DATA TRANSLATION

UM-17757-C

DT Vision Foundry API Manual (for Windows)

Third Edition
September, 2002

Copyright © 2000, 2001, 2002 by Data Translation, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording, or otherwise, without the prior written permission of Data Translation, Inc.

Information furnished by Data Translation, Inc. is believed to be accurate and reliable; however, no responsibility is assumed by Data Translation, Inc. for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of Data Translation, Inc.

Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer software clause at 48 C.F.R, 252.227-7013, or in subparagraph (c)(2) of the Commercial computer Software - Registered Rights clause at 48 C.F.R., 52-227-19 as applicable. Data Translation, Inc., 100 Locke Drive, Marlboro, MA 01752

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

Data Translation® is a registered trademark and DT Vision Foundry is a trademark of Data Translation, Inc.

All other brand and product names are trademarks or registered trademarks of their respective companies.

Table of Contents

About this Manual

This manual describes the API for the main application of DT Vision Foundry 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 DT Vision Foundry or to create a machine vision application using DT Vision Foundry. You should be familiar with Windows programming using the Windows 2000 or Windows XP 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 familiar with COM programming.

What You Should Learn from this Manual

The main DT Vision Foundry 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 to create your own custom tools and machine vision application using the API for DT Vision Foundry and its tools.

The manual is organized as follows:

- Chapter 1, "Introducing the DT Vision Foundry API," provides an overview of the API for DT Vision Foundry main application, and provides installation and technical support information.
- Chapter 2, "Using the DT Vision Foundry API," describes the API for the DT Vision Foundry 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 Barcode Tool API," describes the API for the Barcode tool.
- Chapter 6, "Using the Blob Analysis Tool API," describes the API for the Blob Analysis tool.
- Chapter 7, "Using the Contour Classifier Tool API," describes the API for the Contour Classifier tool.
- Chapter 8, "Using the Custom Script Tool API," describes the API for the Custom Script tool.
- Chapter 9, "Using the Data Matrix Reader Tool API," describes the API for the Digital I/O tool.
- Chapter 10, "Using the Digital I/O Tool API," describes the API for the Digital I/O tool.
- Chapter 11, "Using the Edge Finder Tool API," describes the API for the Edge Finder tool.
- Chapter 12, "Using the File Manager Tool API," describes the API for the File Manager tool.
- Chapter 13, "Using the Filter Tool API," describes the API for the Filter tool.
- Chapter 14, "Using the Gauge Tool API," describes the API for the Gauge tool.
- Chapter 15, "Using the Histogram Tool API," describes the API for the Histogram tool.
- Chapter 16, "Using the Image Classifier Tool API," describes the API for the Image Classifier tool.
- Chapter 17, "Using the Image Modifier Tool API," describes the API for the Image Modifier tool.
- Chapter 18, "Using the Line Profile Tool API," describes the API for the Line Profile tool.

- Chapter 19, "Using the Morphology Tool API," describes the API for the Morphology tool.
- Chapter 20, "Using the Picture Tool API," describes the API for the Picture tool.
- Chapter 21, "Using the Pixel Change Tool API," describes the API for the Pixel Change tool.
- Chapter 22, "Using the Polar Unwrap Tool API," describes the API for the Polar Unwrap tool.
- Chapter 23, "Using the ROI Shape Fitter Tool API," describes the API for the ROI Shape Fitter tool.
- Chapter 24, "Using the Search Tool API," describes the API for the Search tool.
- Chapter 25, "Using the Serial I/O Tool API," describes the API for the Serial I/O tool.
- Chapter 26, "Using the Sound Tool API," describes the API for the Sound tool.
- Chapter 27, "Using the Text Tool API," describes the API for the Text tool.
- Chapter 28, "Using the Threshold Tool API," describes the API for the Threshold tool.
- Chapter 29, "Creating DT Vision Foundry Tools," describes how to create custom tools using DT Vision Foundry.

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.

- Function 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 DT Vision Foundry:

- *DT Vision Foundry User's Manual*, which is shipped with the software.
- DT Vision Foundry online help, which is part of the DT Vision Foundry software.

Where to Get Help

Should you run into problems installing or using DT Vision Foundry, 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.

1

Introducing the DT Vision Foundry API

What is the DT Vision Foundry API?

DT Vision Foundry is an application and an API dedicated to machine vision and 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 or machine vision applications. All tools, all tool APIs, and the DT Vision Foundry 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 DT Vision Foundry API is a small, robust, and easy-to-use core machine vision API that can be used in all areas of imaging. At the center of the DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry main application and all of its tools.

What the API Is Not

Although DT Vision Foundry 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 machine vision application, or a custom algorithm that uses every type of functionality at the same time. Refer to Chapter 29 starting on page 937 for more information on creating custom tools.

Installation

The DT Vision Foundry API is installed during the DT Vision Foundry product installation. Please refer to the installation instructions in the *DT Vision Foundry User's Manual*.

Service and Support

The goal of this manual is to help you use the APIs for the DT Vision Foundry 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 DT Vision Foundry API

Overview of the DT Vision Foundry API

The API for the DT Vision Foundry main application consists of a set of object-oriented classes that are derived from a base DT Vision Foundry object. The classes are 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 objects,
- Graph objects,
- List objects,
- Calibration objects, and
- Device Manager objects.

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 DT Vision Foundry Base Class Object

All objects in the DT Vision Foundry API are derived from a base class named CcHLObject; therefore, all DT Vision Foundry objects contain the following items:

- **Name** –All objects in DT Vision Foundry 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 DT Vision Foundry 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 type, 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 Method

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 63.
HLOBJECT_TYPE_ROI	ROI object. To determine the type of ROI object, see **GetROIType( )** on page 104.
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 Object

Image objects are the core objects of the DT Vision Foundry 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. DT Vision Foundry 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 DT Vision Foundry 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.
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 True-Color HSL 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	DT Vision Foundry 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
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 28 starting on page 925.
- **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 DT Vision Foundry object. If you are writing a tool to use with DT Vision Foundry, 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.
- **Child image method** –This method allows you to get a new image that is defined by the parent image and an ROI.

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.
	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.
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.

Table 4: Image Object Methods (cont.)

Method Type	Method Name	Method Description
24-Bit HSL True Color Image Specialized Methods (cont.)	GetBitmapImageDataHSL( )	Returns a pointer to the HSL image data.
	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.
Child Image Method	GetRegion()	Allows you to create a new (child) image that is defined by the parent image and an ROI.

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

Table 5: Predefined Constant Values (cont.)

Pre-Defined Constant	Displayed Color
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 56) or fast image data access methods (described starting on page 60). For more examples of how to use these methods, refer to Chapter 29 starting on page 937.

The returned pointer points to the start of the image's overlay data. This is position 0,0.

Notes (cont.)

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.
- 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.

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.

Notes (cont.)

- 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 28 starting on page 925 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 DT Vision Foundry) 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 DT Vision Foundry 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, which applies to both RGB and HSL color 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.

To view an image using all of the color tables, you can use the menu item **Display | Grayscale Color Mode** in the DT Vision Foundry main application. For more information, refer to the *DT Vision Foundry 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 28 starting on page 925.

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 28 starting on page 925.

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

The thresholding structure is described as follows:

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

Notes (cont.)

```
//Low Limit of thresholding region
int iRed;
//Color of this region
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);
```

or

```
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 RGB color table, which applies to both RGB and HSL 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 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 *SET_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 90 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 29 starting on page 937. Also, the example change tool provides all the code necessary to rebuild the entire tool. An example of how to access the image data both directly and using the EZ method of access is located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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.

2

Notes (cont.)

The pointer types are as follows:

- 16-bit grayscale image –unsigned short*
- 32-bit grayscale image –int*
- Floating point grayscale image –float*
- 24-bit 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 C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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.

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 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 (cont.):

- 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.

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).

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
//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;
```

Example (cont.)

```
//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 iColorTableTypeFlag

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 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.
- 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 69.

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 DT Vision Foundry main application, DT Vision Foundry 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 DT Vision Foundry, 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 DT Vision Foundry.

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.

2

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 DT Vision Foundry.

Return Values

−1	Unsuccessful
Instance value.	Successful.

Point Conversion Methods

DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry object. If you are writing a tool to use with DT Vision Foundry, 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 was designed specifically for use by the DT Vision Foundry main application. If you are creating a tool for use with the DT Vision Foundry 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 DT Vision Foundry 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 196 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.

Description (cont.):

- 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 28 starting on page 925.

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.
- HSL_ACCESS_SAT –Access the HSL image data by accessing its saturation plane.
- HSL_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.
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 28 starting on page 925.

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( )**.

Notes (cont.)

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.

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

Child Image Method

The **GetRegion()** method allows you to create a new image (called a child image) that is defined by the parent image and an ROI.

GetRegion

Syntax

```
CcImage *GetRegion (
   CcRoiBase *pRoi,
   int iBackColor,
   BOOL bPadWidth) = TRUE);
```

Include File C_Image.h

Description Creates a new (child) image that is defined by the parent image and an ROI.

Parameters

Name: pRoi

Description: A pointer to an ROI object.

Name: iBackColor

Description: Sets the background color of the child image. Values range from 1 to 255.

Name: bPadWidth

Description: If FALSE, the child image is the same width as the bounding box of the ROI. If TRUE, the width of the child image is the closest multiple of four to the width of the bounding box of the ROI. For example, if the bounding box of the ROI is 211 and *bPadWidth* is TRUE, the width of the child image is 212 (the closest multiple of 4).

Notes

No matter what type of ROI is specified by *pRoi*, the new image is shaped like a rectangle and is the same size as the bounding box for the ROI.

Only the following image types support this method:

- IMAGE_TYPE_BINARY
- IMAGE_TYPE_08BIT_GS
- IMAGE_TYPE_16BIT_GS
- IMAGE_TYPE_24BIT_RGB
- IMAGE_TYPE_24BIT_HSL

Return Values

A pointer to the new child image.

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. DT Vision Foundry 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	DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry main application, each viewport can have only one active (selected) ROI at the same time. If you are writing a tool to use with DT Vision Foundry, do not use these methods; they are already in use by the DT Vision Foundry 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 DT Vision Foundry 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 109.

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. DT Vision Foundry 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 DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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. DT Vision Foundry starts a new line segment when you release the left mouse button.

In DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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 DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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 DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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.

2

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. 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 50.

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

In DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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.

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 method
      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 located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, by default. This is also further discussed on Chapter 21 starting on page 795.

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.

Notes (cont.)

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, described in Chapter 21 starting on page 795. All necessary code to rebuild the entire example change tool is located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 DT Vision Foundry 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 DT Vision Foundry 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.

Description (cont.): 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.

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 DT Vision Foundry 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 DT Vision Foundry, an array of points comprises a curve. The points are contained in a DT Vision Foundry 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 Methods	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 DT Vision Foundry 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.
Show/Print Method	ShowGraph( )	Displays the graph and all curves in a window (or prints them to the printer).

Table 10: Graph Object Methods (cont.)

Method Type	Method Name	Method Description
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.
Grid Marking Methods	SetGridMarkings( )	Sets new grid markings for the graph.
	GetGridMarkings( )	Gets the current grid markings for the graph.

Table 10: Graph Object Methods (cont.)

Method Type	Method Name	Method Description
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 CcDTTool::OnPaint( )
{
//This is so the background color
//of the text is the correct color
 ::InvalidateRect(m_hWnd,
   NULL,TRUE);
//Call Begin & End Paint and
//get the HDC
CPaintDC dc(this);
```

Example (cont.)

```
//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 CcDTTool::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);

//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).**

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.

Notes (cont.)

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.

Notes (cont.)

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.

2

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 DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry object and store them in a DT Vision Foundry 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 DT Vision Foundry objects have a name. Before inserting the object into the List object, make sure it has a name associated with it using the DT Vision Foundry 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.

2

Table 11: List Object Methods (cont.)

Method Type	Method Name	Method Description
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.
	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.

Table 11: List Object Methods (cont.)

Method Type	Method Name	Method Description
General Methods (cont.)	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( )**.

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.

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 6 starting on page 307 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	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. 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 an integer that receives the image's width, which is used to calibrate the object.

Name: iHeightOfImage

Description: Pointer to an integer that receives the image's height, which is 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.

Device Manager Objects

A Device Manager object is derived from a single C++ class (CcDeviceManager).

Note: Currently, the Device Manager supports imaging and digital I/O devices only. The API for the Device Manager is intended to be used with the APIs for the Picture tool, described in Chapter 20 starting on page 699, and the Digital I/O tool, described in Chapter 10 starting on page 463.

The methods for the Device Manager class, grouped by method type, are summarized in Table 13.

Table 13: Device Manager Object Methods

Method Type	Method Name	Method Description
Constructor & Destructor Methods	CcDeviceManager( )	Constructor.
	CcDeviceManager( )	Destructor.
Initialize and Uninitialize Methods	Initialize( )	Prepares the Device Manager for use.
	Uninitialize( )	Prepares the Device Manager for termination.
Information Methods	GetPluginNames( )	Returns the names of all plugins for imaging or digital I/O devices.
	GetDeviceNames( )	Returns the names of all devices for a specified plugin.
	GetDeviceObject( )	Returns the device object that is identified by the plugin name, device name, and device interface.

Table 13: Device Manager Object Methods (cont.)

Method Type	Method Name	Method Description
Information Methods (cont.)	GetErrorText( )	Returns the description of the last error that occurred.
Save and Load Methods	SaveDeviceManagerState( )	Saves the current state of all devices in the Device Manager to the named file.
	LoadDeviceManagerState( )	Loads Device Manager settings from a file and applies them to the devices in the Device Manager.

Constructor and Destructor Methods

This section describes the constructor and destructor for the Device Manager object.

CcDeviceManager( ) and ~ CcDeviceManager ( )

Syntax

```
CcDeviceManager
~CcDeviceManager
```

Include File C_DeviceManager.h

Description The standard constructor and destructor for the object.

Initialize and Uninitialze Methods

This section describes the methods that are used to prepare the Device Manager for use and for termination.

Initialize

Syntax

```
int Initialize (
   HWnd hWnd);
```

Include File C_DeviceManager.h

Description Prepares the Device Manager for use.

Parameters

Name: hWnd

Description: The handle to the window that receives initialization status messages. This value must be a valid window handle or NULL if the application does not wish to process status messages.

Notes Call this method once (preferably when the application is initialized) before calling any other Device Manager methods.

When the initialization process begins, the Device Manager sends the message DEVMGR_INIT_BEGIN to the window identified by *hWnd*. The least significant word of the long parameter (*LPARAM*) of this message contains the number of devices to initialize. The Device Manager then sends the message DEVMGR_INIT_UPDATE to the window for each device, as it is initialized. Once all devices have been initialized, the Device Manager sends the message DEVMGR_INIT_DONE.

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Error text buffer.
TCHAR szText[500];

//Initialize the Device Manager.
//Ignore initialization status
//messages.
if (DevMan.Initialize(NULL)) < 0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

Uninitialize

Syntax `int Uninitialize ();`

Include File C_DeviceManager.h

Description Prepares the Device Manager for termination.

Parameters None

Notes Call this method once before the application terminates.

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Error text buffer.
TCHAR szText[500];

//Uninitialize the Device Manager.
if (DevMan.Uninitialize()) < 0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

Information Methods

This section describes the methods that are used to return information for use by the Device Manager.

GetPluginNames

Syntax

```
int GetPluginNames (
   int nDeviceType,
   CcStringList pPluginNames);
```

Include File C_DeviceManager.h

Description Returns the names of all plugins that provide devices with the identified type of device interface.

Parameters

Name: nDeviceType

Description: Specifies the device interface to query. This value must be DEVINTF_IMAGEDEVICE (for imaging devices), DEVINTF_DIGIODEVICE (for digital I/O devices), or DEVINTF_ALL (for both imaging and digital I/O devices).

Name: pPluginNames

Description: A pointer to a string list object that receives the plugin names. (The existing contents of the supplied list are destroyed.) This value must not be NULL.

Notes A plugin is a module that manages one or multiple imaging and/or digital I/O devices. It is the primary means by which specific imaging and digital I/O devices are added to the Device Manager.

Only the names of the plugins that support the device interface specified by *nDeviceType* are returned by this method.

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//List to receive names.
CcStringList Names;
//Error text buffer.
TCHAR szText[500];
//Get the names of all plugins
//with devices that support the
//imaging interface.
if (DevMan.GetPluginNames(
   DEVINTF_IMAGEDEVICE, &Names)) <
   0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

GetDeviceNames

Syntax

```
int GetDeviceNames (
   LPSTR szPluginName,
   int nDeviceType,
   CcStringList pDeviceNames);
```

Include File C_DeviceManager.h

Description Returns the names of all devices for the named plugin that support the specified device interface (imaging and/or digital I/O).

Parameters

Name: szPluginName

Description: Specifies the name of the plugin to query. This value must not be NULL.

Name: nDeviceType

Description: Specifies the device interface to query. This value must be DEVINTF_IMAGEDEVICE (for imaging devices), DEVINTF_DIGIODEVICE (for digital I/O devices), or DEVINTF_ALL (for both imaging and digital I/O devices).

Name: pDeviceNames

Description: A pointer to a string list object that receives the device names. (The existing contents of the supplied list are destroyed.) This value must not be NULL.

Notes You can obtain a list of the plugins that are provided by the Device Manager using the **GetPluginNames** method, described on page 211.

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//List to receive names.
CcStringList Names;
//Error text buffer.
TCHAR szText[500];
```

Example (cont.)

```
TCHAR szPluginName[] = "DT-MACH/MV
   PCI Frame Grabbers";
//Get the names of all devices for
//the plugin named "DT-MACH/MV PCI
//Frame Grabbers" that provide an
//imaging device interface.
if (DevMan.GetDeviceNames(
   szPluginName,
   DEVINTF_IMAGEDEVICE, &Names)) <
   0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

GetDeviceObject

Syntax

```
int GetDeviceObject (
   LPSTR szPluginName,
   LPSTR szDeviceName,
   int nDeviceType,
   void **ppObject);
```

Include File C_DeviceManager.h

Description Returns the device object that is identified by the plugin name, device name, and device interface.

Parameters

Name: szPluginName

Description: Specifies the name of the plugin that supports the target device. This value must not be NULL.

Name:	szDeviceName
Description:	Specifies the name of the target device. This value must not be NULL.
Name:	nDeviceType
Description:	Specifies the device interface to return on the target device. This value must be DEVINTF_IMAGEDEVICE (for imaging devices) or DEVINTF_DIGIODEVICE (for digital I/O devices).
Name:	ppObject
Description:	A pointer to a pointer that receives the requested device object. This value must not be NULL.

Notes

A device object is the primary means through which an application manipulates a device (such as acquiring an image, reading and writing the state of digital I/O lines, and so on).

The object returned by this method is managed internally by the Device Manager and must not be deleted by the application.

The type of object returned depends on the value of *nDeviceType*. If an imaging interface is requested (DEVINTF_IMAGEDEVICE), a pointer to an object of type CcImageDevice is returned. If a digital I/O interface is requested (DEVINTF_DIGIODEVICE), a pointer to an object of type CcDigIODevice is returned. If the requested device interface is not supported, this method returns an error.

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Pointer to receive object.
CcImageDevice *pDevice;
//Error text buffer.
TCHAR szText[500];
//Set up plugin and device names
TCHAR szPluginName[] = "DT-MACH/MV
   PCI Frame Grabbers";
TCHAR szDeviceName[] = "DT3162-1";
//Get a reference to the
//"DT3162-1" device for the plugin
//named "DT-MACH/MV PCI Frame
//Grabbers".
if (DevMan.GetDeviceObject(
   szPluginName, szDeviceName,
   DEVINTF_IMAGEDEVICE, (void **)
   &pDevice)) < 0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
//pDevice now contains a pointer
//to the device object that
//represents the DT3162. This
//object is managed internally by
//the Device Manager and must not
//be deleted by the application.
```

GetErrorText

Syntax

```
int GetErrorText (
   LPSTR szErrorText,
   int nBufSize);
```

Include File C_DeviceManager.h

Description Returns a description of the last error that occurred.

Parameters

Name: szErrorText

Description: Specifies a buffer to receive the last error text. This value must not be NULL.

Name: nBufSize

Description: Specifies the size of the supplied character buffer.

Notes None

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Error text buffer.
TCHAR szText[500];

//Get error text.
DevMan.GetErrorText(szText, 500);
```

Save and Load Methods

This section describes the methods that are used to save Device Manager settings and load Device Manager settings.

SaveDeviceManagerState

Syntax

```
int SaveDeviceManagerState(
   LPSTR szFileName);
```

Include File C_DeviceManager.h

Description Saves the current state of all devices in the Device Manager to the named file.

Parameters

Name: szFileName

Description: Specifies the name of the file in which the state of the Device Manager is persisted. This value must not be NULL.

Notes None

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Error text buffer.
TCHAR szText[500];
//Save the state of the Device
//Manager to the file named
//"Settings.dm".
```

Example (cont.)

```
if (DevMan.SaveDeviceManagerState
   ("Settings.dm") < 0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

LoadDeviceManagerState

Syntax

```
int LoadDeviceManagerState(
   LPSTR szFileName);
```

Include File C_DeviceManager.h

Description Loads the settings from the specified file and applies them to the devices in the Device Manager.

Parameters

Name: szFileName

Description: Specifies the name of the file from which the state of the Device Manager is loaded. This value must not be NULL.

Notes None

Return Values

< 0 The operation failed.

0 The operation was successful.

Example The following is a sample code fragment:

```
//Device Manager object.
CcDeviceManager DevMan;
//Error text buffer.
TCHAR szText[500];
```

Example (cont.)

```
//Load the state of the Device
//Manager from the file named
//"Settings.dm".
if (DevMan.LoadDeviceManagerState
   ("Settings.dm") < 0)
{
  //Get error text.
  DevMan.GetErrorText(szText,
   500);
}
```

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 14.

Table 14: 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 14: 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 14: 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(Cc24BitHSLImage* 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.
- 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.

Notes (cont.)

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 DT Vision Foundry 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( );
```

Example (cont.)

```
//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”);
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”);
```

Example (cont.)

```
//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.

Description (cont.)

The order of operations for this method is as follows:

1. Subtracts 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.

Description (cont.):

- 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 DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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”);
```

3

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 DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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
```

Example (cont.)

