applications where you are using GLI/2 as your own application, it is sometimes desired to either show or hide the tool box from within the custom tool. You can hide/show the floating tool box using this message.

HLC_MANAGE_VIEWPORT

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_MANAGE_VIEWPORT,
   (LPARAM)bFlag);
```

Include File DT_Msg.h

Description Manages the given viewport with respect to hide, show, minimize, maximize, and restore.

Parameters

Name: hViewport

Description: Viewport that you want to control.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_MANAGE_VIEWPORT

Description: Specific type of command message.

Name: bFlag

Description: Flag to determine how to manage the viewport. It can be one of the following values:

- SW_HIDE – Hides the viewport and activates another viewport.
- SW_MAXIMIZE – Maximizes the specified viewport.
- SW_MINIMIZE – Minimizes the specified viewport.
- SW_RESTORE – Activates and displays the viewport. If the viewport is minimized or maximized, this restores it to its original size and position.
- SW_SHOW – Activates the viewport and displays it in its current size and position.

HLC_MANAGE_MAINAPP

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_MANAGE_MAINAPP,
   (LPARAM)bFlag);
```

Include File DT_Msg.h

Description Manages the GLI/2 main application with respect to hide, show, minimize, maximize, and restore.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_MANAGE_MAINAPP

Description: Specific type of command message.

Name: bFlag

Description: Flag to determine how to manage the viewport. It can be one of the following values:

- SW_HIDE – Hides the viewport and activates another viewport.
- SW_MAXIMIZE – Maximizes the specified viewport.
- SW_MINIMIZE – Minimizes the specified viewport.

Description (cont):

- SW_RESTORE – Activates and displays the viewport. If the viewport is minimized or maximized, this restores it to its original size and position.
- SW_SHOW – Activates the viewport and displays it in its current size and position.

HLC_POSITION_VIEWPORT

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_POSITION_VIEWPORT,
   (LPARAM)&stPOS);
```

Include File DT_Msg.h

Description Position and size the given viewport.

Parameters

Name: hViewport

Description: Viewport that you want to control.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_POSITION_VIEWPORT

Description: Specific type of command message.

Name: stPOS

Description: Windows RECT structure describing the new position and size of the viewport.

Notes This positions the viewport with respect to the main application's position, not with respect to the screen.

Example The following is an example of how to use this message:

```
void CcTool::OnPositionViewport( )
{
 RECT stPos;

 stPos.top = 10;
 stPos.bottom = 310;
 stPos.left = 10;
 stPos.right = 310;
::SendMessage(m_hActiveViewport,
   HL_COMMAND,
   HLC_POSITION_VIEWPORT,(LPARAM)&
   stPos);
}
```

HLC_POSITION_MAINAPP

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_POSITION_MAINAPP,
   (LPARAM)&stPOS);
```

Include File DT_Msg.h

Description Position and size the GLI/2 main application.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_POSITION_MAINAPP

Description: Specific type of command message.

Name: stPOS

Description: Windows RECT structure describing the new position and size of the viewport.

Notes This positions the viewport with respect to the main application's position, not with respect to the screen.

Example The following is an example of how to use this message:

```
void CcTool::OnPositionMainapp( )
{
RECT stPos;
stPos.top = 10;
stPos.bottom = 510;
stPos.left = 10;
stPos.right = 510;

::SendMessage(m_hActiveViewport,
   HL_COMMAND,
   HLC_POSITION_MAINAPP,
   (LPARAM)&stPos);
}
```

HLC_ARRANGE_VIEWPORTS

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_ARRANGE_VIEWPORTS,
   (LPARAM)iFlag);
```

Include File DT_Msg.h

Description Arranges all of the GLI/2 viewports. It can be tiled horizontally, tiled vertically, cascaded, or arranged.

Parameters

Name: hViewport

Description: Viewport to which you are sending the command message. This can be any valid viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ARRANGE_VIEWPORTS

Description: Specific type of command message.

Name: iFlag

Description: Flag to specify how to arrange the viewports. It can be one of the following:

- HLV_TILE_HORIZONTAL – Tile horizontally.
- HLV_TILE_VERTICAL – Tile vertically.
- HLV_CASCADE – Cascade.
- HLV_ARRANGE_ICONS – Arrange icons.

HLC_CLOSE_VIEWPORT

Syntax

```
::SendMessage(
   hViewport, HL_COMMAND,
   HLC_CLOSE_VIEWPORT,(LPARAM)0);
```

Include File DT_Msg.h

Description Closes the given viewport.

Parameters

Name: hViewport

Description: Viewport that you want to close.

Name: HL_COMMAND

Description: Required for all command messages

Name: HLC_CLOSE_VIEWPORT

Description: Specific type of command message.

Name: 0

Description: This message does not use the *lParam* parameter, thus place a 0 in this parameter.

Notes You can not close all the viewports in GLI/2. You must always have at least one viewport open. If you try to close the last viewport, the viewport is not closed.

HLC_ACTIVATE_VIEWPORT

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_ACTIVATE_VIEWPORT,
   (LPARAM)0);
```

Include File DT_Msg.h

Description Activates the given viewport.

Parameters

Name: hViewport

Description: Viewport that you want to activate.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ACTIVATE_VIEWPORT

Description: Specific type of command message.

Name: 0

Description: This message does not use the *lParam* parameter, thus place a 0 in this parameter.

HLC_ADD_LIST_TO_MAIN_LIST

Syntax

```
::SendMessage(hViewport,
   HL_COMMAND,
   HLC_ADD_LIST_TO_MAIN_LIST,
   (LPARAM) &CList;
```

Include File DT_Msg.h

Description Adds a user-defined list to the main object list.

Parameters

Name: hViewport

Description: The viewport that you are sending the command message to. It can be any viewport; it does not have to be the active viewport.

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_ADD_LIST_TO_MAIN_LIST

Description: Specific type of command message.

Name: CList

Description: The address of the list to add to the main list.