```
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
 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(
   “image3.bmp”);
```

Example (cont.)

```
//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.

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.

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

DivRGB( ) and **DivHSL( )** are identical to **Div( )**, except that it performs the operation on all three color planes at once.

This method uses images derived from the DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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( );
 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”);
```

Example (cont.)

```
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);
//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.
```

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.

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 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.

This method uses images derived from the DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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( );
//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”);
```

Example (cont.)

```
 C8BitImage2->OpenBMPFile(
   “image2.bmp”);
 CFloatImage->OpenBMPFile(
   “image3.bmp”);
//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”);
//Free memory
delete C8BitImage1;
delete C8BitImage2;
delete CFloatImage;
delete CRectRoi;
}
```

3

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
```

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.

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

All values are converted to 32-bit integers before performing a logical bitwise OR operation.

This method uses images derived from the DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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( );
 CRectRoi = new CcRoiRect( );

//Initialize ROI
 RECT stROI;
 stROI.bottom = 50;
 stROI.top = 150;
 stROI.left = 50;
 stROI.right = 150;
 CRectRoi->SetRoiImageCord(
   (VOID*)&stROI);
```

Example (cont.)

```
//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”);
//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.
```

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.

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 used unless it was specified by ARITH_THRESHOLD.

Notes

All values are converted to 32-bit integers before performing a logical bitwise XOR operation.

This method uses images derived from the DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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
```

Example (cont.)

```
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( );
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”);
```

Example (cont.)

```
//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”);
 //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.

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.

Notes (cont.)

This method uses images derived from the DT Vision Foundry 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 DT Vision Foundry supplied CcRoiBase class. These include all DT Vision Foundry 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( );
```

Example (cont.)

```
//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”);
//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 15.

Table 15: 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 15: 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);

4

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 DT Vision Foundry 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.
}
```

4

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 Result;

Result = AVI.GetFrameType
   (&iFrameType);
if (Result < 0)
{
// Operation failed - handle error.
}
```

4

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 iResult;
iResult = AVI.GetCompatibleImage
   (&pImage);
if (iResult < 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;
}
```

4

5

Using the Barcode Tool API

Overview of the Barcode Tool

The Barcode API has one object only: the CcBarCode class. This tool reads a barcode symbol located within a rectangular ROI on an input image. The input image and input ROI are both DT Vision Foundry objects. For further information on these objects, see Chapter 2 starting on page 11 and the example program at the end of this section.

Table 16 lists the methods of the CcBarCode object.

Table 16: CcBarCode Object Methods

Method Type	Method Name
Constructor & Destructor	CcBarCode( );
	~ CcBarCode( )
Initialize Methods	int ReadTable(char* cFileName, int iBarcodeType);
Barcode Read Methods	char* ReadBarcode(CcImage* CImage, CcRoiRect* CRoi);
Set Barcode Reading Options Methods	int SetBCOptions(STBCOPTIONS* pstBCOptions);
	int GetBCOptions(STBCOPTIONS* pstBCOptions);
	int SetLPOptions(STLPOPTIONS* pstLPOptions);
	int GetLPOptions(STLPOPTIONS* pstLPOptions);
Save and Restore Methods	int RestoreOptions(char* cFileName);
	int SaveOptions(char* cFileName);
Autothreshold Methods	int SetAutothreshold(int iThresholdValue);
	intGetAutothreshold();

Description of CcBarCode Methods

This section describes each method of the CcBarCode class in detail.

GetBCOptions

Syntax

```
int GetBCOptions(
   STBCOPTIONS* pstBCOptions);
```

Include File C_BCode.h

Description Retrieves the options for reading a barcode.

Parameters

Name: pstBCOptions

Description: Pointer to a STBCOPTIONS structure that contains the barcode options.

Notes Use this method to get the barcode options before reading a barcode using **ReadBarcode( )**. The method takes a pointer to a STBCOPTIONS structure, which is defined as follows:

```
struct stBCOptionsTag {
int iBarcodeType;
int iReadBiDirectional;
int iWhiteBarsBlackSpaces;
int iDoErrorChecking;
int iDoHardToRead;
int iDoStopChar;
int iReadVertical;
};
typedef stBCOptionsTag
   STBCOPTIONS;
```

Notes (cont.)

The elements of the structure are described as follows:

- *iBarcodeType* –The type of barcode that you want to read:
  - HLBARCODE_128 = 128 barcode.
  - HLBARCODE_2_5 = 2 of 5 barcode.
  - HLBARCODE_3_9 = 3 of 9 barcode.
  - HLBARCODE_UPCA = UPC Version A barcode.
  - HLBARCODE_EAN13 = EAN13 barcode.
  - HLBARCODE_EAN8 = EAN8 barcode.
  - HLBARCODE_POSTNET = POSTNET barcode.
- *iReadBiDirectional*
  - 1 = Reads bidirectionally.
  - 0 = Does not read bidirectionally.
- *iWhiteBarsBlackSpaces*
  - 1 = Reads white bars with black spaces.
  - 0 = Does not read white bars with black spaces.
- *iDoErrorChecking*
  - 1 = Performs error checking.
  - 0 = Does not perform error checking.
- *iDoHardToRead*
  - 1 = Invokes the specialized barcode algorithm to read distorted, noisy, hard-to-read barcodes.

Notes (cont.)

- 0 = Does not invoke the specialized barcode algorithm.

- *iDoStopChar*
  - 1 = Checks for the stop character.
  - 0 = Does not check for the stop character.
- *iReadVertical*
  - 1 = Reads the barcode in the vertical direction.
  - 0 = Reads the barcode in the horizontal direction.

Return Values

–1	Unsuccessful.
0	Successful.

GetLPOptions

Syntax

```
int GetLPOptions(
   STLPOPTIONS* pstLPOptions);
```

Include File C_BCode.h

Description Gets the line profile options that were used when reading the barcode.

Parameters

Name: pstLPOptions

Description: Pointer to a STLPOPTIONS structure for getting the line profile options.

Notes Use this method to get the line profile options before reading a barcode using **ReadBarcode( )**.

Notes (cont.)

The method takes a pointer to a STLPOPTIONS structure, which is defined as follows:

```
struct stLPOptionsTag {
int iGenLineAve;
float fGenAmp;
float fGenOffset;
int iGenProfileAve;

int i1stDerGrad;
float f1stDerAmp;
float f1stDerOffset;
int i1stDerProfileAve;

int i2ndDerGrad;
float f2ndDerAmp;
float f2ndDerOffset;
int i2ndDerProfileAve;

float fEdgeFindLo;
float fEdgeFindHi;
};
typedef struct stLPOptionsTag
   STLPOPTIONS;
```

The elements of this structure take the same values that you would enter in the line profile's option boxes when finding edges. For more information, refer to the *DT Vision Foundry User's Manual.*

The following values correspond to the profile options in the line profile's options dialog box:

- *iGenLineAve* –The value of the Width entry.
- *fGenAmp* –The value of the Gain entry.

Notes (cont.)

- *fGenOffset* –The value of the Offset entry.
- *iGenProfileAve* –The value of the Ave entry.

These values correspond to the first derivative options in the line profile's options dialog box:

- *i1stDerGrad* –The value of the Slope entry.
- *f1stDerAmp* –The value of the Gain entry.
- *f1stDerOffset* –The value of the Offset entry.
- *i1stDerProfileAve* –The value of the Ave entry.

The following values correspond to the second derivative options in the line profile's profile options dialog box:

- *i2ndDerGrad* –The value of the Slope entry.
- *f2ndDerAmp* –The value of the Gain entry.
- *f2ndDerOffset* –The value of the Offset entry.
- *i2ndDerProfileAve* –The value of the Ave entry.

The following values correspond to the noise limit options in the line profile's find edges dialog box:

- *fEdgeFindLo* –The value of the low-noise limit.
- *fEdgeFindHi* –The value of the high-noise limit.

Notes (cont.) The POSTNET reader does not use Line Profile APIs; therefore, **SETLPOptions** and **GetLPOptions** are not used for reading POSTNET barcodes.

Return Values

–1 Unsuccessful.

0 Successful.

ReadBarcode

Syntax

```
char* ReadBarcode(
  CcImage* CImage,
  CcRoiRect* CRoi);
```

Include File C_BCode.h

Description Reads the barcode in the image *CImage* with respect to the location designated by *CRoi*.

Parameters

Name: CImage

Description: Pointer to a DT Vision Foundry Image object that contains the barcode to read.

Name: CRoi

Description: Pointer to a RECTANGULAR DT Vision Foundry ROI object that is located around the barcode in the image.

Notes Before calling this method to read a barcode, you must initialize the classes' internal table, which translates bar/space widths into human readable text, by calling **ReadTable( )** for each type of barcode that you want to read.

Return Values

NULL	Unsuccessful.
A string that contains the barcode text.	Successful.

ReadTable

Syntax

```
int ReadTable(
  char* cFileName,
  int iBarcodeType);
```

Include File C_BCode.h

Description Initializes the class's internal table for reading a specific barcode type.

Parameters

Name: cFileName

Description: Full path name of the file that contains the table information for a specific barcode.

Name: iBarcodeType

Description: Specific barcode type whose table you are initializing. Choose one of the following options:

- *HLBARCODE_128* –128 barcode.
- *HLBARCODE_2_5* –2 of 5 barcode.
- *HLBARCODE_3_9* –3 of 9 barcode.
- *HLBARCODE_UPCA* –UPC Version A barcode.
- *HLBARCODE_EAN13* = EAN13 barcode.

Description (cont.):

- *HLBARCODE_EAN8* = EAN8 barcode.
- *HLBARCODE_POSTNET* = POSTNET barcode.

Before calling **ReadBarcode( )** to read a barcode, you must initialize the classes' internal table, which translates bar/space widths into human readable text. These tables are stored on disk in ASCII text and must be loaded before you can read the barcode. For each type of supported barcode that you plan to read, you must load the associated table.

Notes

The files you need to load are shipped with DT Vision Foundry and are as follows:

- Filename: Table128.txt;
  Barcode type: HLBARCODE_128;
  128 barcode.
- Filename: N/A;
  Barcode type: HLBARCODE_2_5;
  2 of 5 barcode.
- Filename: Table3_9.txt;
  Barcode type: HLBARCODE_3_9;
  3 of 9 barcode.
- Filename: TableUPCA.txt;
  Barcode type: HLBARCODE_UPCA;
  UPC Version A barcode.
- Filename: TableEAN.txt
  Barcode type: HLBARCODE_EAN13 and HLBARCODE_EAN8; EAN13 and EAN 8 barcodes, respectively.

Notes (cont.)

- Filename: TablePOSTNET.txt
  Barcode type: HLBARCODE_POSTNET; POSTNET barcode.

No table is provided for barcode 2 of 5. Therefore, this method does not need to be called to read 2 of 5 barcodes.

Returned Value

-1	Unsuccessful.
0	Successful.

RestoreOptions

Syntax

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

Include File C_BCode.h

Description Restores all the options that were used when reading the barcode.

Parameters

Name: cFileName

Description: Full path name of the file from which to restore barcode options.

Notes After using the other methods to set all the options in the barcode class for reading a barcode, you can use this method to restore your options.

Notes (cont.) As with most DT Vision Foundry restore methods, you can also use the Barcode tool to easily set up all the barcode options using its GUI, and then use its SaveAs menu item to save these settings to disk. This saves time and effort in programming. You can then use **RestoreOptions( )** to restore all the options without programming these settings.

Returned Value

–1 Unsuccessful.

0 Successful.

SaveOptions

Syntax `int SaveOptions(char* cFileName);`

Include File C_BCode.h

Description Saves all the options that were used when reading the barcode.

Parameters

Name: cFileName

Description: Full path name of the file in which to save the barcode options.

Notes After using the other methods to set all the options in the barcode class for reading a barcode, you can use this method to save your options.

Notes (cont.) As with most DT Vision Foundry save methods, you can also use the Barcode tool to easily set up all the barcode options using its GUI, and then use its SaveAs menu item to save these settings to disk. This saves time and effort in programming. You can then use **RestoreOptions( )** to restore all the options without programming these settings.

Returned Value

–1	Unsuccessful.
0	Successful.

SetBCOptions

Syntax

```
int SetBCOptions(
   STBCOPTIONS* pstBCOptions);
```

Include File C_BCode.h

Description Sets the barcode options for reading a barcode.

Parameters

Name: pstBCOptions

Description: Pointer to a STBCOPTIONS structure for setting or getting the barcode options.

Notes Use this method to set the barcode options before reading a barcode using **ReadBarcode( )**.

Notes (cont.)

The method takes a pointer to a STBCOPTIONS structure, which is defined as follows:

```
struct stBCOptionsTag {
int iBarcodeType;
int iReadBiDirectional;
int iWhiteBarsBlackSpaces;
int iDoErrorChecking;
int iDoHardToRead;
int iDoStopChar;
int iReadVertical;
};
typedef stBCOptionsTag
   STBCOPTIONS;
```

The elements of the structure are described as follows:

- *iBarcodeType* –The type of barcode you wish to read:
  - HLBARCODE_128 – 128 barcode.
  - HLBARCODE_2_5 – 2 of 5 barcode.
  - HLBARCODE_3_9 – 3 of 9 barcode.
  - HLBARCODE_UPCA – UPC Version A barcode.
  - HLBARCODE_EAN13 = EAN13 barcode.
  - HLBARCODE_EAN8 = EAN8 barcode.
  - HLBARCODE_POSTNET = POSTNET barcode.

Notes (cont.)

- *iReadBiDirectional*
  - 1 = Reads bidirectionally.
  - 0 = Does not read bidirectionally.
- iWhiteBarsBlackSpaces
  - 1 = Reads white bars with black spaces.
  - 0 = Does not read white bars with black spaces.
- iDoErrorChecking
  - 1 = Performs error checking.
  - 0 = Does not perform error checking.
- iDoHardToRead
  - 1 = Invokes the specialized barcode algorithm to read distorted, noisy, or hard-to-read barcodes.
  - 0 = Does not invoke the specialized barcode algorithm.
- *iDoStopChar*
  - 1 = Checks for the stop character.
  - 0 = Does not check for the stop character.
- *iReadVertical*
  - 1 = Reads the barcode in the vertical direction.
  - 0 = Reads the barcode in the horizontal direction.

Returned Value

–1	Unsuccessful.
0	Successful.

SetLPOptions

Syntax

```
int SetLPOptions(STLPOPTIONS*
   pstLPOptions);
```

Include File C_BCode.h

Description Sets the line profile options that are used when reading the barcode.

Parameters

Name: pstLPOptions

Description: Pointer to a STLPOPTIONS structure for setting the line profile options.

Notes Use this method to set the line profile options before reading a barcode using **ReadBarcode( )**.

The method takes a pointer to a STLPOPTIONS structure, which is defined as follows:

```
struct stLPOptionsTag {
int iGenLineAve;
float fGenAmp;
float fGenOffset;
int iGenProfileAve;
int i1stDerGrad;
float f1stDerAmp;
float f1stDerOffset;
int i1stDerProfileAve;
int i2ndDerGrad;
float f2ndDerAmp;
float f2ndDerOffset;
int i2ndDerProfileAve;
float fEdgeFindLo;
float fEdgeFindHi;
};
```

Notes (cont.)

```
typedef struct stLPOptionsTag
    STLPOPTIONS;
```

The elements of this structure take the same values that you would enter in the line profile's option boxes when finding edges. For more information, refer to the *DT Vision Foundry User's Manual.*

The following values correspond to the profile options in the line profile's options dialog box:

- *iGenLineAve* –The value of the Width entry.
- *fGenAmp* –The value of the Gain entry.
- *fGenOffset* –The value of the Offset entry.
- *iGenProfileAve* –The value of the Ave entry.

The following values correspond to the first derivative options in the line profile's options dialog box:

- *i1stDerGrad* –The value of the Slope entry.
- *f1stDerAmp* –The value of the Gain entry.
- *f1stDerOffset* –The value of the Offset entry.
- *i1stDerProfileAve* –The value of the Ave entry.

The following values correspond to the second derivative options in the line profile's options dialog box:

- *i2ndDerGrad* –The value of the Slope entry.
- *f2ndDerAmp* –The value of the Gain entry.

Notes (cont.)

- *f2ndDerOffset* –The value of the Offset entry.
- *i2ndDerProfileAve* –The value of the Ave entry.

The following values correspond to the noise limit options in the line profile's find edges dialog box:

- *fEdgeFindLo* –The value of the low noise limit.
- *fEdgeFindHi* –The value of the high noise limit.

The Barcode Reader does not use **SETLPOptions** and **GetLPOptions** for reading POSTNET barcodes.

Returned Value

–1	Unsuccessful.
0	Successful.

SetAutothreshold

Syntax

```
int SetAutothreshold(int
   iThresholdValue);
```

Include File C_BCode.h

Description Sets the autothreshold value that is used when reading POSTNET barcodes.

Parameters

Name: iThresholdValue

Description: The automatic threshold reference number. Values range from 0 to 100.

Notes The Barcode Reader does not use **SETLPOptions** and **GetLPOptions** for reading POSTNET barcodes.

Returned Value

–1	Unsuccessful.
0	Successful.

GetAutothreshold

Syntax `int GetAutothreshold();`

Include File C_BCode.h

Description Returns the autothreshold value that is used when reading POSTNET barcodes.

Parameters None

Notes The Barcode Reader does not use **SETLPOptions** and **GetLPOptions** for reading POSTNET barcodes.

Returned Value

–1	Unsuccessful.
0 to 100	Successful; the automatic threshold reference value is returned.

Example Program Using the Barcode API

This example uses the CcBarCode class to open a saved barcode setting file from disk, read the barcode, and show the barcode text in a standard Windows message box. Note that the settings file was originally generated using the Barcode tool's GUI. All you have to do is open the file from disk to set up the desired settings. The image and ROI are passed on the parameter list. For information on creating images and ROIs, see Chapter 2 and/or the Picture tool in xx.

Note: For clarity, error checking is not included.

```
int SomeFunction(CcImage* CImage, CcRoiRect* CRoi)
{
CcBarCode CBCode;

//First initialize the barcode class by reading in
//the needed tables
CBCode. ReadTable (“C:\\ TABLE128.txt”,
HLBARCODE_128);
CBCode. ReadTable (“C:\\ Table3_9.txt”,
HLBARCODE_3_9);
CBCode. ReadTable (“C:\\ TableUPCA.txt”,
HLBARCODE_UPCA);

//Now restore all settings in barcode class by
//opening the settings file created with DT Vision
//Foundry
CBCode. ReadTable (“C:\\ BarcodeOpts.hbc”);

//Run the operation
pBarcode = ReadBarcode(CImage,CRoi);
```

```
//Show the barcode in a message box
::MessageBox(m_hWnd,pBarcode, “The barcode is:”,
MB_OK);

return(0);
}
```

5

6

Using the Blob Analysis Tool API

Overview of the Blob Analysis Tool API

The API for the Blob Analysis tool uses several DT Vision Foundry API objects. Therefore, it is recommended that you read Chapter 2 starting on page 11 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 17.

Table 17: 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 SetMinBlobHeight(int iBlobHeight);
	int GetMinBlobHeight(void);
	int SetMaxBlobHeight(int iBlobHeight);
	int GetMaxBlobHeight(void);
	int SetMinBlobWidth(int iBlobWidth);
	int GetMinBlobWidth(void);

Table 17: CcBlobFinder Methods (cont.)

Method Type	Method Name
CcBlobFinder Class Methods (cont.)	int SetMaxBlobWidth(int iBlobWidth);
	int GetMaxBlobWidth(void);
	int SetBlobStatsFlags(BLOBSTATSFLAG* GroupFlags);
	BLOBSTATSFLAG* GetBlobStatsFlags();
	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 18.

Table 18: CcBlob Methods

Method Type	Method Name
Constructor & Destructor Methods	[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( );

6

Table 18: CcBlob Methods (cont.)

Method Type	Method Name
CcBlob Class Methods (cont.)	CcList* GetChildBlobList(void);
	int GetNumOfChildBlobs(void);
	int DelChildrenOnDestructor(BOOL bFlag);
	int SetBlobStatsFlags(BLOBSTATSFLAGS* GroupFlags);
	BLOBSTATSFLAG* GetBlobStatsFlags();
	int SetRemoveBoundaryBlobFlag (int iRemove);
	int GetRemoveBoundaryBlobFlag (void);

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 DT Vision Foundry, you are already attached to an application (DT Vision Foundry) 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.

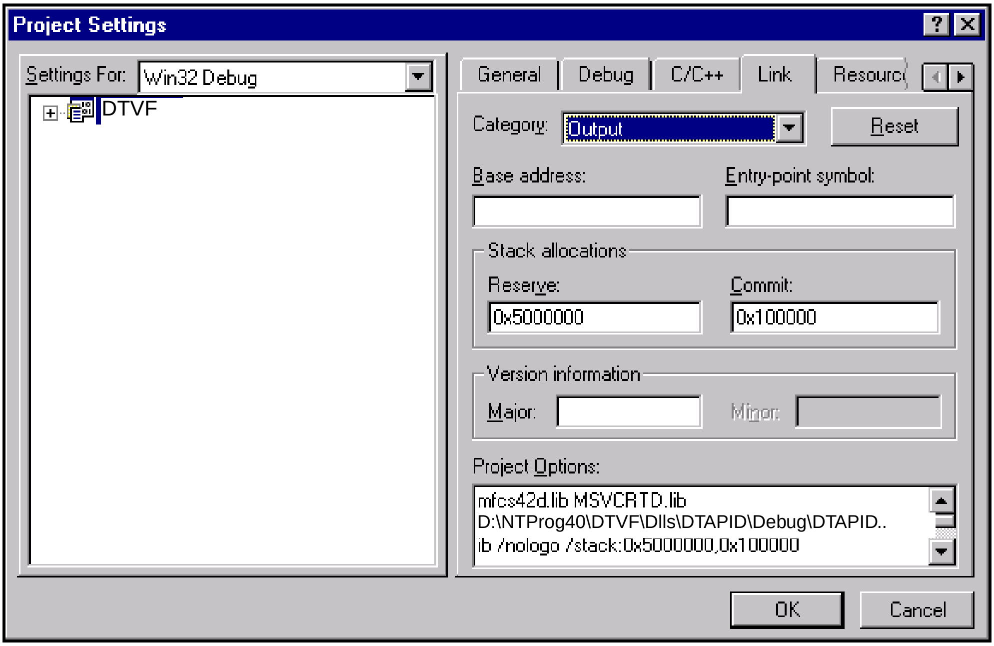


Figure 1: Program Settings

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 *DT Vision Foundry 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 *DT Vision Foundry 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 *DT Vision Foundry 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 *DT Vision Foundry User's Manual.*

Return Values

–1	Unsuccessful.
Maximum blob size.	Successful.

SetMinBlobHeight

Syntax
```
int SetMinBlobHeight(int
   iBlobHeight);
```

Include File C_Blobf.h

Description Sets the minimum blob height (number of pixels).

Parameters

Name: iBlobHeight

Description: Minimum height of the blob.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

GetMinBlobHeight

Syntax `int GetMinBlobHeight(void);`

Include File C_Blobf.h

Description Gets the minimum blob height.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

6

SetMaxBlobHeight

Syntax

```
int SetMaxBlobSize(int
   iBlobHeight);
```

Include File C_Blobf.h

Description Sets the maximum blob height.

Parameters

Name: iBlobHeight

Description: Maximum blob height.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

GetMaxBlobHeight

Syntax `int GetMaxBlobHeight(void);`

Include File C_Blobf.h

Description Returns the maximum blob height.

Notes None.

Return Values

–1 Unsuccessful.

Maximum blob size. Successful.

SetMinBlobWidth

Syntax

```
int SetMinBlobWidth(int
   iBlobWidth);
```

Include File C_Blobf.h

Description Sets the minimum blob width.

Parameters

Name: iBlobWidth

Description: Minimum blob width.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

GetMinBlobWidth

Syntax
```
int GetMinBlobWidth(void);
```

Include File C_Blobf.h

Description Gets the minimum blob width.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

SetMaxBlobWidth

Syntax
```
int SetMaxBlobWidth(int
   iBlobWidth);
```

Include File C_Blobf.h

Description Sets the maximum blob width.

Parameters

Name: iBlobWidth

Description: Maximum blob width.

Notes None.

Return Values

–1 Unsuccessful.

0 Successful.

GetMaxBlobWidth

Syntax

```
int GetMaxBlobWidth(void);
```

Include File C_Blobf.h

Description Returns the maximum blob width.

Notes None.

Return Values

–1 Unsuccessful.

Maximum blob size. Successful.

SetBlobStatsFlags

Syntax

```
int SetBlobStatsFlags(
   BLOBSTATSFLAG* GroupFlags);
```

Include File C_Blobf.h

Description Sets the blob statistics group flags. If a flag is set to TRUE, the values of the corresponding group of statistics are updated.

Parameters

Name: GroupFlags

Description: A pointer to the BlobStatsFlag structure, which is defined as follows:

```
struct BlobStatsFlag
{
BOOL bDoCentroid;
BOOL bDoArea;
BOOL bDoMinMax;
BOOL bDoPixelValue;
BOOL bDoAxis;
BOOL bDoPerimeter;
BOOL bDoRadius;
}
typedef struct BlobStatsFlag
   BLOBSTATSFLAG;
```

Notes

Blob statistics are divided into groups. The following code fragments show the individual statistics in each group:

```
//Always calculate these
//parameters
float fParentNumOfPixels;
float fTotalNumOfPixels;
float fNumOfChildren;

//Centroid Group
float fParentXCentroid;
float fParentYCentroid;
float fParentSumX;
float fParentSumY;
float fTotalXCentroid;
float fTotalYCentroid;
float fTotalSumX;
float fTotalSumY;
```

6

Notes (cont.)

```
//Area Group
float fParentArea;
float fROIArea;
float fParentAreaToROIRatio;
float fChildArea;
float fTotalArea;
float fChildRatio;
float fTotalAreaToROIRatio;

//Min-Max Group
float fMaxX;
float fMaxY;
float fMinX;
float fMinY;
float fYatMaxX;
float fYatMinX;
float fXatMaxY;
float fXatMinY;
float fXDifference;
float fYDifference;
float fBoundingBoxArea;
//fXDifference*fYDifference
float fParentBoxRatio;
//need fParentArea;
float fTotalBoxRatio;
//need fTotalArea

//Pixel Averages Group
float fParentGrayAverage;
float fParentRedAverage;
float fParentGreenAverage;
float fParentBlueAverage;
float fParentGrayTotal;
float fParentRedTotal;
float fParentGreenTotal;
float fParentBlueTotal;
```

Notes (cont.)

```
float fTotalGrayAverage;
float fTotalRedAverage;
float fTotalGreenAverage;
float fTotalBlueAverage;
float fTotalGrayTotal;
float fTotalRedTotal;
float fTotalGreenTotal;
float fTotalBlueTotal;

//Axis group
float fMajorAxisAngle;
float fMinorAxisAngle;
float fMajorAxis;
float fMinorAxis;
float fAxisRatio;
float fTotalSumXX;
float fTotalSumXY;
float fTotalSumYY;
float fParentSumXX;
float fParentSumXY;
float fParentSumYY;

//Perimeter group
float fPerimeter;
float fXPerimeter;
float fYPerimeter;
float fRoundness;
float fPPDA;

//Radius group
float fAvgRadius;
float fMaxRadius;
float fMinRadius;
float fCDistance;
float fMaxRadiusAngle;
float fDiffRadiusAngle;
```

Notes (cont.)

```
float fRadiusRatio;
```

For increased speed, only those statistics in the enabled group are calculated when growing blobs.

Return Values

–1 Unsuccessful.

0 Successful.

GetBlobStatsFlags

Syntax

```
BLOBSTATSFLAG*
   GetBlobStatsFlags();
```

Include File C_Blobf.h

Description Returns the blob statistics group flags.

Parameters None

Notes The BlobStatsFlag structure is defined as follows:

```
struct BlobStatsFlag
{
BOOL bDoCentroid;
BOOL bDoArea;
BOOL bDoMinMax;
BOOL bDoPixelValue;
BOOL bDoAxis;
BOOL bDoPerimeter;
BOOL bDoRadius;
}
typedef struct BlobStatsFlag
   BLOBSTATSFLAG;
```

Return Values

A NULL pointer.	Unsuccessful.
A pointer to the BLOBSTATSFLAG structure.	Successful.

FindChildren

Syntax `int FindChildren(BOOL bFind);`

Include File C_Blobf.h

Description Enables or disables the growing of child blobs during the blob growing process.

Parameters

Name: bFind

Description: Enter a value of TRUE to find all child blobs, enter a value of FALSE if you do not want to find child blobs.

Notes Growing child blobs is the default. When the option is enabled, child blobs are calculated and grown. If the option is disabled, the child blobs are not grown, and the operation speeds up the overall growing of the blobs. This is useful if only the blob's perimeter ROI is important, if you do not care about child blob information, or you know that no child blobs exist.

If more than one level of child blobs is present and you do not grow the 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 *DT Vision Foundry 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 28 starting on page 925 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 DT Vision Foundry API-supplied class that contains a list of DT Vision Foundry objects. You must cast any pointers returned by the CcList* methods. For more information on the CcList class, see the DT Vision Foundry API, described in Chapter 2 starting on 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.

Notes (cont.)

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.

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.

6

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 *DT Vision Foundry 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.

```
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;
```

Example (cont.)

```
//Child(Hole) Info
int iNumOfChildren;
float fChildArea;
float fTotalArea;
float fChildRatio;

int iTotalNumOfPixels;

float fTotalXCentroid;
float fTotalYCentroid;

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;
```

Example (cont.)

```
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;
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 329.

For a detailed view of the structure STBLOBSTATS, refer to the header file C_Blob.h located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Include, by default.

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.

6

Notes (cont.)

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 DT Vision Foundry 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.

Notes (cont.)

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.

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 DT Vision Foundry 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
```

Example (cont.)

```
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.

SetBlobStatsFlags

Syntax

```
int SetBlobStatsFlags(
   BLOBSTATSFLAG* GroupFlags);
```

Include File C_Blob.h

Description Sets the blob statistics group flags. If a flag is set to TRUE, the values of the corresponding group of statistics are updated.

Parameters

Name: GroupFlags

Description: A pointer to the BlobStatsFlag structure, which is defined as follows:

```
struct BlobStatsFlag
{
BOOL bDoCentroid;
BOOL bDoArea;
BOOL bDoMinMax;
BOOL bDoPixelValue;
BOOL bDoAxis;
BOOL bDoPerimeter;
BOOL bDoRadius;
}
typedef struct BlobStatsFlag
   BLOBSTATSFLAG;
```

Notes Blob statistics are divided into groups. The following code fragments show the individual statistics in each group:

```
//Always calculate these
//parameters
float fParentNumOfPixels;
float fTotalNumOfPixels;
float fNumOfChildren;

//Centroid Group
float fParentXCentroid;
float fParentYCentroid;
float fParentSumX;
float fParentSumY;
float fTotalXCentroid;
float fTotalYCentroid;
float fTotalSumX;
float fTotalSumY;

//Area Group
float fParentArea;
float fROIArea;
float fParentAreaToROIRatio;
float fChildArea;
float fTotalArea;
float fChildRatio;
float fTotalAreaToROIRatio;

//Min-Max Group
float fMaxX;
float fMaxY;
float fMinX;
float fMinY;
float fYatMaxX;
float fYatMinX;
float fXatMaxY;
```

Notes (cont.)

```
float fXatMinY;
float fXDifference;
float fYDifference;
float fBoundingBoxArea;
float fParentBoxRatio;
float fTotalBoxRatio;

//Pixel Averages Group
float fParentGrayAverage;
float fParentRedAverage;
float fParentGreenAverage;
float fParentBlueAverage;
float fParentGrayTotal;
float fParentRedTotal;
float fParentGreenTotal;
float fParentBlueTotal;
float fTotalGrayAverage;
float fTotalRedAverage;
float fTotalGreenAverage;
float fTotalBlueAverage;
float fTotalGrayTotal;
float fTotalRedTotal;
float fTotalGreenTotal;
float fTotalBlueTotal;


//Axis group
float fMajorAxisAngle;
float fMinorAxisAngle;
float fMajorAxis;
float fMinorAxis;
float fAxisRatio;
float fTotalSumXX;
float fTotalSumXY;
float fTotalSumYY;
float fParentSumXX;
```

Notes (cont.)

```
float fParentSumXY;
float fParentSumYY;

//Perimeter group
float fPerimeter;
float fXPerimeter;
float fYPerimeter;
float fRoundness;
float fPPDA;

//Radius group
float fAvgRadius;
float fMaxRadius;
float fMinRadius;
float fCDistance;
float fMaxRadiusAngle;
float fDiffRadiusAngle;
float fRadiusRatio;
```

For increased speed, only those statistics in the enabled group are calculated when growing blobs.

Return Values

–1	Unsuccessful.
0	Successful.

GetBlobStatsFlags

Syntax

```
BLOBSTATSFLAG*
   GetBlobStatsFlags();
```

Include File C_Blob.h

Description Returns the blob statistics group flags.

Parameters None

Notes

The BlobStatsFlag structure is defined as follows:

```
struct BlobStatsFlag
{
BOOL bDoCentroid;
BOOL bDoArea;
BOOL bDoMinMax;
BOOL bDoPixelValue;
BOOL bDoAxis;
BOOL bDoPerimeter;
BOOL bDoRadius;
}
typedef struct BlobStatsFlag
   BLOBSTATSFLAG;
```

Return Values

A NULL pointer.	Unsuccessful.
A pointer to the BLOBSTATSFLAG structure.	Successful.

SetRemoveBoundaryBlobFlag

Syntax

```
int SetRemoveBoundaryBlobFlag(
   int iRemove);
```

Include File C_Blob.h

Description Specifies whether or not to remove boundary blobs.

Parameters

Name: iRemove

Description: Sets the boundary blob flag to one of the following:

- 1 –Removes boundary blobs.
- 0 –Does not remove boundary blobs.

Notes None

Returned Value

-1 Unsuccessful.

0 Successful.

GetRemoveBoundaryBlobFlag

Syntax

```
int GetRemoveBoundaryBlobFlag(
    void);
```

Include File C_Blob.h

Description Returns the flag that determines whether or not to remove boundary blobs.

Parameters None

Notes None

Returned Value

1 The flag is set to remove boundary blobs.

0 The flag is set not to remove boundary blobs.

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.

```
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);

//Do all the blob stats

//Set Blob statistic group flag
BLOBSTATSFLAG GroupFlag;
GroupFlag.bDoCentroid = TRUE;
GroupFlag.bDoArea = TRUE;
GroupFlag.bDoMinMax = TRUE;
GroupFlag.bDoPixelValue = TRUE;
GroupFlag.bDoAxis = TRUE;
GroupFlag.bDoPerimeter = TRUE;
GroupFlag.bDoRadius = TRUE;
CBlobFinder->SetBlobStatsFlags(&GroupFlag);

//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);
}
```

7

Using the Contour Classifier Tool API

Introduction

The Contour Classifier tool is a C++ class that is designed to work within the DT Vision Foundry environment. It is a general-purpose classifier for enclosed contours and is meant to be used with the API for the Blob Analysis tool. Refer to Chapter 6 starting on page 307 for more information on the Blob Analysis tool API.

Contours extracted using the Blob Analysis tool are fed into the Contour Classifier tool for the purpose of building contour catalogs and classifying contours under test (using those catalogs).

Given the catalog and the contours under test, the Contour Classifier tool produces the following results:

- The name of the catalog element that best matches the contour under test,
- Three Euler angles (alpha, beta, and gamma) that describe the rotation of the contour under test with respect to the contour in the catalog (see Figure 2), and
- The score, which is a measurement of how good the match is between the contour under test and the contour in the catalog.

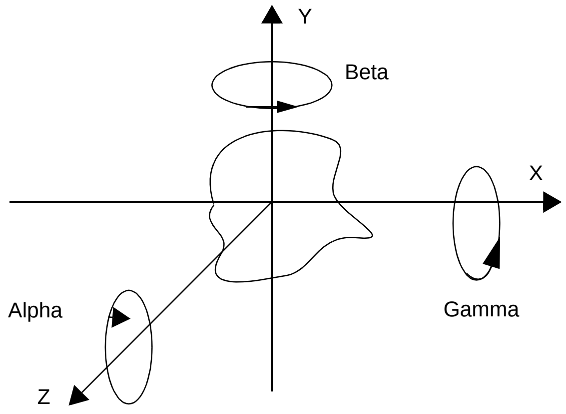


Figure 2: Euler Angles Reported by the Contour Classifier Tool

The API for the Contour Classifier tool has one object only: the CcContour class. This tool performs contour classifier operations 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 CcContour class uses a standard constructor and destructor and the class methods listed in Table 19.

Table 19: CcContour Class Methods

Method Type	Method Name
Constructor & Destructor Methods	CcContour();
	~CContour();
CcContour Class Methods	void Set3Drotation(bool b3DRotation);
	void SetAngleDelimiting (bool bAngleDelimiting);
	void SetExtendedClassification(bool bExtendedClassification);
	bool SetComparisonDepth(int iComparisonDepth);
	bool SetScale(float fScaleMin, float fScaleMax);
	bool SetCenterAngleA(int iCenterAngleA);
	bool SetCenterAngleB(int iCenterAngleB);
	bool SetCenterAngleG(int iCenterAngleG);
	bool SetNegAngleA(int iNegAngleA);
	bool SetNegAngleB(int iNegAngleB);
	bool SetNegAngleG(int iNegAngleG);
	bool SetPosAngleA(int iPosAngleA);
	bool SetPosAngleB(int iPosAngleB);
	bool SetPosAngleG(int iPosAngleG);

Table 19: CcContour Class Methods (cont.)

Method Type	Method Name
CcContour Class Methods (cont.)	bool Get3DRotation(void);
	bool GetAngleDelimiting(void);
	bool GetExtendedClassification(void);
	int GetComparisonDepth(void);
	float GetScaleMin(void);
	float GetScaleMax(void);
	int GetCenterAngleA(void);
	int GetCenterAngleB(void);
	int GetCenterAngleG(void);
	int GetNegAngleA(void);
	int GetNegAngleB(void);
	int GetNegAngleG(void);
	int GetPosAngleA(void);
	int GetPosAngleB(void);
	int GetPosAngleG(void);
	bool SetDelimiterString(char *cDelimiterString);
	void CleanIUTList(void);
	void CleanCatalog(void);
	int GetResultsCount(void);
	int GetCatalogCount(void);
	int GetIUTCount(void);
	bool SupplyContour(CcRoiBase *CRoi);
	int GetError(void);

Table 19: CcContour Class Methods (cont.)

Method Type	Method Name
CcContour Class Methods (cont.)	bool BuildCatalog(void);
	bool RebuildCatalog(void);
	bool NameCatalogElements(char *pcString, int iStartIndex);
	char *GetCatalogString(void);
	bool SaveCatalog(char *cFileName);
	bool LoadCatalog(char *cFileName);
	void ClassifyContours(void);
	STCATRESULT *GetResult(int iIndex);
	CcImage *MakeImageOfCATList(void);
	CcImage *MakeImageOfIUTList(void);

CcContour Methods

This section describes each method of the CcContour class in detail.

Set3Drotation

Syntax `void Set3DRotation(bool b3DRotation);`

Include Files C_Contour.h

Description Allows the contours under test to rotate in three dimensions.

Parameters

Name: b3DRotation

Description: When TRUE, three dimensional rotation is allowed. When FALSE, three dimensional rotation is not allowed.

Notes The tool performs additional computations to classify three-dimensional contours.

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the contour class
CcContour m_CContour;

//Set everything to the defaults
//True is for 3D
m_stContOpt.b3DRotation =
   m_CContour.Get3DRotation();

//Window settings
m_stContOpt.iAffineWindow =
   m_CContour.GetAffineWindow();
m_stContOpt.iRotationWindow =
   m_CContour.GetRotationWindow();
```

Example (cont.)

```
//Post-processing approaches
//correlation is available only
//with the affine method
m_stContOpt.bAngleDelimiting=
  m_CContour.GetAngleDelimiting();
m_stContOpt.bExtendedClassification=
  m_CContour.GetExtendedClassification();

//post-processing operations
m_stContOpt.iEuclidianWindow =
   m_CContour.GetEuclianWindow();
m_stContOpt.iComparisonDepth =
   m_CContour.GetComparisonDepth();

//Angle delimiting
m_stContOpt.fScaleMin =
   m_CContour.GetScaleMin();
m_stContOpt.fScaleMax =
   m_CContour.GetScaleMax();

m_stContOpt.iCenterAngleA =
   m_CContour.GetCenterAngleA();
m_stContOpt.iCenterAngleB =
   m_CContour.GetCenterAngleB();
m_stContOpt.iCenterAngleG =
   m_CContour.GetCenterAngleG();

m_stContOpt.iNegAngleA =
   m_CContour.GetNegAngleA();
m_stContOpt.iNegAngleB =
   m_CContour.GetNegAngleB();
m_stContOpt.iNegAngleG =
   m_CContour.GetNegAngleG();
```

Example (cont.)

```
m_stContOpt.iPosAngleA =
   m_CContour.GetPosAngleA();
m_stContOpt.iPosAngleB =
   m_CContour.GetPosAngleB();
m_stContOpt.iPosAngleG =
   m_CContour.GetPosAngleG();

mCContour.Set3DRotation(m_stContOpt.
   b3DRotation);
mCContour.SetExtendedClassification
   (m_stContOpt.bExtendedClassification);
mCContour.SetAffineWindow(m_stContOpt.
   iAffineWindow);
mCContour.SetRotationWindow(m_stContOpt.
   iRotationWindow);
mCContour.SetAngleDelimiting(m_stContOpt.
   bAngleDelimiting);
mCContour.SetEuclidianWindow(m_stContOpt.
   iEuclidianWindow);
mCContour.SetComparisonDepth(m_stContOpt.
   iComparisonDepth);
mCContour.SetScale(m_stContOpt.
   fScaleMin, m_stContOpt.fScaleMax);
mCContour.SetCenterAngleA(m_stContOpt.
   iCenterAngleA);
mCContour.SetCenterAngleB(m_stContOpt.
   iCenterAngleB);
mCContour.SetCenterAngleG(m_stContOpt.
   iCenterAngleG);
mCContour.SetNegAngleA(m_stContOpt.
   iNegAngleA);
mCContour.SetNegAngleB(m_stContOpt.
   iNegAngleB);
mCContour.SetNegAngleG(m_stContOpt.
   iNegAngleG);
```

Example (cont.)