HLC_SEND_LIST_CHANGE_NOTIFICATION

Syntax

```
::SendMessage(m_hActiveViewport,
   HL_COMMAND,
   HLC_SEND_LIST_CHANGE_
     NOTIFICATION, (LPARAM)String);
```

Include File DT_Msg.h

Description Notifies the tools about a change in one of the lists that is managed by the main application.

Parameters

Name: m_hActiveViewport

Description: The active viewport that you are sending the command message to. I

Name: HL_COMMAND

Description: Required for all command messages.

Name: HLC_SEND_LIST_CHANGE_ NOTIFICATION

Description: Specific type of command message.

Name: String

Description: A character string which contains the name of the list (such as a number list) that has changed.

HLC_ADD_TO_SCRIPT_TOOLS

Syntax

```
::SendMessage(m_hActiveViewport,
   HL_COMMAND,
   HLC_ADD_TO_SCRIPT_TOOLS
   (LPARAM)&stScript);
```

Include File DT_Msg.h

Description Places a tool in the Point and Click Script tool.

Parameters

Name:	m_hActiveViewport
Description:	The active viewport that you are sending the command message to.
Name:	HL_COMMAND
Description:	Required for all command messages.
Name:	HLC_ADD_TO_SCRIPT_TOOLS
Description:	Specific type of command message.
Name:	stScript
Description:	The address of the point and click script structure for the specified tool.

Point and Click Script Messages

Point and click script messages are sent to a tool from the Point and Click Script tool to command or request some type of information.

All messages are sent from the Point and Click Script tool and are routed through the main application to all other tools. This is accomplished using the standard Windows function **SendMessage**. For more information on the **SendMessage( )** function, see the Windows SDK API documentation.

A point and click script message has the following form:

```
SendMessage(hTool,HL_SCRIPT, specific notification
message, message specific information);
```

Information about the event is often contained in the *lParam* parameter of the message. For further information on the contained information, see Chapter 2 on page 11.

All point and click script messages are processed in a tool by processing the HL_SCRIPT message sent by the main application. This message map is already set up to map to the **HLScript( )** message handler in the example change tool that is included in the GLI/2 package (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default). Thus, all notification messages should be processed in the switch statement of the **HLScript)** message handler. You can process none, all, or some of the notification messages in the switch statement. Which notification messages you process is determined by the desired functionality of your tool.

The following example shows starting code for this event handler and the point and click script messages HLN_RUN_SCRIPT and HLS_STEP_SCRIPT:

```
//**************** H L  SCRIPT **************//
LRESULT CcTool::HLScript(WPARAM wParam,
   LPARAM lParam)
{
/*Start of Dec Section*/
/*End  of Dec Section*/

switch(wParam)
{
   case HLS_RUN_SCRIPT:
   (process this message here)
   return(TRUE);
   break;
   case HLN_STEP_SCRIPT:
   (process this message here)
   return(TRUE);
   break;
}
   return(TRUE);
}
//************** H L  SCRIPT ****************//
```

Note: All GLI/2 point and click script messages start with the prefix: HLS_.

The point and click script messages are briefly described in Table 40.

Table 40: Point and Click Script Message s

Point and Click Script Messages	Description
HLS_RUN_SCRIPT	Commands the tool to run the script.
HLS_STEP_SCRIPT	Commands the tool to step through the script.
HLS_INITIALIZE_FOR_RUN	Commands the tool to initialize any components of the tool, such as allocating buffers, that could be reused when the script is run.
HLS_EDIT_SCRIPT	Notifies the tool that the user has pressed the Edit button of the Point and Click Script; the tool should respond appropriately.
HLS_UNINITIALIZE_FROM_RUN	Notifies the tool that the script has been stopped and commands the tool to destroy any data that was previously set up with HLS_INITIALIZE_FOR_RUN.
HLS_CANCEL_EDIT	Notifies the tool that the user has pressed the Cancel button of the Point and Click Script; the tool should respond appropriately.
HLS_SUPPLY_SCRIPT_STRUCT_SIZE	Requests the size of the script structure used by the specified tool. This message is used during upgrades of script structures to new versions.

Table 40: Point and Click Script Messages (cont.)

Point and Click Script Messages	Description
HLS_SUPPLY_SCRIPT_STRUCT _DEFAULTS	Requests the default values for the script elements that were added to a new tool.
HLS_CREATING_SCRIPT_ STRUCT	Notifies the tool that a memory block for a script structure has been created.
HLS_DELETING_SCRIPT_ STRUCT	Notifies the tool that a memory block for a script structure has been deleted.
HLS_CAN_TOOL_BE_PARENT	Requests whether the specified tool can be a parent.
HLS_CAN_BRANCH_TO_ CHILDREN	Requests whether the specified tool can execute the child tools.
HLS_BRANCH_TO_CHILDREN_ DONE	This message is issued when the child tools have completed execution.

HLS_RUN_SCRIPT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_RUN_SCRIPT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Commands the tool to execute the portion of the tool that is responsible for running the point and click script.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes None

Return Values

TRUE Successful.

FALSE Failed.

HLS_STEP_SCRIPT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_STEP_SCRIPT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Commands the tool to step through the point and click script.

Parameters

Name: STSCRIPT *

Description: A pointer to the point and click script.

Notes None

Return Values

TRUE Successful.

FALSE Failed.

HLS_INITIALIZE_FOR_RUN

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_INITIALIZE_FOR_RUN:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Commands the tool to initialize the components of a tool (such as allocating buffers) that can be reuesd when the script is run.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes This message is sent before the HLS_RUN_SCRIPT message.

Return Values

TRUE Successful.

FALSE Failed.

HLS_EDIT_SCRIPT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_EDIT_SCRIPT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Notifies the tool that the user pressed the Edit button of the Point and Click Script tool; the tool should then handle this message appropriately.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes None

Return Values

TRUE Successful.

FALSE Failed.

HLS_UNINITIALIZE_FOR_RUN

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_UNINITIALIZE_FOR_RUN:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Notifies the tool that the point and click script stopped running.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes Once it receives this message, the tool can destroy any data that was set up when it received the HLS_INITIALIZE_FOR_RUN message.