```
mCContour.SetPosAngleA(m_stContOpt.
   iPosAngleA);
mCContour.SetPosAngleB(m_stContOpt.
   iPosAngleB);
mCContour.SetPosAngleG(m_stContOpt.
   iPosAngleG);
mCContour.SetDelimiterString(m_stContOpt.
   cDelimiterString);
```

SetAngleDelimiting

Syntax

```
void SetAngleDelimiting(bool
   bAngleDelimiting);
```

Include Files C_Contour.h

Description Allows the Contour Classifier tool to use the angle and scale preferences.

Parameters

Name: bAngleDelimiting

Description: When TRUE, the tool uses the angle and scale preferences. When FALSE, the tool does not use the angle and scale preferences.

Notes None

Return Values None

Example See the example on page 352.

SetExtendedClassification

Syntax

```
void SetExtendedClassification(bool
   bExtendedClassification);
```

Include Files C_Contour.h

7

Description Allows the Contour Classifier tool to use extended classification.

Parameters

Name: bExtendedClassification

Description: When TRUE, the tool uses extended classification. When FALSE, the tool does not use extended classification.

Notes This option is available only if three-dimensional processing is turned on. Refer to page 352 for more information.

Return Values None

Example See the example on page 352.

SetComparisonDepth

Syntax
```
bool SetComparisonDepth
   (int iComparisonDepth);
```

Include Files C_Contour.h

Description Specifies how many contours in the post-processing stage are used for the comparison/search operation.

Parameters

Name: iComparisonDepth

Description: The number of contours used in the extended classification. Values range from 1 to 5; 1 is the default.

Notes Larger values for *iComparisonDepth* extend the processing time.

Return Values

TRUE The value was between 1 and 5.

FALSE The value was invalid.

Example See the example on page 352.

SetScale

Syntax

```
bool SetScale(float fScaleMin, float
    fScaleMax);
```

Include Files C_Contour.h

Description Specifies how much to scale the contours under test.

Parameters

Name: fScaleMin

Description: The minimum scale factor. Values range from 0.01 to 5; 1.0 is the default.

Name: fScaleMax

Description: The maximum scale factor. Values range from 0.01 to 5; 1.0 is the default.

Notes None

Return Values

TRUE The values are valid and *fScaleMin* is less than *fScaleMax*.

FALSE Value is invalid.

Example If a contour under test is twice as large as the contour in the catalog, and *fScaleMin*=0.1 and *fScaleMax*=1.5, then the contour is not classified.

See the example on page 352 for an example of using this method.

SetCenterAngleA

Syntax `bool SetCenterAngleA(int iCenterAngleA);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the center of the alpha angle.

Parameters

Name: iCenterAngleA

Description: The number of degrees of rotation for the center of alpha angle. Values range from –180 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

SetCenterAngleB

Syntax `bool SetCenterAngleB(int iCenterAngleB);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the center of the beta angle.

Parameters

Name: iCenterAngleB

Description: The number of degrees of rotation for the center of the beta angle. Values range from –180 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

7

SetCenterAngleG

Syntax `bool SetCenterAngleG(int iCenterAngleG);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the center of the gamma angle.

Parameters

Name: iCenterAngleG

Description: The number of degrees of rotation for the center of the gamma angle. Values range from -180 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

SetNegAngleA

Syntax `bool SetNegAngleA(int iNegAngleA);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the negative-going part of the alpha angle.

Parameters

Name: iNegAngleA

Description: The number of degrees of rotation for the negative-going part of the alpha angle. Values range from -180 to 0; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

SetNegAngleB

Syntax `bool SetNegAngleB(int iNegAngleB);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the negative-going part of the beta angle.

Parameters

Name: iNegAngleB

Description: The number of degrees of rotation for the negative-going part of the beta angle. Values range from -180 to 0; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

7

Return Values

TRUE	The value is valid.
FALSE	A value is invalid.

Example See the example on page 352.

SetNegAngleG

Syntax `bool SetNegAngleG(int iNegAngleG);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the negative-going part of the gamma angle.

Parameters

Name: iNegAngleG

Description: The number of degrees of rotation for the negative-going part of the gamma angle. Values range from –180 to 0; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE	The value is valid.
FALSE	A value is invalid.

Example See the example on page 352.

SetPosAngleA

Syntax `bool SetPosAngleA(int iPosAngleA);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the positive-going part of the alpha angle.

Parameters

Name: iPosAngleG

Description: The number of degrees of rotation for the positive-going part of the alpha angle. Values range from 0 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

SetPosAngleB

Syntax `bool SetPosAngleB(int iPosAngleB);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the positive-going part of the beta angle.

Parameters

Name: iPosAngleB

Description: The number of degrees of rotation for the positive-going part of the beta angle. Values range from 0 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

SetPosAngleG

Syntax `bool SetPosAngleG(int iPosAngleG);`

Include Files C_Contour.h

Description Specifies the number of degrees of rotation for the positive-going part of the gamma angle.

Parameters

Name: iPosAngleG

Description: The number of degrees of rotation for the positive-going part of the gamma angle. Values range from 0 to 180; 0 is the default.

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values

TRUE The value is valid.

FALSE A value is invalid.

Example See the example on page 352.

Get3Drotation

Syntax `void Get3DRotation(bool b3DRotation);`

Include Files C_Contour.h

Description Returns whether the contours under test can rotate in three-dimensions.

Parameters

Name: b3DRotation

Description: When TRUE, three-dimensional rotation is allowed. When FALSE, three-dimensional rotation is not allowed.

Notes The tool performs additional computations to classify three-dimensional contours.

Return Values None

Example See the example on page 352.

7

GetAngleDelimiting

Syntax `bool GetAngleDelimiting(void);`

Include Files C_Contour.h

Description Returns whether the angle and scale preferences are used.

Parameters None

Notes None

Return Values

TRUE Uses angle and scale preferences.

FALSE Ignores angle and scale preferences.

Example See the example on page 352.

GetExtendedClassification

Syntax `bool GetExtendedClassification(void);`

Include Files C_Contour.h

Description Returns whether extended classification is used.

Parameters None

Notes This option is available only if three-dimensional processing is turned on. Refer to page 352 for more information.

Return Values

TRUE The tool uses extended classification.

FALSE The tool does not use extended classification.

Example See the example on page 352.

GetComparisonDepth

Syntax `int GetComparisonDepth(void);`

Include Files C_Contour.h

Description Returns the number of contours that are used for the comparison/search operation.

Parameters None

Notes Larger values for *iComparisonDepth* extend the processing time.

Return Values The number of contours used in the extended classification. Values range from 1 to 5; 1 is the default.

Example See the example on page 352.

GetScaleMin

Syntax `float GetScaleMin(void);`

Include Files C_Contour.h

Description Returns the minimum scale factor for the contours under test.

Parameters None

Notes None

Return Values The minimum scale factor. Values range from 0.01 to 5; 1.0 is the default.

Example See the example on page 352.

GetScaleMax

Syntax `float GetScaleMax(void);`

Include Files C_Contour.h

Description Returns the maximum scale factor for the contours under test.

Parameters None

Notes None

Return Values The maximum scale factor. Values range from 0.01 to 5; 1.0 is the default.

Example See the example on page 352.

GetCenterAngleA

Syntax `int GetCenterAngleA(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the center of the alpha angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the center of alpha angle. Values range from -180 to 180; 0 is the default.

Example See the example on page 352.

GetCenterAngleB

Syntax `int GetCenterAngleB(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the center of the beta angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the center of the beta angle. Values range from −180 to 180; 0 is the default.

Example See the example on page 352.

GetCenterAngleG

Syntax `int GetCenterAngleG(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the center of the gamma angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the center of the gamma angle. Values range from -180 to 180; 0 is the default.

Example See the example on page 352.

GetNegAngleA

Syntax `int GetNegAngleA(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the negative-going part of the alpha angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the negative-going part of the alpha angle. Values range from -180 to 0; 0 is the default.

Example See the example on page 352.

GetNegAngleB

Syntax `int GetNegAngleB(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the negative-going part of the beta angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the negative-going part of the beta angle. Values range from –180 to 0; 0 is the default.

Example See the example on page 352.

GetNegAngleG

Syntax `int GetNegAngleG(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the negative-going part of the gamma angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the negative-going part of the gamma angle. Values range from –180 to 0; 0 is the default.

Example See the example on page 352.

GetPosAngleA

Syntax `int GetPosAngleA(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the positive-going part of the alpha angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the positive-going part of the alpha angle. Values range from 0 to 180; 0 is the default.

Example See the example on page 352.

GetPosAngleB

Syntax `int GetPosAngleB(void);`

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the positive-going part of the beta angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the positive-going part of the beta angle. Values range from 0 to 180; 0 is the default.

Example See the example on page 352.

GetPosAngleG

Syntax
```
int GetPosAngleG(void);
```

Include Files C_Contour.h

Description Returns the number of degrees of rotation for the positive-going part of the gamma angle.

Parameters None

Notes Contours which fall within this number of degrees are classified before contours that are not within this number of degrees.

Refer to Figure 2 on page 348 for an illustration of this angle.

Return Values The number of degrees of rotation for the positive-going part of the gamma angle. Values range from 0 to 180; 0 is the default.

Example See the example on page 352.

SetDelimiterString

Syntax
```
bool SetDelimiterString(char
   *cDelimiterString);
```

Include Files C_Contour.h

Description Specifies the delimiter for the string that is supplied to the Contour Classifier tool and for the strings that are returned from the Contour Classifier tool.

Parameters

Name: cDelimiterString

Description: A pointer to a delimiter string.

Notes The delimiter can be any character or multiple characters.

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example See the example on page 352.

CleanIUTList

Syntax `void CleanIUTList(void);`

Include Files C_Contour.h

Description Clears the list of contours under test.

Parameters None

Notes None

Return Values None

Example `CleanIUTList();`

CleanCatalog

Syntax `void CleanCatalog(void);`

Include Files C_Contour.h

Description Clears the list of contours in the catalog.

Parameters None

Notes None

Return Values None

Example `CleanCatalog();`

GetResultsCount

Syntax `int GetResultsCount(void);`

Include Files C_Contour.h

Description Returns the number of results that are available.

Parameters None

Notes None

Return Values The number of results.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
int iNumOfResults;

iNumOfResults =
   m_CContour.GetResultsCount();
```

GetCatalogCount

Syntax `int GetCatalogCount(void);`

Include Files C_Contour.h

Description Returns the number of elements that are in the catalog.

Parameters None

Notes None

Return Values The number of elements in the catalog.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
int iNumOfElements;

iNumOfElements =
   m_CContour.GetCatalogCount();
```

GetIUTCount

Syntax

```
int GetIUTCount(void);
```

Include Files C_Contour.h

Description Returns the number of contours that are under test.

Parameters None

Notes None

Return Values The number of contours under test.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
int iNumOfElements;

iNumOfElements =
   m_CContour.GetIUTCount();
```

SupplyContour

Syntax

```
bool SupplyContour(CcROIBase *CRoi);
```

Include Files C_Contour.h

Description Adds a contour to the list of contours under test.

Parameters

Name: CRoi

Description: A pointer to an ROI that describes the contour.

Notes The ROI must be a freehand type.

Return Values

TRUE The operation was successful.

FALSE The operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;
CcROIBase *pCRoi;

bResult = m_CContour.SupplyContour(pROI);
```

GetError

Syntax

```
int GetError(void);
```

Include Files C_Contour.h

Description If an error occurs during the **SupplyContour** method, returns the error code.

Parameters None

Notes None.

Return Values The error code from the header file.

#define E_SLIVER 0x0001 Supplied contour has a pixel width or height of 1.

#define E_NUM_PIX_EXCEEDED 0x0002 Supplied contour has more than 4096 points. 4096 is the maximum number of points for the input contours.

7

Return Values (cont.)

The error code from the header file.

#define E_NULL_POINTER 0x0003 — An invalid input ROI was specified.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
int iError;

iError = m_CContour.GetError();
```

BuildCatalog

Syntax `bool BuildCatalog(void);`

Include Files C_Contour.h

Description Adds contours to the catalog.

Parameters None

Notes Use this method after you add the contour to the contours under test using **SupplyContour** (see page 376).

Return Values

TRUE — Operation was successful.

FALSE — Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;

bResult = m_CContour.BuildCatalog();
```

RebuildCatalog

Syntax
```
bool RebuildCatalog(void);
```

Include Files C_Contour.h

Description Reinitializes the existing catalog.

Parameters None

Notes Use this method after you change any of the parameters for the contour classification.

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example
```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;

bResult = m_CContour.RebuildCatalog();
```

NameCatalogElements

Syntax
```
bool NameCatalogElements(char *pcString,
   int iStartIndex);
```

Include Files C_Contour.h

Description Names or renames the catalog elements, beginning at a specified element in the catalog.

Parameters

Name: pcString

Description: String with the elements separated by the delimiters.

Name: iStartIndex

Description: Index of the element from which to start naming.

7

Notes The element names should be separated by the delimiter that was specified by the **SetDelimiterString** method.

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;
char cNames[50];

//Assume there are 3 elements in
//the catalog
strcpy(cNames, “A,B,C”);

//Name all the elements starting
//from element 0
bResult = m_CContour.NameCatalogElements(
   cNames, 0);
```

GetCatalogString

Syntax `char *GetCatalogString(void);`

Include Files C_Contour.h

Description Returns a pointer to an allocated string that is filled with the names of the catalog elements.

Parameters None

Notes You must deallocate the string once you have used it.

Return Values

Pointer to a string. Operation succeeded. The string contains the element names separated by the delimiter.

NULL Unable to return the string pointer.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
char *pcString;

pcString = m_CContour.GetCatalogString(
   void);
```

SaveCatalog

Syntax `bool SaveCatalog(char *cFileName);`

Include Files C_Contour.h

Description Saves the catalog to a file.

Parameters

Name: cFilename

Description: Name of the file in which to store the catalog.

Notes None

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;
char cMydata[200];

bResult = m_CContour.SaveCatalog(
   cMydata);
```

LoadCatalog

Syntax `bool LoadCatalog(char *cFileName);`

Include Files C_Contour.h

Description Loads the catalog from a file.

Parameters

Name: cFilename

Description: Name of the file from which to load the catalog.

Notes None

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
bool bResult;
char cMydata[200];

bResult = m_CContour.LoadCatalog(
   cMydata);
```

ClassifyContours

Syntax

```
void ClassifyContours(void);
```

Include Files C_Contour.h

Description Classifies contours by trying to match contours under test with contours stored in the catalog.

Parameters None

Notes None

Return Values None

Example

```
//Instantiate the contour class
CcContour   m_CContour;

m_CContour.ClassifyContours();
```

GetResult

Syntax

```
STCATRESULT *GetResult(int iIndex);
```

Include Files C_Contour.h

Description Returns classification information starting from a specified element in the results list.

Parameters None

Notes The first element in a new list is 0.

The STCATRESULT structure, which is defined as follows:

```
struct stResultTag
{
//Name of matching element
CString CElementName;
//Match confidence measure
double dScore;
```

7

Notes (cont.)

```
//Scale
float fScale;
//Rotation in the XY Plane
float fAlpha;
//Rotation in the ZX Plane
float fBeta;
//Rotation in the YZ Plane
float fGamma;
}
typedef stResultTag STCATRESULT;
```

Return Values

Pointer to a results structure.	Operation succeeded.
NULL	Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
STCATRESULT *pstResult;

//Get the results for the third contour
pstResult = m_CContour.GetResult(3);
```

MakeImageOfCATList

Syntax

```
CcImage *MakeImageOfCATList(void);
```

Include Files C_Contour.h

Description Returns an 8-bit grayscale bitmap image containing all the catalog elements.

Parameters None

Notes This method is meant to be used in a graphical user interface. You are responsible for deallocating this image.

Return Values

Pointer to an image.	Operation was successful.
NULL	Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
CcImage *pImage;

//Get the results for the third contour
pImage = m_CContour.MakeImageOfCATList();
```

MakeImageOfIUTList

Syntax `CcImage *MakeImageOfIUTList(void);`

Include Files C_Contour.h

Description Returns an 8-bit grayscale bitmap image containing all the contours under test.

Parameters None

Notes This method is meant to be used in a graphical user interface. You are responsible for deallocating this image.

Return Values

Pointer to an image.	Operation was successful.
NULL	Operation failed.

Example

```
//Instantiate the contour class
CcContour   m_CContour;
CcImage *pImage;

//Get the results for the third contour
pImage = m_CContour.MakeImageOfIUTList();
```

8

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 DT Vision Foundry API, described in Chapter 2 starting on page 11. The DT Vision Foundry 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 in other ways. 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 390); 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:

8

```
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 DT Vision Foundry Miscellaneous toolbar, 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 20 lists the math operators used by the Custom Script tool.

Table 20: 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 21 shows examples for each of these operators.

Table 21: Examples of Bitwise Operators

<table>
<tr><th>Bitwise Operator</th><th>Example</th></tr>
<tr><td>&</td><td>9 & 5 = 1</td></tr>
<tr><td>|</td><td>3 | 8 = 11</td></tr>
<tr><td>@</td><td>11 @ 3 = 8</td></tr>
<tr><td>SHR</td><td>64 SHR 2 = 16</td></tr>
<tr><td>SHL</td><td>8 SHL 3 = 64</td></tr>
</table>

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 395) are assigned a precedence level. When processing a calculation, the significance or precedence determines the order of solution of each individual operator. Table 22 lists the five levels of precedence and the operators that are contained within each level.

Table 22: Operator Precedence

<table>
<tr><th>Level of Precedence</th><th colspan="6">Operator</th></tr>
<tr><td>Level 1</td><td>~</td><td>NOT</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Level 2</td><td>+</td><td>-</td><td>*</td><td>/</td><td>^</td><td></td></tr>
<tr><td>Level 3</td><td>&</td><td>|</td><td>@</td><td></td><td></td><td></td></tr>
<tr><td>Level 4</td><td><</td><td>></td><td>=</td><td><=</td><td>>=</td><td><></td></tr>
<tr><td>Level 5</td><td>AND</td><td>OR</td><td>SHR</td><td>SHL</td><td>IN</td><td></td></tr>
</table>

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 23 lists the logical operators used in the Custom Script tool.

Table 23: 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 22 on page 395, 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 24: 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
```

8

```
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 function 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 390).

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
```

8

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$** returns the current system date in text or STRING form. The format is always year, month, and day. For example, 20000101 is Jan 1, 2000. The STRING or text form of time is **TIME$**. The clock is a 24-hour clock, so 1 p.m. is hour 13. The format is: hhmmss. In other words, the text for 2:45 pm is 144500.

The numeric form for these two keywords are **DATE** and **TIME**. The date is returned in the form yyyymmdd, such as 20000101 for Jan 1, 2000. The time is returned in the form hhmmss, such as 144500 for 2:45 pm.

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 25 briefly describes the keywords and functions used in the Custom Script tool.

Table 25: Keywords and Functions of the Custom Script Tool

Function Type	Function Name	Description
Math Functions	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.
String Functions	CHR	Converts a number to a text character.
	IN	Determines if one text string is contained in another string.

Table 25: Keywords and Functions of the Custom Script Tool

Function Type	Function Name	Description
String Functions (cont.)	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 and Time Functions	DATE	Returns the system's date in numeric format.
	DATE$	Returns the system's date in text or string format.
	TIME	Returns the system's time in numeric format
	TIME$	Returns the system's time in text or string format.
File Functions	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 25: Keywords and Functions of the Custom Script Tool

Function Type	Function Name	Description
Program Flow Control Functions	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 Functions	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.

Comments

The variable in which the data is returned must be defined as a number.

Example

```
D=DATE
! JANUARY 1, 2000 IS
! RETURNED AS D=20000101
```

DATE$

Syntax

```
S=DATE$
```

Description

Returns the system date in the text format *yyyymmdd*, where *yyyy* represents the year, *mm* represents the month, and *dd* represents the day.

Comments

The variable in which the data is returned must be defined as a string.

Example

```
S=DATE$
! RETURNS THE DATE JANUARY 1, 2000
! AS A STRING
```

TIME

Syntax
`TIME`

Description
Returns the system time in the numeric format *hhmmss,* where *hh* represents the hour, *mm* represents the minute, and *ss* represents the number of seconds.

Comments
The variable in which the data is returned must be defined as a number.

Example
```
S=TIME
! PLACE THE CURRENT TIME VALUE IN
! A NUMERIC VARIABLE.
```

TIME$

Syntax
`TIME$`

Description
Returns the system time in the following text format: *hhmmss,* where *hh* represents the hour, *mm* represents the minute, and *ss* represents the number of seconds.

Comments
The variable in which the data is returned must be defined as a string.

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
```

8

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
```

Example (cont.)

```
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)
!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
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
```

9

Using the Data Matrix Reader Tool API

Overview of the Data Matrix Reader Tool API

The Data Matrix Reader tool operates on an image or ROI containing a two-dimensional data matrix symbol of type ECC200. This tool contains two classes: the CcDMCode class and the CcDMReader class.

Note: The Data Matrix Reader tool requires MFC (Microsoft Foundation Class) support.

The CcDMCode class allows you to define the data matrix symbol you are looking for and return information about that symbol. The image and the ROI, if an ROI is defined, are passed to the **Read** method in the CcDMReader class. This method locates and decodes two-dimensional data matrix symbols in the ROI.

Note: An ROI is not required. However, if an ROI is used, it must be a rectangle ROI.

The class methods are listed in Table 26.

Table 26: CcDMCode and CcDMReader Class Methods

Method Class	Method Category	Method Name
CcDMCode Methods	Initialization	void CcDMCode::Initialize(BOOL ClearSA)
	Sets Values	BOOL CcDMCode::SetAutoSize( )
		BOOL CcDMCode:: SetContrast (int contrast)
		BOOL CcDMCode::SetMinModuleSize(int size)
		BOOL CcDMCode::SetSize(CSize size) BOOL CcDMCode::SetSize(int rows, int cols)
		BOOL CcDMCode::SetTimeout(int timeout)
	Returns Values	double CcDMCode::GetAngle()
		CPoint CcDMCode::GetCenter()
		int CcDMCode::GetCodeFileID()
		int CcDMCode::GetContrast()
		void CcDMCode::GetCorners(CPoint corners[4]) void CcDMCode::GetCorners(POINT corners[4])
		int CcDMCode::GetError()
		int CcDMCode::GetErrorCount()
		int CcDMCode::GetExecTime()
		int CcDMCode::GetFNC1()
		int CcDMCode::GetMinModuleSize(int size)
		CSize CcDMCode::GetModuleSize()
		int CcDMCode::GetProgress()
		int CcDMCode::GetSAInfo(int* n, int* m)
		CSize CcDMCode::GetSize()
		char* CcDMCode::GetText()

Table 26: CcDMCode and CcDMReader Class Methods (cont.)

Method Class	Method Category	Method Name
CcDMCode Methods (cont.)	Returns Values (cont.)	int CcDMCode::GetTextLen()
		double CcDMCode::GetTimeout()
		BOOL CcDMCode::IsSASetComplete()
		CString CcDMCode::ReportError()
CcDMReader Methods	Reads Symbols	int CcDMRead::Read (CcImage* image, CcRoiBase* roi, CcDMCode* code)

CcDMCode Methods

This section describes the methods of the CcDMCode class in detail.

SetAutoSize

Syntax

```
BOOL CcDMCode::SetAutoSize(
   );
```

Include File CDMCode.h

Description Uses built-in algorithms to automatically find the size of the two-dimensional data matrix symbol.

Parameters None

Notes By default, the size is automatically measured.

Calling this method cancels the effect of a prior call to the **SetSize** method, described on page 441.

Return Values

0 The method failed.

1 The method was successful.

SetContrast

Syntax

```
BOOL CcDMCode::SetContrast(
   int contrast
   );
```

Include File CDMCode.h

Description Sets the contrast value used in edge detection when searching for a two-dimensional data matrix symbol.

Parameters

Name: contrast

Description: Grayscale gradient threshold. Ranges from 1 to 255. The default value is 20.

Notes

It may be desirable to reduce the value for very low contrast images or images with blurred edges. When working with noisier and more textured images where background features may be more distinct than the two-dimensional data matrix symbol, it may be desirable to increase the value. Larger values require steeper edge gradients before an edge is detected.

The effect of some example values is as follows:

255		
38		
102		
12		

See also **GetContrast**, described on page 445.

Return Values

0 The method failed.

1 The method was successful.

SetMinModuleSize

Syntax

```
BOOL CcDMCode::SetMinModuleSize(
   int size
   );
```

Include File CDMCode.h

Description Provides an estimate of the minimum module size, in pixels.

Parameters

Name: size

Description: The minimum module size, in pixels. Ranges from 2 to 200 pixels.

Notes See also **GetMinModuleSize**, described on page 450.

Return Values

0 The size is out of range.

1 This size is valid.

SetSize

Syntax

```
BOOL CcDMCode::SetSize(
   CSize size
   );
```

or

```
BOOL CcDMCode::SetSize(
   int rows,
   int cols,
   );
```

Include File CDMCode.h

Description Sets the size of the two-dimensional data matrix symbol. This is used only in cases where the pitch modules are of poor quality and automatic detection fails. The default is to automatically determine the size.

Parameters

Name: size

Description: size.cx is the number of columns; size.cy is the number of rows.

Name: rows

Description: The number of rows.

Name: cols

Description: The number of columns.

Notes See also **GetSize**, described on page 453, and **SetAutoSize**, described on page 439.

Return Values

0 The size is invalid.

1 This size is valid.

SetTimeout

Syntax

```
BOOL CcDMCode::SetTimeout(
   int timeout
   );
```

Include File CDMCode.h

Description Sets the timeout value, in milliseconds. The timeout condition is checked at various points during the read operation. If a timeout condition is detected, the read operation is aborted and the error status is set to IDS_ERR_TIMEOUT.

Parameters

Name: timeout

Description: The maximum number of milliseconds allowed after the read operation starts and before the operation times out. The valid range is 1 to 30000 milliseconds (or 5 minutes).

Notes See also **GetTimeout**, described on page 455.

Return Values

0 The method failed.

1 The method was successful.

GetAngle

Syntax

```
double CcDMCode::GetAngle(
   );
```

Include File CDMCode.h

Description Reports the code axis angle with respect to the image axis. The value is given in radians and specifies the angle to the base side of the "L" (with respect to the x-axis) that defines the border of the two-dimensional data matrix symbol. A value of 0 indicates that the base is parallel to the x-axis and is pointing to the right. Angle values increase in a counterclockwise direction.

Parameters None

Notes None

Return Values The angle, in radians.

GetCenter

Syntax
```
CPoint CcDMCode::GetCenter(
   );
```

Include File CDMCode.h

Description Reports the center position of the two-dimensional data matrix symbol, in pixels. The position given is the mean of the x- and y-ordinates of the four corners.

Parameters None

Notes None

Return Values The pixel position of the center of the code, relative to the image.

GetCodeFileID

Syntax
```
int CcDMCode::GetCodeFileID(
   );
```

Include File CDMCode.h

Description Returns the file identifier for the structured append data set.

Parameters None

Notes None

Return Values

File identifier	The code is part of a structured append set.
-1	The code is not part of a structured append set.

GetContrast

Syntax

```
int CcDMCode::GetContrast(
   );
```

Include File CDMCode.h

Description Returns the current contrast value that was set with **SetContrast**.

Parameters None

Notes See also **SetContrast**, described on page 439.

Return Values The contrast value; values range from 1 to 255.

GetCorners

Syntax

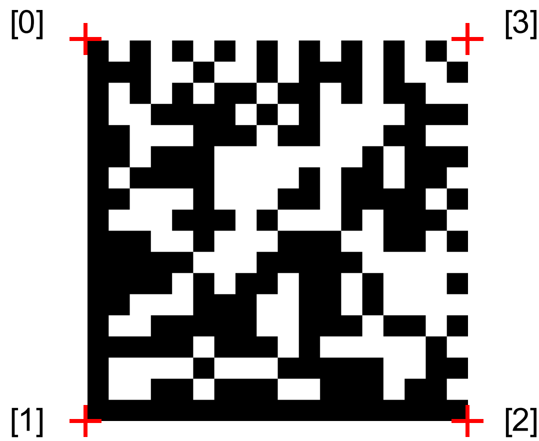
```
void CcDMCode::GetCorners(
   CPoint corners[4]
   );
```

or

```
void CcDMCode::GetCorners(
   POINT corners[4]
   );
```

Include File CDMCode.h

Description Returns the four corner points of the two-dimensional data matrix symbol, as follows:



Parameters None

Name: corners

Description: The position of each corner, where corner[0] is the top of the vertical side of the "L" that defines the border of the two-dimensional data matrix symbol; corner[1] is the left base corner; corner[2] is the right base corner; corner[3] is the top right corner; and corner[4] is the right top corner, irrespective of rotation.

Notes None

Return Values None

GetError

Syntax

```
int CcDMCode::GetError(
   );
```

Include File CDMCode.h

Description Returns the numerical error code for the error. This error is the last recorded error. The read operation may have reached a more advanced stage before recording the final error code.

Parameters None

Notes See also **GetProgress**, described on page 451.

Return Values

9001 – IDS_ERR_NOMEMORY	Memory allocation error.
9002 –IDS_ERR_REEDSOL	Error correction error – too many errors to correct.
9003 –IDS_ERR_SCHEME	Invalid data encoding.
9004 – IDS_ERR_INVCELLNUM	A possible code has been located and the autosize scan has completed, but the size is not valid.
9005 – IDS_ERR_MODULE_SIZE	Module size measured during the autosize scan is invalid.
9006 –IDS_ERR_SCANERR	ROI access error while scanning for the "L" that defines the border of the two-dimensional data matrix symbol.
9007 –IDS_ERR_ROI_SIZE	The supplied ROI (or image) is too small to contain a code.
9008 –IDS_ERR_ROI_TYPE	The supplied ROI is not a supported type.
9009 –IDS_ERR_TIMEOUT	The read operation has timed out.

Return Values (cont.)

9010 –IDS_ERR_NOCODE	No possible code detected.
9011 –IDS_ERR_SASET	Structured append values are invalid (not in the range 1-16 of 1-16).
9012 –IDS_ERR_SAFILE	Stuctured append code's file ID does not match that of the seed CcCode object. It is assumed that the new code is not part of the set. If no seed ID is supplied (==0), then no error occurs.
9013 –IDS_ERR_SADUP	Duplicate structured append code. The seed code object has already logged a symbol with the new code index *n* of *m*.
9014 –IDS_ERR_SAM	Set size mismatch for the structured append code. The seed object has a different number of symbols in the set than the newly read code.

GetErrorCount

Syntax

```
int CcDMCode::GetErrorCount(
   );
```

Include File CDMCode.h

Description Returns the number of errors that were found and corrected in the read operation.

Parameters None

Notes One code word corresponds to eight modules in the matrix.

Return Values

0	No errors were found.
-1	Error correction failed due to too many errors.
The number of data code words (bytes) that contained errors.	Errors are detected and corrected.

GetExecTime

Syntax

```
int CcDMCode::GetExecTime(
   );
```

Include File CDMCode.h

Description Returns the time taken to read the code, in milliseconds.

Parameters None

Notes None

Return Values The time in milliseconds that it took to read the code.

GetFNC1

Syntax

```
int CcDMCode::GetFNC1(
   );
```

Include File CDMCode.h

Description Returns the symbology identifier option value.

Parameters None

Notes **GetText**, described on page 454, returns the symbology identifier as part of the data.

9

Return Values

-1 (no identifier)	Symbology ID not enabled.
1 (]d1 identifier)	ECC200.
2 (]d2 identifier)	ECC200, FNC1 1st or 5th.
3 (]d3 identifier)	ECC200, FNC1 2nd or 6th.
4 (]d4 identifier)	ECC200, with ECI protocol.
5 (]d5 identifier)	ECC200, FNC1 1st or 5th with ECI.
6 (]d6 identifier)	ECC200, FNC1 2nd or 6th with ECI.

GetMinModuleSize

Syntax

```
int CcDMCode::GetMinModuleSize(
   );
```

Include File CDMCode.h

Description Returns the current minimum module size.

Parameters None

Notes Refer to the **SetMinModuleSize** method, described on page 441, for more information.

Return Values The current minimum module size.

GetModuleSize

Syntax

```
CSize CcDMCode::GetModuleSize(
   );
```

Include File CDMCode.h

Description Reports the mean pixel width and height of a module as measured.

Parameters None

Notes This method can give useful feedback for fine-tuning the *MinModule* value. The values for width and height are aligned with the symbol sides rather than with the image pixel grid.

Return Values The mean module size of the two-dimensional data matrix symbol that was found.

GetProgress

Syntax

```
int CcDMCode::GetProgress(
   );
```

Include File CDMCode.h

Description Returns the numerical progress value for the read operation. This value represents the most advanced stage reached in the read operation.

For example, the read operation may locate the sides of what seems to be a code but which has an invalid size format. The read attempt continues and may record the last error as IDS_ERR_NOCODE if the final scan misses the symbol. The progress code contains the best stage reached.

Parameters None

Notes None

Return Values

IDS_PRG_NONE	No progress has been made.
IDS_PRG_1ST_EDGE	An edge has been found while scanning for the "L" that defines the border of the two-dimensional data matrix symbol.
IDS_PRG_VL_EDGE	A possible vertical side of an "L" that defines the border of the two-dimensional data matrix symbol has been found.
IDS_PRG_BOX_CODE	Four sides of a code have been located.
IDS_PRG_MEAS_MODS	Pitch modules have been measured.
IDS_PRG_VAL_SIZE	A valid code size has been detected.
IDS_PRG_MOD_SAMP	The data modules have been sampled.
IDS_PRG_BYTES	Byte data has been extracted from the sampled module data.
IDS_PRG_ECC	Error correction has succeeded.
IDS_PRG_DEC	Data has been decoded.

GetSAInfo

Syntax

```
int CcDMCode::GetSAInfo(
   int *n,
   int *m
   );
```

Include File CDMCode.h

Description If the code is part of a structured append data set, this method returns the position of the symbol in the set and the total number of symbols in the set.

Parameters

Name: *n

Description: As an input parameter, a pointer to an integer that receives that symbol index.

As an output parameter, the index of this symbol within the *m* total symbols of the set.

Name: *m

Description: As an input parameter, a pointer to an integer that receives the total number of symbols in the set.

As an output parameter, the total number of symbols in the set.

Notes None

Return Values The number of symbols decoded so far, which indicates the reading progress.

m The entire structured append data set has been read.

0 The code is not a symbol in the structured append data set.

GetSize

Syntax

```
CSize CcDMCode::GetSize(
    );
```

Include File CDMCode.h

Description Returns the size of the symbols as the number of modules (such as 32 x 32).

Parameters None

Notes If this method is called after calling **SetSize**, described on page 441, and before calling **CcDMRead::Read( )**, described on page 458, the values (column and row) set by **SetSize** are returned in the CSize structure. After a read operation, the detected values (column and row) are returned in the CSize structure.

Return Values The size of the symbol in the modules.

GetText

Syntax
```
char* CcDMCode::GetText(
    );
```

Include File CDMCode.h

Description Returns the decoded data string as ASCII text.

Parameters None

Notes The text includes special codes and headers, if they are specified by ECI, FNC1, or macro controls, which are embodied in the code bitstream.

Return Values The decoded text.

GetTextLen

Syntax
```
int CcDMCode::GetTextLen(
    );
```

Include File CDMCode.h

Description Returns the length of the decoded data string.

Parameters None

Notes None

Return Values The number of bytes in the decoded text/data, not including terminating NULLs.

GetTimeout

Syntax

```
int CcDMCode::GetTimeout(
   );
```

Include File CDMCode.h

Description Returns the current timeout value, in milliseconds.

Parameters

Notes See also **SetTimeout**, described on page 442.

Return Values The timeout value.

Initialize

Syntax

```
void CcDMCode::Initialize(
   BOOL ClearSA
   );
```

Include File CDMCode.h

Description Resets the code object to a default state.

Parameters

Name: ClearSA

Description: Determines whether to reset the structured append data set. A value of 0 leaves the structured append data set unchanged. A value of 1 resets the structured append data set.

Notes None

Return Values None

IsSASetComplete

Syntax

```
BOOL CcDMCode::IsSASetComplete(
    );
```

Include File CDMCode.h

Description Tests the completion state for the structured append data set that is defined by this code object.

Parameters None

Notes None

Return Values

0 All the symbols in the set have not been decoded.

1 All symbols in the set have been decoded.

ReportError

Syntax

```
CString CcDMCode::ReportError(
    );
```

Include File CDMCode.h

Description Returns any errors that occur during the read operation, in descriptive format. The text corresponding to error codes is loaded from the string table.

Parameters None

Notes

Error messages are in the form: “Error message [Progress report]”

The following is an example of an error message that may be returned:
“Invalid data detected while decoding encodation [Error correction successful]”

Return Values

The error message.

CcDMReader Methods

This section describes the **Read** method of the CcDMReader class in detail.

Read

Syntax

```
int CcDMRead::Read(
   CcImage* image,
   CcRoiBase* roi,
   CcDMCode* code
   );
```

Include File C_DMRead.h

Description Reads a two-dimensional data matrix symbol.

Parameters

Name: image

Description: A pointer to a Data Translation CcImage class object.

Name: roi

Description: A pointer to a region of interest that defines the code search region. If NULL, the entire image is searched.

Name: code

Description: A CcCode object that is populated with decoded information. This is a returned value.

Notes Currently, only rectangle ROI can be used.

The ROI bounding box should be larger than the code object.

Return Values

The return value indicates whether the read operation was successful or whether an error occurred.

Result states are bit flags. Test for data available with retcode & DM_OK. Test for structured append data sets with retcode & DM_MULTIPLE, and so on.

Note that a return of DM_CODE_ERROR can be tested as retcode == DM_CODE_ERROR or retcode & DM_CODE_ERROR. Call **GetError**, described on page 447, or **ReportError**, described on page 456, to get more information on the nature of the error. If GetError() == IDS_ERR_SCHEME, invalid data was encountered in the final stages of the read operation. You can retrieve the data up to the position where the error occurred with **GetText**, described on page 454.

DM_OK	Normal read of single code. Data returned.
DM_COMPLETE_SET	Last of a structured append data set decoded. Data returned.
DM_MULTIPLE	More than one nonstructured append symbol in view. First data returned.
DM_CODE_SET	One or more symbols of a structured append set decoded. Only incomplete data available.
DM_CODE_ERROR	Some error occurred in reading the code.

Example Program Using the Data Matrix Reader Tool API

This program shows a basic example of reading a two-dimensional data matrix symbol.

```
CcDMReader Reader;
CcDMCode Code;
CString result;
int ret;

// Set code reading parameters
Code.SetMinModuleSize(5);
Code.SetTimeout(1000);
// Read code
ret = Reader.Read(pImage,pRoi,&Code);
if(ret&DM_OK)
   {
      result = Code.GetText();
      // do something with the result
   }
else if (ret==DM_CODE_ERROR)
   {
      result = Code.GetErrorReport();
      // handle the error
   }
```

If you are working with structured append data sets, the code object holds intermediate data; therefore, you need to pass the same object to successive **Read** methods to allow the full data to be compiled, as shown in this example:

```
CcDMReader Reader;
// Default code object
CcDMCode Code;
CString *result=NULL;
// An array of CStrings to hold each symbol data
CString FinalData;
// To hold combined structured append data sets
int ret;
int SymbolNumber;
int SetSize;
int FileID;

// Set code reading parameters
Code.SetMinModuleSize(5);
Code.SetTimeout(100);
// Read code
ret = Reader.Read(pImage,pRoi,&Code);
if(ret&DM_OK)
   {
      if(ret&DM_CODE_MULTIPLE)
      {
      Code.GetSAInfo(&SymbolNumber,&SetSize);
      if(result==NULL)
         {
            result = new CString[SetSize];
         }
      // Save this result
      result[SymbolNumber]=Code.GetText();
      // Test for complete set read
```

```
            if(ret& DM_COMPLETE_SET)
            // OR if(Code.IsSASetComplete())
               {
                  // Concatenate all the data in number
                  // sequence
                  FinalData = “”; // Clear
                  for(i=0;i<SetSize;i++)
                  {
                     FinalData+=result[i]; // Concatenate
                  }
               // Report the result
               MessageBox(FinalData,”Structured append
                  data”,MB_OK);
               }
            }
         }
      else if (ret==DM_CODE_ERROR)
         {
            result = Code.GetErrorReport();
            // handle the error
         }
```

10

Using the Digital I/O Tool API

Overview of the Digital I/O Tool API

The Digital I/O API has one object only: the CcDigIODevice class. Its method is to control the input and output lines of a specified digital I/O device. The Digital I/O Tool API is intended to be used with the Device Manager API, described on page 206.

The CcDigIODevice class uses a standard constructor and destructor and the class methods listed in Table 27.

Table 27: CcDigIODevice Object Methods

Method Type	Method Type	Method Name
Constructor & Destructor	–	CcDigIODevice( );
	–	~ CcDigIODevice( );
CcDigIODevice Class Methods	Setting Properties	int ClearIntOnChangeConfig ( );
		int EnableAsyncWrite (bool bEnable);
		int EnableCachedWrite (bool bEnable);
		int EnableIntOnChange (int nLine, bool bEnable);
		int EnableLatchedRead (bool bEnable);
		int EnableWaitOnRead (bool bEnable);
		int SetDeviceConfig (LPSTREAM pStream);
		int SetDeviceProperty (int nPropId, int nValue);
		int SetReadTimeout (int nTimeout);
		int ShowDeviceConfigDialog (HWND hParent);

Table 27: CcDigIODevice Object Methods (cont.)

Method Type	Method Type	Method Name
CcDigIODevice Class Methods (cont.)	Retrieving Properties	int GetDeviceCaps (int* pnDeviceCaps);
		int GetDeviceConfig (LPSTREAM pStream);
		int GetDeviceConfigFileDesc (LPSTR szFileDesc, int nBufSize);
		int GetDeviceConfigFileExt (LPSTR szFileExt, int nBufSize);
		int GetDeviceProperty (int nPropId, int* pnValue);
		int GetErrorText (LPSTR szErrorText, int nBufSize);
		int GetInputLineCount (int* pnCount);
		int GetOutputLineCount (int* pnCount);
		int GetReadTimeout (int* pnTimeout);
		bool IsAsyncWriteDone ( );
		bool IsAsyncWriteEnabled ( );
		bool IsCachedWriteEnabled ( );
		bool IsIntOnChangeEnabled (int nLine);
		bool IsLatchedReadEnabled ( );
		bool IsWaitOnReadEnabled ( );
	Controlling the Digital I/O Lines	int ExecuteCachedWrite ( );
		int ExecuteLatchedRead ( );
		int ReadInputLine (int nLine, bool* pbLineState);
		int WriteOutputLine (int nLine, bool bLineState, int PulseWidth);

Description of CcDigIODevice Methods

This section provides a detailed description of each method of this class.

ClearIntOnChangeConfig

Syntax

```
int ClearIntOnChangeConfig (
   );
```

Include File C_digitalio.h

Description Clears (disables) the current interrupt-on-change configuration for all digital input lines.

Parameters None

Notes None

Returned Value

< 0 The method was unsuccessful.

0 The method was successful.

Example

```
 //Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Clear the current
//interrupt-on-change
//configuration.
if (pdio->ClearIntOnChangeConfig (
   ) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

EnableAsyncWrite

Syntax

```
int EnableAsyncWrite (
   bool bEnable);
```

Include File C_digitalio.h

Description Enables/disables asynchronous-write mode.

Parameters

Name: bEnable

Description: A Boolean variable that specifies whether asynchronous-write mode is enabled or disabled. If TRUE, asynchronous-write mode is enabled. If FALSE, asynchronous-write mode is disabled.

Notes When asynchronous-write mode is enabled, all calls to **WriteOutputLine**, described on page 498, or **ExecuteCachedWrite** (if cached-write mode is enabled)**,** described on page 473, return immediately allowing other processing to continue while the write operation is in progress. Asynchronous-write mode is especially useful when generating output pulses that are somewhat lengthy in duration (greater than 500 ms), since it allows application-specific processing to be performed in parallel with output pulse generation.

This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

< 0	Method was unsuccessful.
0	Method was successful.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Error text buffer.
TCHAR szText[500];
// Enable asynchronous-write mode.
if (pdio->EnableAsyncWrite (
   TRUE) < 0)
{
  // Get error text.
  pdio->GetErrorText (szText,500);
}
```

EnableCachedWrite

Syntax

```
int EnableCachedWrite (
   bool bEnable);
```

Include File C_digitalio.h

Description Enables or disables cached-write mode.

Parameters

Name: bEnable

Description: A Boolean variable that specifies whether cached-write mode is enabled or disabled. If TRUE, cached-write mode is enabled. If FALSE, cached-write mode is disabled.

Notes When cached-write mode is enabled, all write operations generated by calls to **WriteOutputLine**, described on page 498, are stored in internal memory and are not written to the digital I/O device until the **ExecuteCachedWrite** method, described on page 473, is called. Cached-write mode is useful if you wish to write to multiple output lines at the same time.

Notes (cont.) This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

< 0	Method was unsuccessful.
0	Method was successful.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Error text buffer.
TCHAR szText[500];
// Enable cached-write mode.
if (pdio->EnableCachedWrite (
   TRUE) < 0)
{
  // Get error text.
  pdio->GetErrorText (szText,500);
}
```

EnableIntOnChange

Syntax

```
int EnableIntOnChange (
   int nLine,
   bool bEnable);
```

Include File C_digitalio.h

Description Enables interrupt-on-change for a specified digital input line.