Return Values

TRUE Successful.

FALSE Failed.

HLS_CANCEL_EDIT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_CANCEL_EDIT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Notifies the tool that the user pressed the Cancel button of the Point and Click Script tool; the tool should then handle this message appropriately.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes None

Return Values

TRUE Successful.

FALSE Failed.

HLS_SUPPLY_SCRIPT_STRUCT_SIZE

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_SUPPLY_SCRIPT_STRUCT_
   SIZE:
return(sizeof(STSCRIPT));
break;
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Requests the size of the script structure that is used by a specified tool.

Parameters None

Notes

This message along with HLS_SUPPLY_SCRIPT_STRUCT_DEFAULTS provides a mechanism for upgrading script structures.

For example, assume that you modified a tool by extending its capabilities; in the process, you were forced to expand the script structure. To be able to run a script that was created with the original, unmodified tool, you need to use the HLS_SUPPLY_SCRIPT_STRUCT_SIZE message. When it receives the HLS_SUPPLY_SCRIPT_STRUCT_SIZE message, the tool must respond with a proper value (such as sizeof(my_tools_structure)). The point and click script can then allocate the proper size memory block and fill it with the values from the previously recorded script. When the HLS_SUPPLY_SCRIPT_STRUCT_DEFAULTS message is received, the new fields can then be initialized.

Return Values

int — Size of the tool's point and click script structure.

HLS_SUPPLY_SCRIPT_STRUCT_DEFAULTS

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_SUPPLY_SCRIPT_STRUCT_
   DEFAULTS:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Requests the default values for the script elements that were added to a new tool.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes This message is sent only if the script structure size that was recorded by the Point and Click Script tool differs from the one that was supplied in response to the HLS_SUPPLY_SCRIPT_STRUCT_SIZE message.

Notes (cont.)

This message along with HLS_SUPPLY_SCRIPT_STRUCT_SIZE provides a mechanism for upgrading script structures.

For example, assume that you modified a tool by extending its capabilities; this automatically expanded the script structure. To be able to run a script that was created with the original, unmodified tool, you need to use the HLS_SUPPLY_SCRIPT_STRUCT_SIZE message. When it receives the HLS_SUPPLY_SCRIPT_STRUCT_SIZE message, the tool must respond with a proper value (such as sizeof(my_tools_structure)). The point and click script can then allocate the proper size memory block and fill it with the values from the previously recorded script. When the HLS_SUPPLY_SCRIPT_STRUCT_DEFAULTS message is received, the new fields can then be initialized.

Return Values

TRUE	Successful.
FALSE	Failed.

HLS_CREATING_SCRIPT_STRUCT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_CREATING_SCRIPT_STRUCT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File

DT_Msg.h

Description

This message is sent when a point and click script internally creates a memory block for a script structure of specified tool.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes

A memory block is created only when a script is loaded from disk or when a tool is added to a point and click script.

When it receives this message, the tool can initialize any internal structures that require initialization (such as instantiating any class needed by the tool).

Return Values

TRUE	Successful.
FALSE	Failed.

HLS_DELETING_SCRIPT_STRUCT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_DELETING_SCRIPT_STRUCT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description This message is sent when a point and click script internally deletes a memory block for a script structure of specified tool.

Parameters

Name:	STSCRIPT *
Description:	A pointer to the point and click script.

Notes This message provides a mechanism for deleting anything that was initialized when the HLS_CREATING_SCRIPT_STRUCT message was received.

Return Values

TRUE	Successful.
FALSE	Failed.

HLS_CAN_TOOL_BE_PARENT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_CAN_TOOL_BE_PARENT:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE); //if can be parent
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Requests whether a tool can invoke other tools to perform additional processing when the point and click script is run.

Parameters None

Notes If a tool responds with TRUE to this message, the Point and Click Script tool allows new "child" tools to be added underneath this "parent" tool. The Point and Click Script tool branches to the child tools in response to the HLS_CAN_BRANCH_TO_CHILDREN message.

Notes (cont.) This message is used to handle asynchronous acquires by the Picture tool, but can be used by any tool that needs to perform additional processing under special circumstances.

Return Values

BOOL If TRUE, the tool can be a parent; if FALSE, the tool cannot be a parent.

FALSE Failed.

HLS_CAN_BRANCH_TO_CHILDREN

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_CAN_BRANCH_TO_CHILDREN:
return(TRUE);
//if branching is desired
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description Notifies the point and click script whether or not to execute its child tools.

Parameters

Name: STSCRIPT *

Description: A pointer to the tool-specific point and click script structure.

Notes None

Return Values

TRUE	Executes child tools.
FALSE	Does not execute child tools.