Parameters

Name: nLine

Description: The input line to read. Values range from 0 to *n* – 1, where *n* is the number of input lines supported by the digital I/O device.

Name: bEnable

Description: A Boolean variable that specifies whether interrupt-on-change is enabled or disabled for the specified digital input line. If TRUE, interrupt-on-change is enabled. If FALSE, interrupt-on-change is disabled.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Enable interrupt-on-change for
//line 0.
if (pdio->EnableIntOnChange (0,
   TRUE) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

EnableLatchedRead

Syntax

```
int EnableLatchedRead (
   bool bEnable);
```

Include File C_digitalio.h

Description Enables latched-read operations.

Parameters

Name: bEnable

Description: A Boolean variable that specifies whether to enable or disable latched-read operations. If TRUE, latched-read operations are enabled. If FALSE, latched-read operations are disabled.

Notes

When latched-read operations are enabled, an application must call **ExecuteLatchRead**, described on page 474, to store the current state of all digital input lines in memory. While latched reads are enabled, all subsequent calls to **ReadInputLine,** described on page 492, return values from memory instead of reading the digital I/O device directly.

Returned Value

< 0	The method was unsuccessful.
0	The method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Enable latched reads.
if (pdio->EnableLatchedRead (TRUE)
   < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

EnableWaitOnRead

Syntax

```
int EnableWaitOnRead (
   bool bEnable);
```

Include File C_digitalio.h

Description Enables wait-on-read for all digital input lines for which interrupt-on-change was enabled.

Parameters

Name: bEnable

Description: A Boolean variable that specifies whether wait-on-read is enabled or disabled. If TRUE, wait-on-read is enabled. If FALSE, wait-on-read is disabled.

Notes When wait-on-read is enabled, all calls to **ReadInputLine**, described on page 492, and **ExecuteLatchedRead**, described on page 474, do not return until one or more digital input lines for which interrupt-on-change was enabled changes state or until the wait-on-read timeout period expires.

This method is available only if the device supports the DIO_CAP_INTONCHANGE capability.

Returned Value

< 0 The method was unsuccessful.

0 The method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Enable wait-on-read.
if (pdio->EnableWaitOnRead (TRUE)
   < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

ExecuteCachedWrite

Syntax

```
int ExecuteCachedWrite (
   );
```

Include File C_digitalio.h

Description Executes a cached-write operation.

Parameters None

Notes When cached-write mode is enabled, this method performs a single write to the digital I/O device. This write operation outputs all values to the digital I/O device that were written to memory (with calls to **WriteOutputLine**) since the last time **ExecuteCachedWrite** was called or since cached-write mode was enabled. Refer to page 498 for more information of **WriteOutputLine** and to page 473 for more information on **ExecuteCachedWrite**.

Notes (cont.) This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

< 0	The method was unsuccessful.
0	The method was successful.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Error text buffer.
TCHAR szText[500];
// Execute a cached-write.
if (pdio->ExecuteCachedWrite() <
   0)
{
  // Get error text.
  pdio->GetErrorText (szText,500);
}
```

ExecuteLatchedRead

Syntax

```
int ExecuteLatchedRead (
   );
```

Include File C_digitalio.h

Description Executes a latched-read operation.

Parameters None

Notes When latched-read mode is enabled, this method reads and stores the state of all input lines into internal memory. Subsequent calls to **ReadInputLine,** described on page 492, return values from memory instead of reading the input lines directly.

Notes (cont.) This method is available only if the device supports the DIO_CAP_INPUTLINES capability.

Returned Value

< 0 The method was unsuccessful.

0 The method was successful.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Error text buffer.
TCHAR szText[500];
// Execute a latched-read.
if (pdio->ExecuteLatchedRead() <
   0)
{
  // Get error text.
  pdio->GetErrorText(szText,500);
}
```

GetDeviceCaps

Syntax

```
int GetDeviceCaps (
   int* pnDeviceCaps);
```

Include File C_digitalio.h

Description Returns the digital I/O capabilities of the digital I/O device.

Parameters

Name: pnDeviceCaps

Description: A pointer to a variable in which the digital I/O capabilities of the digital I/O device are returned. Possible values are as follows:

- DIO_CAP_INPUTLINES – Device provides input lines.
- DIO_CAP_INTONLINES – Device supports interrupt on change.
- DIO_CAP_OUTPUTLINES – Device provides output lines.
- DIO_CAP_DEVICEPROPS - Device supports programmable properties
- DIO_CAP_DEVICECONFIG – Device supports config persistence
- DIO_CAP_CONFIGDIALOG – Device provides a configuration dialog box.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Variable to receive capabilities
int nDeviceCaps;
//Error text buffer.
TCHAR szText[500];
```

Example (cont.)

```
//Get the device capabilities
if (pdio->GetDeviceCaps
   (&nDeviceCaps) < 0)
{
  //Get error text.
  Pdio->GetErrorText (szText,
   500);
}
//Does the device provide
//input lines?
  if (nDeviceCaps &
   DIO_CAP_INPUTLINES)
{
  //Yes, it does!
}
```

GetDeviceConfig

Syntax

```
int GetDeviceConfig (
   LPSTREAM pStream);
```

Include File C_digitalio.h

Description Returns the configuration to the digital I/O device.

Parameters

Name: pStream

Description: A pointer to a windows stream object that contains the configuration information for the digital I/O device.

Notes None

Returned Value

< 0	Method was unsuccessful.
0	Method was successful.

Example

```
 // Digital I/O API object.
CcDigIODevice *pdio;
// Config stream.
LPSTREAM pStream;
// Error text buffer.
TCHAR szText[500];
// Create or get a reference to a
// stream object, etc.
pStream = pSomeStream;
// Get the current device
//configuration.
if ( pdio->GetDeviceConfig(
   pStream ) < 0 )
{
  // Get error text.
  pdio->GetErrorText(szText,500);
}
```

GetDeviceConfigFileDesc

Syntax

```
int GetDeviceConfigFileDesc (
   LPSTR szFileDesc,
   int nBufSize);
```

Include File C_digitalio.h

Description Returns the file description to use when displaying information about configuration files that are generated by a digital I/O device. This text appears in the “Save as type” box of the “Save config” dialog box that is used to generate device-specific configuration files in the device manager.

Parameters

Name: szFileDesc

Description: A pointer to the character buffer that receives the file description.

Name: nBufSize

Description: The size of the text buffer (in characters) pointed to by the *szFileDesc* parameter. This value must be greater than 0.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Buffer to receive text.
TCHAR szFileDesc[100]
//Error text buffer.
TCHAR szText[500];
//Get the configuration file
   description.
if
   (pdio->GetDeviceConfigFileDesc(
   szFileExt, 100) < 0)
```

Example (cont.)

```
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

GetDeviceConfigFileExt

Syntax

```
int GetDeviceConfigFileExt (
   LPSTR szFileExt,
   int nBufSize);
```

Include File C_digitalio.h

Description Returns the file extension to use when generating configuration files for a digital I/O device. This text appears as the file extension in the “Save as type” box of the “Save config” dialog box that is used to generate device-specific configuration files in the device manager.

Parameters

Name: szFileExt

Description: A pointer to the character buffer that receives the file extension.

Name: nBufSize

Description: The size of the text buffer (in characters) pointed to by the *szFileExt* parameter. This value must be greater than 0.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Buffer to receive text.
TCHAR szFileExt[100];
//Error text buffer.
TCHAR szText[500];
//Get the configuration file
   extension.
if (pdio->GetDeviceConfigFileExt(
   szFileExt, 100) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

GetDeviceProperty

Syntax

```
int GetDeviceProperty (
   int nPropId,
   int* pnValue);
```

Include File C_digitalio.h

Description Returns the value of a vendor-specific property on a digital I/O device.

Parameters

Name: nPropId

Description: A vendor-specific value that identifies the property to set.

Name: pnValue

Description: A pointer to a variable in which the value for the property is returned.

Notes Supported properties vary from vendor to vendor. Note that currently this method is not supported by Data Translation devices. Refer to your vendor-specific documentation for more information.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Variable that receives the
//property value
int nValue;
//Error text buffer.
TCHAR szText[500];
//Get the vendor-specific
//property.
if (pdio->GetDeviceProperty
   (nPropId, &nValue) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

GetErrorText

Syntax

```
int GetErrorText (
   LPSTR szErrorText,
   int nBufSize );
```

Include File C_digitalio.h

Description Returns the description of the last error that occurred.

Parameters

Name: szErrorText

Description: A pointer to a buffer in which the error text is returned.

Name: nBufSize

Description: The size of the buffer (in characters).

Notes None

Returned Value

< 0 The method was unsuccessful.

0 The method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Get error text.
pdio->GetErrorText (szText, 500);
```

GetInputLineCount

Syntax

```
int GetInputLineCount (
   int* pnCount);
```

Include File C_digitalio.h

Description Returns the number of input lines that are supported by the digital I/O device.

Parameters

Name: pnCount

Description: A pointer to an integer variable in which the number of input lines that are supported by the digital I/O device is returned. This value must not be NULL.

Notes This method is available only if the device supports the DIO_CAP_INPUTLINES capability.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Variable to receive the count.
int nCount;
//Error text buffer.
TCHAR szText[500];
//Get the number of input lines
//supported by the device.
if (pdio->GetInputLineCount
   (&nCount) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

GetOutputLineCount

Syntax

```
int GetOutputLineCount (
   int* pnCount);
```

Include File C_digitalio.h

Description Returns the number of output lines that are supported by the digital I/O device.

Parameters

Name: pnCount

Description: A pointer to an integer variable in which the number of output lines that are supported by the digital I/O device is returned. This value must not be NULL.

Notes This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
 // Digital I/O API object.
CcDigIODevice *pdio;
// Variable to receive the count.
int nCount;
// Error text buffer.
TCHAR szText[500];
// Get the number of output lines
// supported by the device.
if ( pdio->GetOutputLineCount (
   &nCount ) < 0 )
{
  // Get error text.
  pdio->GetErrorText (szText,500);
}
```

GetReadTimeout

Syntax

```
int GetReadTimeout (
   int* pnTimeout);
```

Include File C_digitalio.h

Description Returns the timeout value for wait-on-read operations.

Parameters

Name: pnTimeout

Description: A pointer to an integer variable in which the current wait-on-read timeout value, in milliseconds, is returned. This value must not be NULL.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Variable to receive the timeout
   value
int nTimeout;
//Error text buffer.
TCHAR szText[500];
//Get the current read timeout
   value.
if (pdio->GetReadTimeout(
   &nTimeout) < 0)
```

Example (cont.)

```
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

IsAsyncWriteDone

Syntax

```
bool IsAsyncWriteDone (
   );
```

Include File C_digitalio.h

Description Returns whether asynchronous-write mode is finished or not.

Parameters None

Notes This method is available only if a device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

TRUE The asynchronous-write operation has finished.

FALSE The asynchronous-write operation has not finished.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Is the pending
//asynchronous-write operation
//done?
if (pdio->IsAsyncWriteDone() )
{
  // Yes, it has.
}
```

Example (cont.)

```
  else
{
  // No, it hasn’t.
}
```

IsAsyncWriteEnabled

Syntax

```
bool IsAsyncWriteEnabled (
   int nLine);
```

Include File C_digitalio.h

Description Returns whether asynchronous-write mode is enabled or disabled.

Parameters None

Notes This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

TRUE	Asynchronous-write mode is enabled.
FALSE	Asynchronous-write mode is disabled.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Is asynchronous-write mode
// enabled?
if ( pdio->IsAsyncWriteEnabled() )
{
  // Yes, it’s enabled.
}
  else
{
  // No, it isn’t
}
```

IsCachedWriteEnabled

Syntax
```
bool IsCachedWriteEnabled (
   );
```

Include File C_digitalio.h

Description Returns whether cached-write mode is enabled or disabled.

Parameters None

Notes This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

TRUE Cached-write mode is enabled.

FALSE Cached-write mode is disabled.

Example
```
// Digital I/O API object.
CcDigIODevice *pdio;
// Is cached-write mode enabled?
if (pdio->IsCachedWriteEnabled() )
{
  // Yes, it's enabled.
}
  else
{
  // No, it isn't
}
```

IsIntOnChangeEnabled

Syntax
```
bool IsIntOnChangeEnabled (
   int nLine);
```

Include File C_digitalio.h

Description Returns whether interrupt-on-change is enabled or disabled for the specified digital input line.

Parameters

Name: nLine

Description: The input line to read. Values range from 0 to $n - 1$, where n is the number of input lines supported by the digital I/O device.

Notes None

Returned Value

TRUE Interrupt-on-change is enabled for the specified digital input line.

FALSE Interrupt-on-change is disabled for the specified digital input line.

Example

```
 //Digital I/O API Object.
CcDigIODevice *pdio;
//Is interrupt-on-change enabled
//for line 0?
if (pdio->IsIntOnChangeEnabled (
   0) )
{
  //Yes, it’s enabled.
}
else
{
  //No, it isn’t.
}
```

IsLatchedReadEnabled

Syntax

```
bool IsLatchedReadEnabled (
   );
```

Include File C_digitalio.h

Description Returns whether latched-read operations are enabled or disabled.

Parameters None

Notes None

Returned Value

TRUE Latched-read operations are enabled.

FALSE Latched-read operations are disabled.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Are latched reads enabled?
if (pdio->IsLatchedReadEnabled ( )
   )
{
  //Yes, it's enabled.
}
else
{
  //No, it isn't.
}
```

IsWaitOnReadEnabled

Syntax

```
bool IsWaitOnReadEnabled (
   );
```

Include File C_digitalio.h

Description Returns whether wait-on-read is enabled or disabled.

Parameters None

Notes This method is available only if the device supports the DIO_CAP_INTONCHANGE capability.

Returned Value

TRUE Wait-on-read is enabled.

FALSE Wait-on-read is disabled.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Is wait-on-read enabled?
if (pdio->IsWaitOnRead ( ) )
{
  //Yes, it’s enabled.
}
else
{
  //No, it isn’t.
}
```

ReadInputLine

Syntax

```
int ReadInputLine (
   int nLine,
   bool* pbLineState);
```

Include File C_digitalio.h

Description Returns the current state of the specified digital input line.

Parameters

Name: nLine

Description: Values range from 0 to *n* - 1 where *n* is the number of input lines supported by the digital I/O device.

Name: pbLineState

Description: A pointer to a Boolean variable in which the state of the specified digital input line is returned. If TRUE, the input line is high. If FALSE, the input line is low. This value must not be NULL.

Notes None

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Variable to receive the line
   state.
BOOL bLineState;
//Error text buffer.
TCHAR szText[500];

//Get the state of line 0.
if (pdio->ReadInputLine (0,
   &bLineState) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

SetDeviceConfig

Syntax

```
int SetDeviceConfig (
   LPSTREAM pStream);
```

Include File C_digitalio.h

Description Applies the configuration to the digital I/O device.

Parameters

Name: pStream

Description: A pointer to a windows stream object that contains the configuration information to apply to the digital I/O device.

Notes None.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
// Digital I/O API object.
CcDigIODevice *pdio;
// Config stream.
LPSTREAM pStream;
// Error text buffer.
TCHAR szText[500];
// Create or get a reference to
// a stream object, that
// contains a valid device
// configuration.
pStream = pSomeStream;
// Restore the current device
   configuration.
if (pdio->SetDeviceConfig (
   pStream) < 0)
```

Example (cont.)

```
{
  // Get error text.
  pdio->GetErrorText(szText,500);
}
```

SetDeviceProperty

Syntax

```
int SetDeviceProperty (
   int nPropId,
   int nValue);
```

Include File C_digitalio.h

Description Sets a vendor-specific property on a digital I/O device.

Parameters

Name: nPropId

Description: A vendor-specific value that specifies the property to set.

Name: nValue

Description: The desired value for the property.

Notes Supported properties vary from vendor to vendor. Note that currently this method is not supported by Data Translation devices. Refer to your vendor-specific documentation for more information.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];

//Set the vendor-specific
//property.
if (pdio->SetDeviceProperty
   (nPropId, 255) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

SetReadTimeout

Syntax

```
int SetReadTimeout (
   int nTimeout);
```

Include File C_digitalio.h

Description Sets the timeout value for wait-on-read operations.

Parameters

Name: nTimeout

Description: The timeout value, in milliseconds. This value must be greater than or equal to 0.

Notes A wait-on-read operation is a read call that does not return until one or more input lines, in a preconfigured set of lines, changes state or until the timeout period expires. If the time period set using this method expires before a line changes state, the call returns and an error is generated.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Error text buffer.
TCHAR szText[500];
//Set the read timeout to 1
   second.
if (pdio->SetReadTimeout(1000) <
   0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

ShowDeviceConfigDialog

Syntax

```
int ShowDeviceConfigDialog(
   HWND hParent);
```

Include File C_digitalio.h

Description Displays the device configuration dialog box for a digital I/O device.

Parameters

Name: hParent

Description: The handle of the window that acts as the parent window for the device configuration dialog box. This value must be a value window handle (it cannot be NULL).

Notes This method is available only if the device supports the DIO_CAP_CONFIGDIALOG capability.

Returned Value

< 0 Method was unsuccessful.

0 Method was successful.

Example

```
//Digital I/O API Object.
CcDigIODevice *pdio;
//Parent window.
HWND hParent;
//Error text buffer.
TCHAR szText[500];

//Get a window handle.
hParent = n_hWnd;

//Display the device configuration
//dialog box.
if (pdio->ShowDeviceConfigDialog(
   hParent) < 0)
{
  //Get error text.
  pdio->GetErrorText (szText,
   500);
}
```

WriteOutputLine

Syntax

```
int WriteOutputLine (
   int nLine,
   bool bLineState,
   int nPulseWidth);
```

Include File C_digitalio.h

Description Sets the line state and pulse width for the specified output line.

Parameters

Name: nLine

Description: Specifies the output line to write to. Values range from 0 to *n* –1, where *n* is the number of output lines that are supported by the digital I/O device.

Name: bLineState

Description: The desired state of the output line. If TRUE, the specified output line is set to the high state. If FALSE, the specified output line is set to the low state.

Name: nPulseWidth

Description: The width of the pulse to generate on the specified output line, in milliseconds. This value must be greater than or equal to 0. If 0, no pulse is generated.

Notes If cached-write mode is enabled, the line state and the pulse width are stored in memory and are not written to the specified output line until the **ExecuteCachedWrite** method, described on page 473, is invoked.

This method is available only if the device supports the DIO_CAP_OUTPUTLINES capability.

Returned Value

< 0	Method was unsuccessful.
0	Method was successful.

Example

```
 // Digital I/O API object.
CcDigIODevice *pdio;
// Error text buffer.
TCHAR szText[500];
// Generate a 100 ms high-going
// pulse on output line 0.
if (pdio->WriteOutputLine (0,
   TRUE, 100) < 0 )

{
  // Get error text.
  pdio->GetErrorText (szText,500);
}
```

11

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 DT Vision Foundry 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 28.

Table 28: 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** FindEdges (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 DT Vision Foundry 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 DT Vision Foundry, the origin of the image is the lower, left corner of the image, by default. Therefore, a rectangle in DT Vision Foundry 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 (white to black transitions) are found.
- 3 - RISING_EDGE –All rising edges (black to white 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);
```

FindEdges

Syntax

```
CcRoiBase** FindEdges(
   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
CNewRois = CEdgeFinder.FindEdges(
   &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)
   {
   . . . .
   }
}
```

12

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 DT Vision Foundry, and saves an DT Vision Foundry 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 29.

Table 29: 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 DT Vision Foundry 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 DT Vision Foundry 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 29 starting on page 937. 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 DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry images must be divisible by four. This is due to the way DT Vision Foundry 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 DT Vision Foundry 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;
}
```

13

Using the Filter Tool API

Overview of the Filter Tool 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 30.

Table 30: 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;
```

Notes (cont.)

```
int iKernel[7][7];
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;
```

Notes (cont.)

```
int Kernel[7][7];
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.

13

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.

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);

}
```

14

Using the Gauge Tool API

Overview of the Gauge Tool API

The API for the Gauge tool has one object only: the CcRoiGauge class. The CcRoiGauge class is designed to work within the DT Vision Foundry 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 gauging 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 31.

Table 31: CcRoiGauge Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcRoiGauge(void);
	~CcRoiGauge(void);

Table 31: 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 31: CcRoiGauge Object Methods (cont.)

Method Type	Method Name
CcRoiGauge Class Methods (cont.)	MaxDirectedDistance();
	MinOppositeDistance();
	MaxOppositeDistance();
	MinPerpendicularDistance();
	MaxPerpendicularDistance();
	Roundness();
	GrayAverage();
	RedAverage();
	GreenAverage();
	BlueAverage();
	HueAverage();
	SatAverage();
	LumAverage();
	GrayValue();
	RedValue();
	GreenValue();
	BlueValue();
	HueValue();
	SatValue();
	LumValue();
	XIntersection();

Table 31: CcRoiGauge Object Methods (cont.)

Method Type	Method Name
CcRoiGauge Class Methods (cont.)	YIntersection();
	StMResult * GetResults();
	CcList * GetMethodList();
	HeightBoundingRect();
	WidthBoundingRect();
	AngleBoundingRect();

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 DT Vision Foundry 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 DT Vision Foundry 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;
```

14

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 DT Vision Foundry 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 Gauge tool.

Parameters

Name: InputImage

Description: Pointer to a DT Vision Foundry 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 Gauge tool.

Parameters

Name: InputImage

Description: Pointer to a DT Vision Foundry 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 Gauge tool.

Parameters

Name: InputImage

Description: Pointer to a DT Vision Foundry 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 600.

Example

The following is a sample code fragment:

```
float     Angle;
CcImage   *CImageIn1,*CImageIn2;
CcImage   *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOO        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.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.

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 600.

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);
```

Example (cont.)

```
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 boundry box that encompasses the ROI.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 600.

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);
```

Example (cont.)

```
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetRoi2(CRoi2);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);

// Invoke the gauging 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 600.

Example The following is a sample code fragment:

```
float        Angle;
CcImage     *CImageIn1, *CImageIn2;
CcImage     *CImageIn3;
CcRoiBase   *CRoi1, *CRoi2, *CRoi3;
BOOL         bStatus;
CcRoiGauge   CRoiGauge;
```

Example (cont.)

```
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.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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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);
```

Example (cont.)

```
CRoiGauge.SetAngle(Angle);
// Invoke the gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging 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 600.

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 gauging method
CRoiGauge.Roundness();

// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

GrayAverage

Syntax

```
GrayAverage(
   );
```

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 600.

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 gauging method
CRoiGauge.GrayAverage();

// 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 600.

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 gauging 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 600.

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
```

14

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 gauging 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 600.

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 gauging method
CRoiGauge.BlueAverage();

// 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 600.

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.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 600.

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.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 600.

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();

// 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.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 600.

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.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 600.

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.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 600.

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.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 600.

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.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 600.

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 600.

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.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 600.

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.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 600.

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);
//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 600.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1, *CImageIn2;
CcImage *CImageIn3;
CcRoiBase *CRoi1, *CRoi2, *CRoi3;
BOOL bStatus;
CcRoiGauge CRoiGauge;
```

Example (cont.)

```
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 gauging 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);
CRoiGauge.SetRoi3(CRoi3);
CRoiGauge.SetAngle(Angle);
// Get the list of measurement
//methods
TheList = CRoiGauge.GetMethodList
   ();
```

Example (cont.)

```
// 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 gauging 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 gauging method pointers. This list provides a way to associate text names of the gauging methods with pointers to these methods so that you can invoke the gauging 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 600.

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);
// 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”);
```

Example (cont.)

```
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();
```

HeightBoundingRect

Syntax

```
HeightBoundingRect(
   );
```

Include File C_RoiGauge.h

Description For a freehand ROI or poly freehand ROI, returns the height of the minimum bounding box by area.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 600.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1;
CcRoiBase *CRoi1;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//will hold 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.SetRoi1(CRoi1);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.HeightBoundingRect();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

WidthBoundingRect

Syntax

```
WidthBoundingRect(
   );
```

Include File C_RoiGauge.h

Description For a freehand ROI or poly freehand ROI, returns the width of the minimum bounding box by area.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 600.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1;
CcRoiBase *CRoi1;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
//will hold result of a
//measurement
```

14

Example (cont.)

```
//Fill the images and ROIs with
//data
. . .
// Set all the necessary inputs to
// the CRoiGauge class
CRoiGauge.SetImage1(CImageIn1);
CRoiGauge.SetRoi1(CRoi1);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.WidthBoundingRect();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

AngleBoundingRect

Syntax

```
AngleBoundingRect(
   );
```

Include File C_RoiGauge.h

Description For a freehand ROI or poly freehand ROI, returns the angle of the minimum bounding box by area.

Parameters None

Return Values Values are returned by the **GetResults** method, described on page 600.

Example The following is a sample code fragment:

```
float Angle;
CcImage *CImageIn1;
CcRoiBase *CRoi1;
BOOL bStatus;
CcRoiGauge CRoiGauge;
stMResult *pResult;
```

Example (cont.)

```
//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.SetRoi1(CRoi1);
CRoiGauge.SetAngle(Angle);
// Invoke the gauging method
CRoiGauge.AngleBoundingRect();
// Retrieve pointer to the result
pResult = CRoiGauge.GetResults();
```

15

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 DT Vision Foundry 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 32.

Table 32: 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 DT Vision Foundry-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 DT Vision Foundry-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;
```

Notes (cont.)

```
//Lowest value in histogram with
//nonzero value
float fMax;
//Highest value in histogram with
//nonzero value
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);
}
```

16

Using the Image Classifier Tool API

Overview of the Image Classifier Tool API

The Image Classifier tool is a C++ class that is designed to work within the DT Vision Foundry environment. It is a general-purpose classifier for grayscale images.

The Image Classifier tool compares images against a catalog of images that you built previously.

Images must meet the following requirements for use with the Image Classifier tool:

- The size of the input images must be exactly the same size in the classification session, and
- The catalog images, mask images, and the images under test must contain exactly the same dimensions.

Note: This tool is light sensitive; therefore, it is recommended that you use appropriate lighting while using this tool.

The API for the Contour Classifier tool has one object only: the CcImgCL class. The CcImgCL class uses a standard constructor and destructor and the class methods listed in Table 33.

Table 33: CcImgCL Class Methods

Method Type	Method Name
Constructor & Destructor Methods	CcImgCL();
	~CcImgCL();

Table 33: CcImgCL Class Methods (cont.)

Method Type	Method Name
CcImgCL Class Methods	int Classify(CcImage *pImageIn, CcRoiBase *pcRoi);
	STIMGCLRESULT *GetResult(void);
	RECT *GetRoiIn(void);
	int LoadCatalog(char *cFname);
	int SaveCatalog (char *cFname);
	bool *SetRoiIn(RECT *stInRoiRect);
	void SetLightDesens(bool bLight);
	bool SetExtendedClassificationDepth(float fDepth);
	void SetScoreCalculation(bool bScore);
	int CountNumHypos(int iHypoType, double dAngleStart, double dAngleStep, double dAngleEnd, int iShiftInX, int iShiftInY);
	bool InitializeTrainingProcedure(int iNumHypos);
	void SetInputImageWidth(int iWidth);
	void SetInputImageHeight(int iHeight);
	bool SetBackgroundImage(CcImage *pImage);
	bool SetInputImage(CcImage *pImage);
	bool SetInputMask(CcImage *pImage);
	bool SetImageName(char *cImageName);
	bool SetHypothesisType(int iHypoType);
	bool SetShiftInX(int iShiftInx);
	bool SetShiftInY(int iShiftInY);
	bool SetAngleStart(double dAngleStart);
	bool SetAngleStep(double dAngleStep);

Table 33: CcImgCL Class Methods (cont.)

Method Type	Method Name
CcImgCL Class Methods (cont.)	bool SetAngleEnd(double dAngleEnd);
	void AddImage(void);
	bool BuildCatalog(void);
	void UseNormalizedMetric(bool);

CcImgCL Methods

This section describes each method of the CcImgCL class in detail.

Classify

Syntax

```
int Classify(CcImage *pImageIn,
   CcRoiBase *pCRoi);
```

Include File C_ImgCL.h

Description Classifies an image under test by comparing it to the images in the catalog.

Parameters

Name: pImageIn

Description: A pointer to an input image.

Name: pCRoi

Description: A pointer to a region of interest inside the input image.

Notes The Image Classifier tool works on the pixels contained within the ROI specified by *pCRoi*.

Return Values

–1 Classification failed.

0 Classification was successful.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcImage  *InImage;
CcRoiBase *InRoi;
//Classify the target
m_CImgCL.Classify(InImage, InRoi);
```

GetResult

Syntax

```
STIMGCLRESULT *GetResult(void);
```

Include File C_ImgCL.h

Description Returns the results of the classification.

Parameters None

Notes Call this method after calling **Classify**.

Return Values The pointer to the `STIMGCLRESULT` structure, which is defined as follows:

```
struct stResultTag
{
 //Name of the matching element
 CString CMatch;
 //Match confidence measure
 double dScore;
 //Angle of rotation
 float fAngle;
 //Shift in X (in pixels)
 int iShiftInX;
 //Shift in Y (in pixels)
 int iShiftInY;
};
typedef stResultTag STIMGCLRESULT;
```

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcImage  *InImage;
CcRoiBase *InRoi;
STIMGCLRESULT *pstResult;
//Classify the target
m_CImgCL.Classify(InImage, InRoi);
pstResult = m_CImgCL.GetResult();
```

GetRoiIn

Syntax `RECT *GetRoiIn(void);`

Include File C_ImgCL.h

Description Returns a pointer to a RECT structure that describes an ROI, which is used to build the catalog.

Parameters None

Notes Use this method after calling **SetRoiIn** or **LoadCatalog**.

Return Values A pointer to the `RECT` structure.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcRoiBase *InRoi;
//Get the ROI
InRoi = m_CImgCL.GetRoiIn();
```

LoadCatalog

Syntax `int LoadCatalog(char *cFname);`

Include File C_ImgCL.h

Description Restores the image catalog.

Parameters

Name: cFname

Description: The name of the file from which to load the image catalog.

Notes None

Return Values

–1 Operation failed.

0 Catalog was successfully loaded.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

pstResult = m_CImgCL.LoadCatalog
   ("catalog.cat")
```

SaveCatalog

Syntax `int SaveCatalog(char *cFname);`

Include File C_ImgCL.h

Description Saves the image catalog.

Parameters

Name: cFname

Description: The name of the file into which to save the image catalog.

Notes None

Return Values

–1 Operation failed.

0 Catalog was successfully saved.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
```

Example (cont.)

```
pstResult = m_CImgCL.SaveCatalog
   ("catalog.cat")
```

SetRoiIn

Syntax `RECT *SetRoiIn(RECT *stInRoiRect);`

Include File C_ImgCL.h

Description Saves a RECT structure that describes an ROI, which is used to build the catalog.

Parameters

Name: stInRoiRect

Description: A pointer to a RECT structure that describes the input ROI.

Notes None

Return Values

True Operation succeeded.

False Operation failed. An invalid RECT structure was supplied.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcRoiBase *InRoi;

//Set the ROI
InRoi = m_CImgCL.SetRoiIn();
```

SetLightDesens

Syntax

```
void SetLightDesens(bool bLight);
```

Include File

C_ImgCL.h

Description

Specifies that the tool will be less sensitive to changes in lighting conditions.

Parameters

Name: bLight

Description: If TRUE, the tool will be less sensitive to changes in lighting conditions. If FALSE, the tool will not be less sensitive to changes in lighting conditions.

Notes

It is recommended that you specify *bLight* as TRUE in an environment where you can accurately control the light intensity. Otherwise, you may get inaccurate results.

You may need to experiment with this option.

Return Values

None

Example

The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Turn the option on
InRoi = m_CImgCL.SetLightDesens(
   TRUE);
```

SetExtendedClassificationDepth

Syntax

```
bool SetExtendedClassification
   Depth(float fDepth);
```

Include File C_ImgCL.h

Description Specifies the classification depth.

Parameters

Name: fDepth

Description: A value from 0 to 1. A value of 0 means that no additional processing will be performed.

Notes A higher value for *fDepth* increases the accuracy of the tool since more processing is done. This is most suitable when you need to distinguish between very similar images.

Return Values

TRUE Operation was successful.

FALSE Invalid input value.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Specify a classification depth
InRoi=m_CImgCL.SetExtended
   ClassificationDepth(0.10);
```

SetScoreCalculation

Syntax

```
void SetScoreCalculation(bool
   bScore);
```

Include File C_ImgCL.h

Description Enables/disables score calculations.

Parameters

Name: bScore

Description: A value of TRUE enables score calculations; a value of FALSE disables score calculations.

Notes Calculating the score extends the processing time.

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Turn the option on
InRoi=m_CImgCL.
   SetScoreCalculation(TRUE);
```

CountNumHypos

Syntax

```
int CountNumHypos(int iHypoType,
   double dAngleStart,
   double dAngleStep,
   double dAngleEnd,
   int iShiftInX, int iShiftInY);
```

Include File C_ImgCL.h

Description

Computes the hypotheses number that is used by the **InitializeTrainingProcedure** method.

Parameters

Name: iHypoType

Description: Hypothesis selection for this image. The hypothesis selections are defined in the *eHypoType* enumeration, which is defined as follows:

```
typedef enum eHypoType
{
  //Each image is classified; only
  //the best matching image name
  //is returned
  HYPOTHESIS_PER_IMAGE,

  //Classification result for
  //every rotation of the image is
  //returned with the image name
  HYPOTHESIS_PER_ROT,

  //Classification result for
  //every shift of the image is
  //returned with the image name
  HYPOTHESIS_PER_SHIFT,

  //Only the image name is
  //returned regardless of the
  //rotation or shift of the image
  HYPOTHESIS_FORALL_ROTS_AND_
   FORALL_SHIFTS,
```

Description (cont):

```
  //Only the image name is
  //returned regardless of the
  //image rotations
  HYPOTHESIS_FORALL_ROTS,

  //Only the image name is
  //returned regardless of any
  //image shifts
  HYPOTHESIS_FORALL_SHIFTS,

  //Classification result for
  //every image rotation,
  //regardless of any shifts,
  //is returned
  HYPOTHESIS_PER_ROT_IND_SHIFTS,

  //Classification result for
  //every combination of rotations
  //is returned with the image
  //name
  HYPOTHESIS_PER_ROT_AND_PER_
   SHIFTS,

  //Classification result for
  //every shift of the image,
  //regardless of any rotations,
  //is returned with the
  //image name
HYPOTHESIS_PER_SHIFT_IND_ROTS,

//Classification result for
//every combination of shifts is
//returned with the image name
BACKGROUND_IMAGE,
//Invalid entry
HYPOTHESIS_INVALID
};
```

Name: dAngleStart

Description: The starting angle for generating rotated images in the training process of the tool.

Name: dAngleStep

Description: The amount of rotation to use between each angle for generating rotated images in the training process of the tool.

Name: dAngleEnd

Description: The ending angle for generating rotated images in the training process of the tool.

Name: iShiftInX

Description: The shift in the X-direction for generating images in the training process of the tool.

Name: iShiftInY

Description: The shift in the Y-direction for generating images in the training process of the tool.

Notes: Invoke this method for each input image that you supply to the tool. Supply the summed results of this method to the **InitializeTrainingProcedure** method to ensure that the training catalog is complete.

Return Values The number of hypotheses is calculated based on the supplied inputs.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Turn the option on
```

Example (cont.)

```
int iHypos=m_CImgCL.CountNumHypos(
   HYPOTHESIS_PER_IMAGE), 0.0,
   1.0, 20.0, 4, 4);
```

InitializeTrainingProcedure

Syntax

```
bool InitializeTrainingProcedure(
   int iNumHypos);
```

Include File C_ImgCL.h

Description Initializes the training procedure prior to training the Image Classifier tool.

Parameters

Name: iNumHypos

Description: The number calculated using the **CountNumHypos** method.

Notes You must invoke the **CountNumHypos** method for each input image that is supplied to the tool and sum the results prior to calling this method.

You must invoke this method before training the Image Classifier tool. You must destroy then construct the Image Classifier object each time a new training session is started.

Return Values

TRUE Operation was successful.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
```

Example (cont.)

```
int iNumHypos;

//Initialize the class before
//you begin the training
InRoi=m_CImgCL.
   InitializeTrainingProcedure(
   iNumHypos);
```

SetInputImageWidth

Syntax

```
void SetInputImageWidth(
   int iWidth);
```

Include File C_ImgCL.h

Description Sets the image width that is used globally by the tool.

Parameters

Name: iWidth

Description: The width of the input image in pixels.

Notes You should invoke this method after calling the **InitializeTrainingProcedure** method.

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;


//Set the width of the input image
m_CImgCL.SetInputImageWidth(36);
```

SetInputImageHeight

Syntax

```
void SetInputImageHeight(
   int iHeight);
```

Include File C_ImgCL.h

Description Sets the image height that is used globally by the tool.

Parameters

Name: iHeight

Description: The height of the input image in pixels.

Notes You should invoke this method after calling the **InitializeTrainingProcedure** method.

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Set the height of the
//input image
m_CImgCL.SetInputImageHeight(36);
```

SetBackgroundImage

Syntax

```
bool SetBackgroundImage(
   CcImage *pImage);
```

Include File C_ImgCL.h

Description Sets the background image that is required by the tool to generate an intermediate image.

Parameters

Name: pImage

Description: A pointer to a valid background image.

Notes You should invoke this method after calling the **SetInputImageWidth** and **SetInputImageHeight** methods.

Return Values

TRUE Image was valid and was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcImage  *pImage;

//Set the background image
bool bRetVal= m_CImgCL.
   SetBackgroundImage(pImage);
```

SetInputImage

Syntax

```
bool SetInputImage(
   CcImage *pImage);
```


Include File C_ImgCL.h

Description Specifies the input image to add to the catalog.

Parameters

Name: pImage

Description: A pointer to a valid input image.

Notes You should invoke this method once for each supplied input image.

Return Values

TRUE Image was valid and was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcImage  *pImage;

//Set the input image
bool bRetVal= m_CImgCL.
   SetInputImage(pImage);
```

SetInputMask

Syntax

```
bool SetInputMask(
   CcImage *pImage);
```

Include File C_ImgCL.h

Description Specifies the input mask that the tool uses to generate additional input images.

Parameters

Name: pImage

Description: A pointer to a valid input image.

Notes In general, you should invoke this method once for each supplied input image, although some images may not require a mask.

Return Values

TRUE Image was valid and was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
CcImage  *pImage;

//Set the input image
bool bRetVal= m_CImgCL.
   SetInputMask(pImage);
```

SetImageName

Syntax

```
bool SetImageName(
   char *cImageName);
```

Include File C_ImgCL.h

Description Specifies the name of the image that is entered using the **SetInputImage** method.

Parameters

Name: pImageName

Description: A pointer to a valid input image name string.

Notes After the image is classified, this name is assigned to the “Match” portion of the results structure.

Return Values

TRUE String was assigned.

FALSE Operation failed.

16

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
char  cName[200];

strcpy(cName, “whatever”);

//Set the image name
bool bRetVal= m_CImgCL.
   SetInputMask(cName);
```

SetHypothesisType

Syntax

```
bool SetHypothesisType(
   int iHypoType);
```

Include File C_ImgCL.h

Description Specifies the hypothesis selection for a particular training image.

Parameters

Name: iHypoType

Description: Hypothesis selection for this image. The hypothesis selections are defined in the *eHypoType* enumeration, which is defined as follows:

```
typedef enum eHypoType
{
  //Each image is classified; only
  //the best matching image name
  //is returned
  HYPOTHESIS_PER_IMAGE,
```

Description (cont):

```
 //Classification result for
  //every rotation of the image is
  //returned with the image name
  HYPOTHESIS_PER_ROT,

//Classification result for
  //every shift of the image is
  //returned with the image name
  HYPOTHESIS_PER_SHIFT,

//Only the image name is
  //returned regardless of the
  //rotation or shift of the image
  HYPOTHESIS_FORALL_ROTS_AND_
   FORALL_SHIFTS,

//Only the image name is
  //returned regardless of the
  //image rotations
  HYPOTHESIS_FORALL_ROTS,

  //Only the image name is
  //returned regardless of any
  //image shifts
  HYPOTHESIS_FORALL_SHIFTS,

  //Classification result for
  //every image rotation,
  //regardless of any shifts,
  //is returned
  HYPOTHESIS_PER_ROT_IND_SHIFTS,
```

Description (cont):

```
 //Classification result for
  //every combination of rotations
  //is returned with the image
  //name
  HYPOTHESIS_PER_ROT_AND_PER_
   SHIFTS,

  //Classification result for
  //every shift of the image,
  //regardless of any rotations,
  //is returned with the
  //image name
  HYPOTHESIS_PER_SHIFT_IND_ROTS,

//Classification result for
  //every combination of shifts is
  //returned with the image name
  BACKGROUND_IMAGE,
  //Invalid entry
  HYPOTHESIS_INVALID
};
```

Notes None

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Assign a hypothesis for every
//image
```

Example (cont.)

```
bool bRetVal= m_CImgCL.
   SetHypothesisType(
   HYPOTHESIS_PER_IMAGE);
```

SetShiftInX

Syntax

```
bool SetShiftInX(
   int iShiftInX);
```

Include File C_ImgCL.h

Description Specifies the shift in the X-direction of the input image that is required to generate the intermediate training images.

Parameters

Name: iShiftInX

Description: The number of pixels by which the input image is shifted in the X-direction.

Notes The Image Classifier automatically generates images with the specified shift range for the purpose of training the input images.

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Allow up to 4 pixel shifts +/-
bool bRetVal=m_CImgCL.SetShiftInX(
   4);
```

16

SetShiftInY

Syntax

```
bool SetShiftInY(
   int iShiftInY);
```

Include File C_ImgCL.h

Description Specifies the shift in the Y-direction of the input image that is required to generate the intermediate training images.

Parameters

Name: iShiftInY

Description: The number of pixels by which the input image is shifted in the Y-direction.

Notes The Image Classifier automatically generates images with the specified shift range for the purpose of training the input images.

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Allow up to 4 pixel shifts +/-
bool bRetVal=m_CImgCL.SetShiftInY(
   4);
```

SetAngleStart

Syntax

```
bool SetAngleStart(
   double dAngleStart);
```

Include File C_ImgCL.h

Description Specifies the starting angle for the generated rotated images in the training process of the tool.

Parameters

Name: dAngleStart

Description: The starting angle for generating rotated images, in degrees.

Notes The Image Classifier automatically generates images with the specified rotation for the purpose of training the input images.

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//Start at 0 degrees
bool bRetVal=m_CImgCL.
   SetAngleStart(0.0);
```

SetAngleStep

Syntax

```
bool SetAngleStep(
   double dAngleStep);
```

Include File C_ImgCL.h

Description Specifies the amount of rotation to use between each angle for generating rotated images in the training process of the tool.

This value must be less than *dAngleStart* minus *dAngleEnd*.

Parameters

Name: dAngleStep

Description: The amount of rotation to use between each angle, in degrees. Specifies the amount of rotation to use between each angle for generating rotated images in the training process of the tool. This value must be less than *dAngleStart* minus *dAngleEnd.*

Notes The Image Classifier automatically generates images with the specified rotation for the purpose of training the input images.

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;
//One degree step
bool bRetVal=m_CImgCL.
   SetAngleStep(1.0);
```

SetAngleEnd

Syntax

```
bool SetAngleEnd(
   double dAngleEnd);
```

Include File C_ImgCL.h

Description Specifies the ending angle for the generated rotated images in the training process of the tool.

Parameters

Name: dAngleEnd

Description: The ending angle for generating rotated images, in degrees.

Notes The Image Classifier automatically generates images with the specified rotation for the purpose of training the input images.

Return Values

TRUE Valid integer was assigned.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL  m_CImgCL;

//End at 60 degrees
bool bRetVal=m_CImgCL.
   SetAngleEnd(60.0);
```

AddImage

Syntax `void AddImage(void);`

Include File C_ImgCL.h

Description Adds the image to the catalog.