HLS_BRANCH_TO_CHILDREN_DONE

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcTool::HLScript(WPARAM
   wParam,LPARAM lParam)
{
switch(wParam)
{
case HLS_BRANCH_TO_CHILDREN_DONE:
STSCRIPT* stScriptStruct =
   (STSCRIPT*)lParam;
(process message accordingly...)
return(TRUE);
}
//***** H L SCRIPT ********//
```

Include File DT_Msg.h

Description This message is issued when the child tools have completed execution.

Parameters

Name:	STSCRIPT *
Description:	A pointer to the tool-specific point and click script structure.

Notes None

Return Values

TRUE	Successful.
FALSE	Failed.

Example Tool Implementation

This section shows how to create, install, and run a custom tool that is based on an example change tool that is included in the GLI/2 package (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default). This tool sets all of the pixels in an active viewport to a user-defined value with respect to the viewport's active ROI. The image used as the input image can be any type of image, and the ROI used as the active ROI can also be of any type.

Note: This example change tool does not include point & click scripting. If you want to include point & click scripting in your custom tool, refer to the Visual C++ project for the example change tool, located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

This example consists of the following main tasks, which are described in the following subsections:

- Create a base tool.
- Register the tool with GLI/2.
- Customize the look of the tool.
- Add functionality using the command and request messages.
- Add functionality using the notification messages.
- Separate the tool into separate modules.

Creating a Base Tool

First, create a base tool that has no functionality. You can easily accomplish this task by using the example change tool that is included in the GLI/2 package (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default). The example change tool is provided with all the necessary code and a workspace that together serve as a starting place for all GLI/2 tools. All GLI/2 tools were created using the example change tool.

Note: If you are building your custom tool in release mode, you need to link it with the release DTAPI.LIB and run it with the release version of GLI/2 and the release versions of all the tools.

If you are building your custom tool in debug mode, you need to link it with the debug DTAPID.LIB, and run it with the debug version of GLI/2 and the debug versions of all the tools.

Both versions of each tool are supplied and are located in the same directory. The release version of a tool does not have a prefix and the debug version of the same tool starts with the prefix *D_* (where *D* stands for debug).

The workspace (for both versions) is named DT_TOOL.DSW and the project workspace file is named DT_TOOL.DSW.

Do not intermix debug versions with release versions.

To create your own custom tool, perform the following procedure:

1. Start Visual C++ for Windows 98, Windows 2000 , or Windows ME, and load the example change tool's project workspace file from within the Visual Workbench. As with all tools, the name of this file is DT_TOOL.DSW; it is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default).

   Note: If you need more help, refer to your Visual C/C++ documentation.

2. If you did not install the GLI/2 application using the install program, change the include path for all GLI/2 include files to the following directory: GLI\GLI\Include.

   Note: If you used the install program, the default path of ..\..\include should work, regardless of where you installed the application.

3. Check to make sure that the files in the project are correct. The following is a list of the files that are contained in the project:
   - DT_Tool.CPP – Tool's DLL module.*
   - DT_Tool.DEF – Tool's DLL definition file.*
   - DT_Tool.RC – Tool's resource file.
   - d_cTool.CPP – Tool's dialog box procedure.
   - DTBaseTL.CPP – Base class GLI/2 tool file.*
   - stdafx.CPP – MFC standard project file.*

Note: Do not rename any of the above files.

The files marked with * above should never need to be modified. They are supplied only for advanced Windows programmers (for those who may wish to further understand how a tool attaches to the main application).

4. From the Visual C++ tool bar, click **Rebuild All** to compile and build the tool.

Registering a Tool with GLI/2

After building the tool, you need to register the tool with the GLI/2 main application. An initialization file named DTTOOLS.INI is located in the same directory (C:\GLI\GLI, by default) as the GLI executable (GLI2.EXE). You must edit this file using the Data Translation INI Editor (from the Windows Start menu, select **Programs\Data Translation, Inc\GLOBAL LAB Image2\GLOBAL LAB Image2**).

To add a new tool, select **New Tool Entry** from the **Edit** menu. Double-click **LOCATION**, browse to the location of the new tool .DLL file, then click **Open**. Double-click **AUTOSTART** and specify whether you want GLI/2 to automatically start the tool at program startup. If desired, double-click **DESCRIPTION** and enter a brief description of the tool.

The following is an example of the format of a tool entry in the DTTOOLS.INI file:

```
[TOOL1]
LOCATION=D:\GLI\GLI\TOOLS\DTARITH\DTARITH.DLL
```

To add a new tool, add the following two lines to the DTTOOLS.INI file:

```
[TOOL#]
LOCATION=Path
```

where # is the unique sequential number given to the tool and *Path* is the complete path to the tool's .DLL file. Note that the tool numbers must be unique and sequential starting from 1. If you want GLI/2 to automatically start the tool at program startup, you can place the line `AUTOSTART=YES` in the tool entry.

For example, to start tool number 1 automatically when GLI/2 starts, enter the following for tool 1:

```
[TOOL1]
LOCATION=D:\GLI\GLI\TOOLS\DTARITH\DTARITH.DLL
AUTOSTART=YES
```

Note: If you are compiling a debug version of your custom tool, you need to run the debug version of GLI/2 and edit its associated DTTOOLS.INI file. These versions are located in the directory (C:\GLI\GLI\DEBUGVER\).

In this example, the tool is named START.DLL (D_START.DLL for the debug version). START.DLL is located in the following directory:

```
C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL\RELEASE
```

or

```
C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL\DEBUG
```
(for the debug version)

To register the tool, add the following two lines to the end of the DTTOOLS.INI file:

```
[TOOL19]
LOCATION=C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE
   TOOL\RELEASE\START.DLL
```

If your tool was named MYTOOL.DLL and it was located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL\MYSTUFF, and you had 20 other tools in the system, you would add the following two lines to the end of the DTTOOLS.INI file:

```
[TOOL21]
LOCATION=C:\GLI\GLI\DEVELOPMENT EXAMPLES\
   MYSTUFF\MYTOOL.DLL
```

After saving the DTTOOLS.INI file, you can start GLI/2. Your new tool should appear in both the tool's menu and tool box.

For further information on this subject, see Chapter 2 starting on page 11.

Customizing the Look of Your Tool

Now that you have your custom tool up and running with GLI/2, you need to customize the tool for your application by changing the tool's name, bitmaps, icon, and help file. You do this through the graphical interface of the integrated resource editor of Visual C++. No programming is required.

Note: Do not rename the workspace or any of the files in the workspace. Do not rename or change any ID's in any of the files or the RC file.

Editing the String Table in the RC File

To change the name that appears in the GLI/2 tool menu, the name of the help file, or the number of instances that the tool can have, edit the string table of the DT_Tool.RC file.

The string associated with the ID DT_TOOL_MENU_TEXT is the tool's name that appears in the tools menu in the main application. Change this text to the name you desire.

The string associated with the ID DT_TOOL_NUM_OF_INST is the number of instances that can be created for this tool. In other words, it is the number of tools that can be open at the same time for this type of tool. The maximum number is 100 and the minimum number is 1.

The string associated with the ID DT_TOOL_HELP_FILE is the name of the help file associated with this tool. If you have no help file, enter NONE in this field. The help file must be placed in the same directory as the tool, so you do not place the full path name in this field. Enter the base name of the help file which includes the file name and the .HLP extension (for example: MYTOOL.HLP).

Editing the Bitmaps and Icon in the RC File

Next, you need to change the tool's icons. These are the icons that appear in the tool box and when the tool is iconized. The icons that appear in the tool box are the bitmaps in the RC file named "Pressed" and "Unpressed." They represent how the icons in the tool box appear when the buttons are selected (pressed) and unselected (unpressed). The icon named IDI_TOOL is the icon given to the tool when it is iconized.

When editing icons, change only the area inside the black rectangle. so that all GLI/2 icons look the same way. The icon and the unpressed bitmap are the same in most cases, so when you like how the icon looks, you can cut and paste the image into another icon.

Editing the Dialog Box in the RC File

The tool itself is a dialog box. The dialog template for the tool is the IDD_TOOL dialog. Set the caption of the dialog box to the same name that you gave the tool in the DT_TOOL_MENU_TEXT ID in the string table. This is a guideline for creating a tool because it simplifies operation for the operator.

An example RC file with these changes is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

Note: In Visual C++, you can have more than one RC file open at a time; therefore, you can cut and paste code from one file to another.

You are now done editing the RC file. Save the RC file and rebuild the tool. Since the tool is already registered with GLI/2, you can now run GLI/2 and see your changes.

Note: In the sample RC file, the color of the depressed bitmap is changed. Do not do this in your own tool.

Adding Functionality Using Command and Request Messages

At this point you should have your own custom tool with its own name, help file, and custom icons up and running with GLI/2.

This part of the program adds an edit control to the tool so that an operator can enter a value into it. Then, it adds a button, which when clicked, changes all the pixel values in the image in the active viewport to the value in the edit control with respect to the active ROI.

Note: It is assumed that you know how to add a button and an edit box to the dialog box using Visual C++. If you need more information on this, refer to your Visual C++ documentation.

The code for the dialog box procedure is in the modules c_CTool.CPP and d_CTool.H (d_ stands for dialog box procedure and c stands for class).

Note: The code for this step is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL. The code has error checking and variable declaration removed to simplify the code and amplify the main idea of how to use GLI/2 messaging. All added code for this section of the program has the comment //STEP2 above it. To quickly see all the changes needed for this section of code, search for this comment.

The procedure for the button click is as follows:

```
void CcTool::OnOk()
{
CcImage* CImage;
CcRoiBase* CRoi;
```

The steps required to implement this procedure are as follows:

1. Obtain a handle to the active viewport by sending the GLI/2 main application a message that asks for the active viewport's handle. If you do not receive a valid handle, abort the program. The following code gets the handle of the active viewport:

   ```
   //Get Handle to Active Viewport
   m_hActiveViewport=(HWND)::SendMessage(
   m_hMainApplication,HL_GET_ACTIVE_VIEWPORT,0,0L);
   ```

2. Using the handle to the active viewport, obtain a pointer to the image by sending the request message HLR_SUPPLY_IMAGE_OBJECT, and obtain a pointer to the active ROI associated with the viewport by sending the request message HLR_SUPPLY_ACTIVE_ROI_OBJECT. If either pointer is not valid, abort the program.
   You must cast the pointer to the type of object you are expecting. In the case of both the image and the ROI, cast the pointers to their base class pointers. Do not worry about the *type* of image or the *type* of ROI since both of these objects contain virtual methods where needed. For more information, see the ROI and Image classes in Chapter 2 starting on page 11.

   The following code illustrates how to get a pointer to the image and to the ROI associated with the viewport:

   ```
   //Send a message to the Active Viewport to
   //Get its Image Pointer
   CImage = (CcImage*)
   ::SendMessage(hActiveViewport,
      HL_REQUEST,HLR_SUPPLY_IMAGE_OBJECT,0L);
   //Get a pointer to the active ROI within the
   ```

```
//active viewport
CRoi=(CcRoiBase*)
::SendMessage(hActiveViewport,
   HL_REQUEST,HLR_SUPPLY_ACTIVE_ROI_OBJECT,0L);
```

3. Using the image and ROI pointers, change the data in the image to the new value with respect to the ROI by obtaining the x- and y-coordinates for each point in the ROI, and then set the image at these points to the new value:

```
//First get the location of the bounding
//rectangle for the given ROI (without knowing
//its type)

pstROI =(RECT*)CRoi->GetBoundingRect( );

//Then go from the bottom of the ROI to its top,
//getting the location of all pixels in each
//horizontal row

for(y=pstROI->bottom; y<pstROI->top; y++)
   {
   //Get pointer to array of x-locations of each
   //pixel in this horizontal row for the given
   //y-location
   piRoiData=CRoi->GetXBoundary(y,
   &iNumOfROIPoints);

   //Extract x-location for each point from
   //array

      for(z=0; z<iNumOfROIPoints; z++)
      {x=piRoiData[z];

      //Using X- and Y-location for each point,
      //change image value
      //First set data image pointer within
      //image class to point to correct position
```

```
            Image(x,y);

            //Then set the new value for the pixel at
            //this location. This is the value you
            //extracted from the edit box that the
            //user entered
            Image=fNewValue;

            }
         }
```


4. Now that the image data has changed, redraw the image in the viewport by sending the active viewport the command message HLC_REDRAW_VIEW:

```
::SendMessage(hActiveViewport,
   HL_COMMAND,HLC_REDRAW_VIEW,0L);
}
```

5. After adding this code to the dialog box procedure module (d_CTool.cpp and d_CTool.h), rebuild the tool and test it by running GLI/2.

Adding Functionality Using Notification Messages

Notification messages are sent from the main application to the tools when something significant happens in the main application. For example, when the user activates a viewport, the main application informs all open tools. In this example, the notification message is processed and the name of the image is placed in the newly active viewport in the button of the tool we just created. This section of the code does not add any image processing functionality; it provides a simple demonstration of how to use GLI/2 notification messages.

Note: The code for this section of the program is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default. All added code for this section of the program has the comment //STEP3 above it. To quickly see all the changes needed for this section of the program, search for this comment.

When a viewport becomes active, the HLN_VIEWPORT_ACTIVATED notification message is sent. To use notification messages, edit the following section of code in the example change tool:

```
//*********************************************//
//*************** H L  N O T I F Y *************//
//*********************************************//
LRESULT CcTool::HLNotify(WPARAM,LPARAM)
{
/*Start of Dec Section*/
/*End  of Dec Section*/
switch(wParam)
  {
  case HLN_xxxxxxxx:
   return(TRUE);
   break;
   }
```

```
return(TRUE);
}
//***********************************************//
//************* H L  N O T I F Y ***************//
//***********************************************//
```

To use the HLN_VIEWPORT_ACTIVATED message, change this code to the following:

```
//*************** H L  N O T I F Y
*******************//
LRESULT CcTool::HLNotify(WPARAM wParam,
   LPARAM lParam)
{
/*Start of Dec Section*/
 char* cName;
 CcImage* CImage;
 CButton* CButton1;
/*End  of Dec Section*/

switch(wParam)
  {
case HLN_VIEWPORT_ACTIVATED:
   //Get pointer to button object
    CButton1 = (CButton*)GetDlgItem(ID_OK);
    if(CButton1==NULL) return(TRUE);
   //Cast lParam to the active viewport given with
   //message
    hActiveViewport = (HWND)lParam;
   //Send a request message to the active viewport
   //requesting its associated image
 CImage = (CcImage*)
 ::SendMessage(hActiveViewport,
      HL_REQUEST,HLR_SUPPLY_IMAGE_OBJECT,0);
    if(CImage==NULL)
    {CButton1->SetWindowText("");
     return(TRUE);}
```

```
//Get name of image from image object and set this
//text to the button
   cName = CImage->GetName( );
   CButton1->SetWindowText(cName);
return(TRUE);
break;
      }
return(TRUE);
}
//************ H L  N O T I F Y ******************//
```

For the message HLN_VIEWPORT_ACTIVATED, the handle to the newly activated viewport is given in *lParam*. For more information on this parameter, refer to the HLN_VIEWPORT_ACTIVATED message on page 721. Cast this value to *HWND* to obtain a usable handle to the active viewport. Using this handle, you can send a request message to the active viewport to request its Image object. Once you have the Image object, ask for its name by calling the method **GetName( )**. Then, place the name of the image into the button.

All notification messages get routed to all tools using the **HLNotify( )** method. You do not (and should not) need to route any notification messages yourself. You can place as many notification messages in the above switch statement as you like. All notification messages are processed the same way as this one.

After adding this code, rebuild your tool and run GLI/2. Each time you click a new viewport with an image in it, you should see the name of the image in the button on your tool.

Note: The variable *hActiveViewport* is declared in the above code. You also could have used *m_hActiveViewport*. *m_hActiveViewport* is provided for your convenience but is not required.

Separating the Tool into Modules

If the new functionality of your tool is contained within the dialog box procedure module, the functionality cannot be used by other tools or other image processing applications. If, however, you separate the image processing functionality into its own module, any application or tool can use it.

Note: The code for this section of the program is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default. All added code for this section of the program has the comment //STEP4 above it. To quickly see all the changes needed for this section of the program, search for this comment.

Add the module c_ezchg.cpp to your project. This module is a scaled-down version of the c_change.cpp module used by the example change tool that is provided in the GLI/2 package (located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default).

The following code is responsible for the change functionality in the current example tool:

```
pstROI =(RECT*)CRoi->GetBoundingRect( );
for(y=(int)pstROI->bottom; y<(int)pstROI->top; y++)
   {

piRoiData=CRoi->GetXBoundary(y,&iNumOfROIPoints);
   for(z=0; z<iNumOfROIPoints; z++)
   {
   x=piRoiData[z];
   Image(x,y);
   Image=fNewValue;
   }
}
```

To separate the functionality of the tool from the user interface, place the code that performs the change operation in a public method of a class. The parameters for the method of the class are the objects that are used in the calculations. For example, we could use the following method of the CcChange class (with error checking and variable declaration removed):

```
int CcChange::Change(CcImage* Cimage,
   CcRoiBase* Croi,float fNewValue)
{
CcImage &Image = CImage;
pstROI =(RECT*)CRoi->GetBoundingRect( );
for(y=(int)pstROI->bottom; y<(int)pstROI->top; y++)
{
piRoiData=CRoi->GetXBoundary(y,&iNumOfROIPoints);
for(z=0; z<iNumOfROIPoints; z++)
{x=piRoiData[z];
 Image(x,y);
 Image=fNewValue;}
}
return(0);
}
```

Now any tool or application can use the change functionality by using this class.

All the code necessary to rebuild the example change tool is provided in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default. Not only does this code give you an example of how to create your own tool and separate a tool into modules, it gives you an example of how to speed up the execution of your tool's functionality.

Note: The example change tool does not include point & click scripting. If you want to include point & click scripting in your custom tool, refer to the Visual C++ project for the example change tool, located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.

Speeding Up the Execution of a Tool

You can use an image processing software package to either derive image processing algorithms or execute already known and established algorithms. This section explores these two uses in more detail.

Deriving Algorithms with GLI/2

When deriving image processing algorithms, it is nice to work in a simple environment. This lets you concentrate on the algorithm and not on computer programming. It is also important to try different image types when deriving algorithms, such as 8-bit grayscale, 32-bit grayscale, floating-point grayscale, or 24-bit RGB color images. Again, this lets you concentrate on the algorithm, and not on scaling, overflow, and data loss due to the limitations of the variable type holding the pixel values. GLI/2 provides such an environment through its image classes.

For example, assume that you want to derive a convolution image processing algorithm. Also assume that you want to create a method that takes an input image, an output image, and a rectangular ROI and calculates the summation of the center pixel and all of its four-connected neighbors for each point in the ROI. The calculated sum is then placed in the output image in the same location as the center point of the input image.

Without worrying about image types, you could write the following method:

```
void EZSum(CcImage* CImageIn, CcImage* CImageOut,
   RECT* stROI)
{
int x,y;
CcImage& ImageIn = *CImageIn;
CcImage& ImageOut = *CImageOut;
```

```
for(y=stROI->bottom; y<stROI->top; y++)
for(x=stROI->left; x<stROI->right; x++)
{
ImageOut(x,y);

ImageOut
= ImageIn(x,y)
+ ImageIn(x-1,y) + ImageIn(x+1,y)
+ ImageIn(x,y-1) + ImageIn(x,y+1);
}
}
```


Note: X,Y represents the center pixel coordinates relative to the lower-left corner (0,0) of the image.

After writing the method, you can then call the method from a created tool. You can use the other tools to analyze and view the image data produced by this method. For example, you could use the Memory Images tool to test this method with different types of images to see their effects. Remember, the above code works on 8-bit, 32-bit, and floating-point grayscale images as well as 24-bit RGB color images. It also works with your own image types, if you need to derive one.

Executing Algorithms with GLI/2

After you have derived an algorithm, you will most likely want to run it repeatedly. If you are running a imaging application on a manufacturing line or controlling a real-time process, you may need the algorithm to run extremely fast.

The highlights of the algorithm that you just wrote are as follows:

- It works with any image type.
- It is easy to read and understand.
- It is easy to debug.
- It has built-in error checking so you cannot crash the system.

However, it does not execute as fast as if you were to access the image data directly using pointers.

For some applications, the algorithm will execute fast enough; for others, however, the code will execute too slowly. GLI/2 provides the mechanisms to easily access the image data directly. All Image objects contain a method that return a pointer to their image data. They also have a method that lets you know what type of image you are dealing with. For example, **GetBitMapImageData( )** returns a pointer to the image data, and **GetImageType( )** returns the type of image you are dealing with. You can use these and other methods to speed up your algorithms by accessing the image data directly.

In the following code example (method **FastSum**), the algorithm is rewritten to speed up its execution. First, the pointers to each pixel that is accessed are calculated. Using the pointers, the pixels are then summed. For each consecutive center point along the horizontal row, all the pointers are incremented by 1; the pixels are then summed again. This process then repeats.

The pixels are arranged in the standard convolution format, shown in Table 41, (where 5 is the center pixel).

Table 41: Standard Convolution Format

7	8	9
4	5	6
1	2	3

Note: You lose the ability to handle all image types with the same code, so this example is for 32-bit input images only. Because no pointers are used for the output image, you can use any type of image. To further speed up the calculation, you can use pointers for the output image.

The example change tool makes use of both of these methods; all code for the example change tool is located in C:\GLI\GLI\DEVELOPMENT EXAMPLES\CHANGE TOOL, by default.


```
void FastSum(CcGrayImageInt32* CImageIn,CcImage*
   CImageOut,RECT* stROI)
{
int  x,y;
int iDibHeight,iDibWidth;

long lIndex;
int *InputData;
int *p5;//Center pixel
int *p2,*p4,*p6,*p8;  //4 connected neighbors
int *pEnd;

//Check Input Image's type
if(CImageIn->GetImageType( ) != IMAGE_TYPE_32BIT_GS
)
return(-1);

//Get Height&Width of Input Image
CImageIn->GetHeightWidth(&iDibHeight,&iDibWidth);

//Get Pointer to Input Image Data
InputData = (int*) CImageIn->GetBitMapImageData( );

//Go through ROI from Bottom to Top
```

```
for(y=stROI->bottom; y<stROI->top; y++)
{
//Assign Pointers for Input Image for 3x3 Kernel
lIndex = iDibWidth*y + stROI->left;

p5=&InputData[lIndex];
p4=p5-1; p6=p5+1;
p2=p5 - iDibWidth;
p8=p5 + iDibWidth;

lIndex = iDibWidth*y + stROI->right;
pEnd = &InputData[lIndex];

x=stROI->left;

//Sum points along this horizontal row
while(p5 < pEnd)
{
ImageOut(x++,y);
ImageOut =
*p2++ + *p4++ + *p5++ + *p6++ + *p8++;
}
}

//Tell Image to Rescale its data when showing
CImageOut->ReScaleImageOnShow( );
}
```

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X

Y

Data Translation Support Policy

Data Tr**an**slation, Inc. (Data Translation) offers support upon the following terms and conditions at prices published by Data Translation from time to time. Current price information is available from Data Translation, or its authorized distributor. If Licensee elects to obtain support services from Data Translation, Licensee must complete the Support Order Form attached hereto and submit to Data Translation the completed form, along with Licensee's purchase order for support. Support will only be provided for all (not less than all) Licensed Processors (as defined in the Data Translation Software License Agreement).

1. DEFINITIONS. Capitalized terms used herein and not otherwise defined shall have the meanings assigned thereto in the applicable Data Translation Software License Agreement (the Agreement). The following terms have the meanings set forth below:

Enhanced Release means a new release of any Product that contains new features and may contain corrections to previously identified errors. Enhanced Releases are designated in the tenths digit of the release designation (e.g., 1.2 is an Enhanced Release from 1.1.x).

Maintenance Release means a new release of any Product that contains corrections to previously identified errors. Maintenance Releases are designated in the hundredths digit of the release designation (e.g., 1.2.2 is a Maintenance Release from 1.2.1).

Major Release means a new version of any Product that involves major feature changes. Major Releases are designated in the ones digit of the release designation (e.g., 2.0, 3.0, etc., are Major Releases).

2. DATA TRANSLATION'S OBLIGATIONS. Subject to the terms of the Agreement, and this Support Policy, Data Translation will provide the following support services (Support Services) for the Products comprising the Software, as they may be used with the Licensed Processors:

(a) problem reporting, tracing and monitoring by internet electronic mail; (b) telephone support for problem determination, verification and resolution (or instruction as to work-around, as applicable) on a call-back basis during Data Translation's normal weekday business hours of 8:30 a.m. to 5 p.m. Eastern Time, excluding holidays; (c) one (1) copy of each Maintenance Release for the Products comprising the Software; (d) commercially reasonable efforts to diagnose and resolve defects and errors in the Software and Documentation; and (e) furnishing of the maintenance and technical support described above, for the current release and the immediately previous Enhanced Release of the Software. Support Services will be delivered in English. Enhanced Releases and Major Releases can be purchased by Licensee at a discount of twenty five percent (25%) off the then-current list prices for such releases.

3. EXCLUSIONS. Support Services do not include: (a) the provision of or support for Products other than those identified in the Agreement as to which the applicable license and support fees shall have been paid, including without limitation, compilers, debuggers, linkers or other third party software or hardware tools or components used in conjunction with any Product; (b) services required as a result of neglect, misuse, accident, relocation or improper operation of any Product or component thereof, or the failure to maintain proper operating and environmental conditions; (c) support for processors other than Licensed Processors or for Products modified by or on behalf of Licensee; (d) repair or restoration of any Software arising from or caused by any casualty, act of God, riot, war, failure or interruption of any electrical power, air conditioning, telephone or communication line or any other like cause.

Data Translation Support Policy

It is Licensee's responsibility to have adequate knowledge and proficiency with the use of the compilers and various software languages and operating systems used with the Products, and this Support Policy does not cover training of, or detailed direction on the correct use of these compilers, operating systems, or components thereof. On-site assistance shall not be provided hereunder, but may be available on a per call basis at Data Translation's then current rates (Specialized Application Support Charges) for labor, travel time, transportation, subsistence and materials during normal business hours, excluding holidays observed by Data Translation. The troubleshooting of faulty Licensee programming logic may also be subject to Specialized Application Support Charges and is not covered under this Support Policy. Direct authoring or development of customized application code is not provided hereunder but may be available on a per call basis upon payment of Specialized Application Support Charges.

4. LICENSEE'S OBLIGATIONS. Licensee agrees: (a) that the Designated Contact persons identified on the Support Order Form (or such other replacement individuals as Licensee may designate in writing to Data Translation) shall be the sole contacts for the coordination and receipt of the Support Services set forth in Section 2 of this Support Policy; (b) to maintain for the term of the support, an internet address for electronic mail communications with Data Translation; (c) to provide reasonable supporting data (including written descriptions of problems, as requested by Data Translation) and to aid in the identification of reported problems; (d) to install and treat all software releases delivered under this Support Policy as Software in accordance with the terms of the Agreement; and (e) to maintain the Agreement in force and effect.

5. TERM AND TERMINATION.

5.1 Term. For each Product comprising the Software, Support Services will begin on the later of the date the Software warranty granted in the Agreement expires or the date of Licensee's election to obtain Support Services and will apply to such Product for an initial term of one (1) year, unless an alternative commencement date is identified in the Support Order Form. The initial term will automatically be extended for additional terms of one (1) year unless Support Services are terminated at the expiration of the initial term or any additional term, by either party upon thirty (30) days prior written notice to the other party.

5.2 Default. If Licensee is in default of its obligations under the Agreement (except for Licensee's obligation to maintain valid licenses for the Software, in which case termination is immediate) and such default continues for thirty (30) days following receipt of written notice from Data Translation, Data Translation may, in addition to any other remedies it may have, terminate the Support Services.

6. CHARGES, TAXES AND PAYMENTS.

6.1 Payment. The Support Fee in respect of the initial term, and, as adjusted pursuant to Section 5.2 in respect of additional terms, is payable in full prior to the commencement of the initial term or any additional term, as applicable.

6.2 Changes From Term to Term. The Support Fee and the terms and conditions of this Support Policy may be subject to change effective at the end of the initial term or any additional term by giving Licensee at least sixty (60) days prior written notice.

Data Translation Support Policy

6.3 Taxes. The charges specified in this Support Policy are exclusive of taxes. Licensee will pay, or reimburse Data Translation, for all taxes imposed on Licensee or Data Translation arising out of this Support Policy except for any income tax imposed on Data Translation by a governmental entity. Such charges shall be grossed-up for any withholding tax imposed on Data Translation by a foreign governmental entity.

6.4 Additional Charges. Licensee agrees that Data Translation or its authorized distributor will have the right to charge in accordance with Data Translation's then-current policies for any services resulting from (a) Licensee's modification of the Software, (b) Licensee's failure to utilize the then-current release, or the immediately previous Enhanced Release, of the Software, (c) Licensee's failure to maintain Data Translation Support Services throughout the term of the Agreement, (d) problems, errors or inquiries relating to computer hardware or software other than the Software, or (e) problems, errors or inquiries resulting from the misuse or damage or of the Software or from the combination of the Software with other programming or equipment to the extent such combination has not been authorized by Data Translation. Pursuant to Section 2.4 of the Agreement, the Support Fee will also be adjusted in accordance with Data Translation's then current fee schedule as additional Licensed Processors are added. Support Fees do not include travel and living expenses or expenses for installation, training, file conversion costs, optional products and services, directories, shipping charges or the cost of any recommended hardware, third party software, or third party software maintenance fees or operating system upgrade.

7. WARRANTY LIMITATION. EXCEPT AS EXPRESSLY STATED IN THIS SUPPORT POLICY, THERE ARE NO EXPRESS OR IMPLIED WARRANTIES WITH RESPECT TO THE SUPPORT SERVICES PROVIDED HEREUNDER (INCLUDING THE FIXING OF ERRORS THAT MAY BE CONTAINED IN THE APPLICABLE DATA TRANSLATION SOFTWARE), INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE WARRANTIES AND REMEDIES SET FORTH IN THIS SUPPORT POLICY ARE EXCLUSIVE, AND ARE IN LIEU OF ALL OTHER WARRANTIES WHETHER ORAL OR WRITTEN, EXPRESS OR IMPLIED.

8. GENERAL PROVISIONS. Upon the election by Licensee to obtain Support Services, the terms of this Support Policy shall be governed by and are made a part of the Agreement.