Parameters None

Notes None

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL m_CImgCL;

//Assign the image
m_CImgCL.AddImage();
```

BuildCatalog

Syntax `bool BuildCatlog(void);`

Include File C_ImgCL.h

Description Creates the catalog of images with which to compare images under test.

Parameters None

Notes None

Return Values

TRUE Catalog was built successfully.

FALSE Operation failed.

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL m_CImgCL;

//Create the catalog
bool bRetVal=m_CImgCL.
   BuildCatalog();
```

UseNormalizedMetric

Syntax `void UseNormalizedMetric(bool);`

Include File C_ImgCL.h

Description Specifies how you want to represent the score that is assigned to a match.

Parameters

TRUE The score can range from 0.0 to 1.0, where 0.0 is the worst possible match and 1.0 is the best possible match.

FALSE The score can range from 0.0 to negative infinity, where 0.0 is the best possible match and the lower the negative value, the worse the match.

Notes None

Return Values None

Example The following is a sample code fragment:

```
//Instantiate the image classifier
//class
CcImgCL m_CImgCL;

//Specify a score of 0.0 to 1.0
m_CImgCL.UseNormalizedMetrics(
   TRUE);
```

17

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 34.

Table 34: 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

//Allocate memory for objects
```

Example (cont.)

```
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();
//Open image from disk (or get
//image data from frame grabber
```

Example (cont.)

```
C8BitImageIn->OpenBMPFIle(
   “InImage.bmp”);

//Scale the image by 50%
iScaleFactor = 50;
//Set the fill color to black
iFillValue = 0;

CImgMod.Scale(C8BitImageIn,
   C8BitImageOut,iScaleFactor,
   iFillValue);
//Save output to disk
C8BitImageOut->SaveBMPFile(
   “OutImage.bmp”);

//Free memory
delete C8BitImageIn;
delete C8BitImageOut;
}
```

18

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 DT Vision Foundry 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 35.

Table 35: CcLineProfile Object Methods

Method Type	Method Name
Constructor & Destructor Method	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);
	int GetExactPoint(float fPixelLocation,float* fExactX, float* fExactY,CcCalibration* CalibrationObject);

Table 35: CcLineProfile Object Methods (cont.)

Method Type	Method Name
CcLineProfile Class Methods (cont.)	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 DT Vision Foundry-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 starting on page 11.

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 starting 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

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.

Notes (cont.)

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( )**.

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

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

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.

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.

18

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*).

Notes (cont.)

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 *DT Vision Foundry 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 *DT Vision Foundry 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);
}
```

19

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 36.

Table 36: 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 36: 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:

Description: 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( )**.

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);
}
```

20

Using the Picture Tool API

Overview of the Picture Tool API

The Picture tool is derived from the base picture class CcPictureTool. CcPictureTool allows you to acquire frames from a frame grabber board. The Picture Tool API is intended to be used with the Device Manager API, described on page 206.

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 37.

Table 37: CcPictureTool Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcPictureTool(void);
	~CcPictureTool(void);
CcPictureTool Class Methods	int GetDeviceCaps(int* pnDevCaps);
	BOOL IsDeviceCapSupported(int nDeviceCap);
	int SetDeviceProperty (int nPropId, int nValue);
	int GetDeviceProperty (int nPropId, int* pnValue);
	int GetInputSourceCount(int *pnSource);
	int SetInputSource(int nSource);
	int GetInputSource(int *pnSource);
	int SetTimeout(int nTimeOut);
	int GetTimeout(int pnTimeOut);
	int SetImageScale(int nHorzScale, int nVertScale);

Table 37: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int GetImageScale(int *pnHorzScale, int *pnVertScale);
	int SetHorzImageScale(int nHorzScale);
	int GetHorzImageScale(int *pnHorzScale);
	int SetVertImageScale(int nVertScale);
	int GetVertImageScale(int *pnVertScale);
	int GetScaledImageDims(int *pnWidth, int *pnHeight);
	int GetScaledImageWidth(int *pnWidth);
	int GetScaledImageHeight(int *pnHeight);
	int GetImageScaleLimits(PDL_IMAGE_SCALE_LIMITS *pLimits);
	int SetImageDims(int nWidth, int nHeight);
	int GetImageDims(int *pnWidth, int *pnHeight);
	int SetImageWidth(int nWidth);
	int GetImageWidth(int *pnWidth);
	int SetImageHeight(int nHeight);
	int GetImageHeight(int *pnHeight);
	int GetImageDimsLimits(PDL_IMAGE_DIMS_LIMITS *pLimits);
	int SetImageType(int nImageType);
	int GetImageType(int *pnImageType);
	int SetImageTypeEx(int nImageTypeEx);
	int u(int *pnImageTypeEx);
	int GetImageTypeLimits(PDL_IMAGE_TYPE_LIMITS *pLimits);

Table 37: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int GetCompatibleImage(CcImage **ppImage);
	int SetImageAverage(int nImageCount);
	int EnableTimeStamping(BOOL bEnable);
	BOOL IsTimeStampingEnabled();
	int StartStreaming();
	int StopStreaming();
	int WaitForImage();
	BOOL IsStreamingInProgress();
	int AcquireImage(CcImage *pImage);
	int TimedAcquireToAVI(CcAVI pAVI, int nImageCount, int nTimeDelay);
	int TimedAcquireToDisc(LPCSTR szDir, LPCSTR szBaseFileName, int nImageCount,int nCountStart, int nTimeDelay);
	int TimedAcquireToMemory(LPCSTR szBaseImageName, int nImageCount,int nCountStart, int nTimeDelay, CcList *pImageList);
	int StartLiveVideo(HWND hWindow);
	int StopLiveVideo();
	BOOL IsLiveVideoRunning();
	int SetDeviceConfig(LPSTREAM pStream);
	int GetDeviceConfig(LPSTREAM pStream);
	int LoadDeviceConfig(LPSTR szFileName);
	int SaveDeviceConfig(LPCSTR szFileName);
	int GetDeviceConfigFileExt(LPSTR szFileExt, int nBufSize);

Table 37: CcPictureTool Object Methods (cont.)

Method Type	Method Name
CcPicture Class Methods (cont.)	int GetDeviceConfigFileDesc(LPSTR szFileDesc, int nBufSize);
	int ShowDeviceConfigDialog(HWND hParent);
	void GetErrorText(LPSTR szErrorText, int nBufSize);

CcPictureTool Methods

This section describes each method of the CcPictureTool class in detail.

GetDeviceCaps

Syntax

```
int GetDeviceCaps(
   int* pnDevCaps
   );
```

Include File C_PicTool.h

Description Returns the capabilities for the imaging device that is associated with the CcImageDevice object.

Parameters

Name: pnDevCaps

Description: A pointer to an integer that receives the device capabilities bit field. Possible values are as follows:

- **IMG_CAP_TIMEOUT –** Indicates whether the device supports timeouts. If this flag is enabled, the device supports the following methods: **SetTimeout**, described on page 718, and **GetTimeout**, described on page 720.

Description (cont):

- **IMG_CAP_IMAGESCALE –** Indicates whether the device supports scaling. If this flag is enabled, the device supports the following methods: **GetImageScaleLimits**, described on page 733, **GetImageScale**, described on page 722, **SetImageScale**, described on page 721, **GetHorzImageScale**, described on page 725, **SetHorzImageScale**, described on page 724, **GetVertImageScale**, described on page 727, **SetVertImageScale**, described on page 726, **GetScaledImageDims**, described on page 729, **GetScaledImageWidth**, described on page 730, and **GetScaledImageHeight**, described on page 731.
- **IMG_CAP_STREAMING –** Indicates whether the device supports streaming. If this flag is enabled, the device supports the following methods: **StartStreaming**, described on page 760, and **StopStreaming**, described on page 761.
- **IMG_CAP_LIVEVIDEO –** Indicates whether the device supports displaying live video in a window. If this flag is enabled, the device supports the following methods: **StartLiveVideo**, described on page 774, and **StopLiveVideo**, described on page 775.

Description (cont.):

- **IMG_CAP_DEVICEPROPS –** Indicates whether the device supports device-specific programmable properties. If this flag is enabled, the device supports the following methods: **SetDeviceProperty**, described on page 712, and **GetDeviceProperty**, described on page 713.
- **IMG_CAP_DEVICECONFIG –** Indicates whether the device allows the current device configuration (for the current source) to be persistent. If this flag is enabled, the device supports the following methods: **GetDeviceConfig**, described on page 780, **SetDeviceConfig**, described on page 777, **LoadDeviceConfig**, described on page 783, **SaveDeviceConfig**, described on page 784, **GetDeviceConfigFileExt**, described on page 787, and**GetDeviceConfigFileDesc**, described on page 789.
- **IMG_CAP_CONFIGDIALOG –** Indicates whether the device provides a configuration dialog box or property page that can be used to configure the device. If this flag is enabled, the device supports the **ShowDeviceConfigDialog** method, described on page 791.

Description (cont.):

- **IMG_CAP_CFGPERSOURCE –** Indicates whether the device configurations are supported on a per-source basis or on a per-device basis. If this flag is enabled, the device supports a separate device configuration for each source. If this flag is cleared, only a single configuration is supported for the entire device regardless of the current video input source.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to image device object.
CcImageDevice *pid;
//Integer to receive the device
//capabilities.
int nDevCaps;
//Holds the error text.
TCHAR szText[200];

//Determine whether the device
//supports live video
if (pid->GetDeviceCaps(&nDevCaps)
   <0)
{
  //Get error text
  //pid->GetErrorText(szText,
     200);
}
```

Example (cont.)

```
//Does the device support
//streaming?
if (nDevCaps & IMG_CAP_STREAMING)
{
  //Yes, streaming is supported.
}
else
{
  //No, streaming is not
  //supported.
}
```

IsDeviceCapSupported

Syntax

```
BOOL IsDeviceCapSupported(
   int nDeviceCap
   );
```

Include File C_PicTool.h

Description Determines whether the specified capability is supported by the current device.

Parameters

Name: nDeviceCap

Description: The device capability. The value can be one of the following:

- **IMG_CAP_TIMEOUT –** Indicates whether the device supports timeouts. If this flag is enabled, the device supports the following methods: **SetTimeout**, described on page 718, and **GetTimeout**, described on page 720.

Description (cont.):

- **IMG_CAP_IMAGESCALE –** Indicates whether the device supports scaling. If this flag is enabled, the device supports the following methods: **GetImageScaleLimits**, described on page 733, **GetImageScale**, described on page 722, **SetImageScale**, described on page 721, **GetHorzImageScale**, described on page 725, **SetHorzImageScale**, described on page 724, **GetVertImageScale**, described on page 727, **SetVertImageScale**, described on page 726, **GetScaledImageDims**, described on page 729, **GetScaledImageWidth**, described on page 730, and **GetScaledImageHeight**, described on page 731.
- **IMG_CAP_STREAMING –** Indicates whether the device supports streaming. If this flag is enabled, the device supports the following methods: **StartStreaming**, described on page 760, and **StopStreaming**, described on page 761.
- **IMG_CAP_LIVEVIDEO –** Indicates whether the device supports displaying live video in a window. If this flag is enabled, the device supports the following methods: **StartLiveVideo**, described on page 774, and **StopLiveVideo**, described on page 775.

Description (cont.):

- **IMG_CAP_DEVICEPROPS –** Indicates whether the device supports device-specific programmable properties. If this flag is enabled, the device supports the following methods: **SetDeviceProperty**, described on page 712, and **GetDeviceProperty**, described on page 713.
- **IMG_CAP_DEVICECONFIG –** Indicates whether the device allows the current device configuration (for the current source) to be persistent. If this flag is enabled, the device supports the following methods: **GetDeviceConfig**, described on page 780, **SetDeviceConfig**, described on page 777, **LoadDeviceConfig**, described on page 783, **SaveDeviceConfig**, described on page 784, **GetDeviceConfigFileExt**, described on page 787, and**GetDeviceConfigFileDesc**, described on page 789.
- **IMG_CAP_CONFIGDIALOG –** Indicates whether the device provides a configuration dialog box or property page that can be used to configure the device. If this flag is enabled, the device supports the **ShowDeviceConfigDialog** method, described on page 791.

Description (cont.):

- **IMG_CAP_CFGPERSOURCE –** Indicates whether the device configurations are supported on a per-source basis or on a per-device basis. If this flag is enabled, the device supports a separate device configuration for each source. If this flag is cleared, only a single configuration is supported for the entire device regardless of the current video input source.

Notes None

Return Values

TRUE The current device supports the specified capability.

FALSE The current device does not support the specified capability.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Does the device support
//streaming?
if (pid->IsDeviceCapSupported
   (IMG_CAP_STREAMING))
{
  //Yes, streaming is supported
}
```

SetDeviceProperty

Syntax

```
int SetDeviceProperty (
   int nPropId,
   int nValue
   );
```

Include File C_PicTool.h

Description Sets a vendor-specific property on a imaging device.

Parameters

Name: nPropId

Description: A vendor-specific value that identifies the property to set.

Name: nValue

Description: The desired value for the property.

Notes Supported properties vary from device to device. Refer to Appendix A starting on page 1091 for more information on the values for *nPropId* and *nValue*.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Error text buffer.
TCHAR szText[500];

//Set the vendor-specific
//property.
if (pid->SetDeviceProperty
   (FirstActivePixel, 100) < 0)
{
  //Get error text.
  pid->GetErrorText (szText, 500);
}
```

GetDeviceProperty

Syntax

```
int GetDeviceProperty (
   int nPropId,
   int* pnValue
   );
```

Include File C_PicTool.h

Description Returns the value of a vendor-specific property on an imaging device.

Parameters

Name: nPropId

Description: A vendor-specific value that identifies the property to get.

Name: nValue

Description: A pointer to a variable in which the value for the property is returned.

Notes

Supported properties vary from device to device. Refer to Appendix A starting on page 1091 for more information on the values for *nPropId* and *nValue*.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable that receives the
//property value
int nValue;
//Error text buffer.
TCHAR szText[500];

//Get the vendor-specific
//property.
if (pid->GetDeviceProperty
   (FirstActivePixel, &nValue) <
   0)
{
  //Get error text.
  pid->GetErrorText (szText, 500);
}
```

GetInputSourceCount

Syntax

```
int GetInputSourceCount(
   int* pnCount
   );
```

Include File C_PicTool.h

Description Returns the number of input sources that are supported by the current device.

Parameters

Name: pnCount

Description: A pointer to the integer variable that receives the source count. Values range from 1 to *n*, where *n* is the number of input sources supported by the imaging device.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to the image device
//object.
CcImageDevice *pid;
//Integer to receive source count.
int nCount;
//Holds error text.
TCHAR szText[200];
//Get the number of input sources
//provided by the device
```

Example (cont.)

```
if (pid->GetInputSourceCount(
   &nCount) < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

SetInputSource

Syntax

```
int SetInputSource(
   int nSource
   );
```

Include File C_PicTool.h

Description Sets the input source on the imaging device.

Parameters

Name: nSource

Description: The input source. The value can range from 0 to n - 1, where *n* is the number of input sources supported by the current device.

Notes The first input source supported by a device is always zero. Therefore, possible return channel values for a device that has four input channels are 0, 1, 2 and 3.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to the image device
//object.
CcImageDevice *pid;
//Holds the error text
TCHAR szText[200];
//Set the input source to
//source 0.
if (pid->SetInputSource(0)<0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

GetInputSource

Syntax

```
int GetInputSource(
   int* pnSource
   );
```

Include File

C_PicTool.h

Description

Returns the current input source from the imaging device.

Parameters

Name: pnSource

Description: A pointer to an integer variable that receives the current input source. The value can range from 0 to n - 1, where *n* is the number of input sources supported by the current device.

Notes

The first input source supported by a device is always zero. Therefore, possible return channel values for a device that has four input channels are 0, 1, 2 and 3.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to the image device
//object.
CcImageDevice *pid;
//Variable to receive the input
//source
int nSource
//Holds the error text
TCHAR szText[200];
//Get the input source.
if (pid->GetInputSource(&nSource)
   < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

SetTimeout

Syntax

```
int SetTimeout(
   int nTimeOut
   );
```

Include File C_PicTool.h

Description Sets the timeout period for acquire operations.

Parameters

Name: nTimeout

Description: The timeout period, in seconds.

Notes This method is available only if the device supports the IMG_CAP_TIMEOUT capability.

A timeout error is returned if the acquisition does not complete within the time specified by this method.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the timeout to 10 seconds.
if (pid->SetTimeout(10) < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

GetTimeout

Syntax

```
int GetTimeout(
   int *pnTimeOut
   );
```

Include File C_PicTool.h

Description Returns the current timeout period.

Parameters

Name: pnTimeout

Description: A pointer to an integer that receives the timeout value, in seconds.

Notes This method is available only if the device supports the IMG_CAP_TIMEOUT capability.

A timeout error is returned if the acquisition does not complete within the time specified by this method.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Get the timeout value.
if (pid->GetTimeout(&nTimeout) <
   0)
```

Example (cont.)

```
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

SetImageScale

Syntax

```
int SetImageScale(
   int nHorzScale,
   int nVertScale
   );
```

Include File C_PicTool.h

Description Sets the current horizontal and vertical scale factors for the output image.

Parameters

Name: nHorzScale

Description: An integer that specifies the horizontal scale factor. Values range from 0 to 100 percent.

Name: nVertScale

Description: An integer that specifies the vertical scale factor. Values range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the horizontal and vertical
//scale factors to 50%.
if (pid->SetImageScale(50,50) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageScale

Syntax

```
int GetImageScale(
   int *pnHorzScale,
   int *pnVertScale
   );
```

Include File C_PicTool.h

Description Returns the current horizontal and vertical scale factors for the output image.

Parameters

Name: pnHorzScale

Description: A pointer to an integer variable that receives the current horizontal scale factor. Values for the horizontal scale factor range from 0 to 100 percent.

Name: pnVertScale

Description: A pointer to an integer variable that receives the current vertical scale factor. Values for the vertical scale factor range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the scale
// values.
int nHorzScale, nVertScale;
//Holds error text.
TCHAR szText[200];
//Get the horizontal and vertical
//scale factors.
if (pid->GetImageScale
   (&nHorzScale, &nVertScale) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetHorzImageScale

Syntax

```
int SetHorzImageScale(
   int nHorzScale
   );
```

Include File C_PicTool.h

Description Sets the current horizontal scale factor for the output image.

Parameters

Name: nHorzScale

Description: An integer that specifies the horizontal scale factor. Values range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the horizontal scale factor
//to 50%.
```

Example (cont.)

```
if (pid->SetHorzImageScale(50) <
   0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetHorzImageScale

Syntax

```
int GetHorzImageScale(
   int *pnHorzScale
   );
```

Include File C_PicTool.h

Description Returns the current horizontal scale factor for the output image.

Parameters

Name: pnHorzScale

Description: A pointer to an integer variable that receives the current horizontal scale factor. Values for the horizontal scale factor range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the
//horizontal scale value.
int nHorzScale;
//Holds error text.
TCHAR szText[200];
//Get the horizontal scale factor
if (pid->GetHorzImageScale
   (&nHorzScale) < 0)
{
  //Get error text.
  //pid -> GetErrorText(szText,
   200);
}
```

SetVertImageScale

Syntax

```
int SetVertImageScale(
   int nVertScale
   );
```

Include File C_PicTool.h

Description Sets the current vertical scale factor for the output image.

Parameters

Name: nVertScale

Description: An integer that specifies the vertical scale factor. Values range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the vertical scale factor to
//50%.
if (pid->SetVertImageScale (50) <
   0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetVertImageScale

Syntax

```
int GetVertImageScale(
   int *pnVertScale
   );
```

Include File C_PicTool.h

Description Returns the current vertical scale factor for the output image.

Parameters

Name: pnVertScale

Description: A pointer to an integer variable that receives the current vertical scale factor. Values for the vertical scale factor range from 0 to 100 percent.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the vertical
//scale value.
int nVertScale;
//Holds error text.
TCHAR szText[200];
//Get the vertical scale factor
if (pid->GetVertImageScale(
   &nVertScale) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetScaledImageDims

Syntax

```
int GetScaledImageDims(
    int* pnWidth,
    int* pnHeight
    );
```

Include File C_PicTool.h

Description Returns the dimensions (width and height) of the output image after the current horizontal and vertical scale factors have been applied.

Parameters

Name: pnWidth

Description: A pointer to an integer variable that receives the scaled width of the image.

Name: pnHeight

Description: A pointer to an integer variable that receives the scaled height of the image.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0	The method was successful.
< 0	An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variables to receive scaled
//width and height
int nWidth, nHeight;
//Holds error text.
TCHAR szText[200];
//Get the scaled image dimensions
if (pid->GetScaledImageDims(
   &nWidth, &nHeight) < 0)
{
  //Get error text.
  //pid -> GetErrorText(szText,
   200);
}
```

GetScaledImageWidth

Syntax

```
int GetScaledImageWidth(
   int *pnWidth
   );
```

Include File C_PicTool.h

Description Returns the width of the image after the current horizontal scale factor has been applied

Parameters

Name: pnWidth

Description: A pointer to an integer variable that receives the scaled image width.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive scaled
//width.
int nWidth;
//Holds error text.
TCHAR szText[200];
//Get the scaled image width
if (pid->GetScaledImageWidth(
   &nWidth) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetScaledImageHeight

Syntax

```
int GetScaledImageHeight(
   int *pnHeight
   );
```

Include File C_PicTool.h

Description Returns the height of the image after the current vertical scale factor has been applied.

Parameters

Name: pnHeight

Description: A pointer to an integer variable that receives the scaled image height.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive scaled
//image height.
int nHeight;
//Holds error text.
TCHAR szText[200];
//Get the scaled height
if (pid->GetScaledImageHeight(
   &nHeight) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageScaleLimits

Syntax

```
int GetImageScaleLimits(
   PDL_IMAGE_SCALE_LIMITS *pLimits
   );
```

Include File C_PicTool.h

Description Returns the operating limits on image scaling.

Parameters

Name: pLimits

Description: A pointer to the PDL_IMAGE_SCALE_LIMITS structure that receives the scaling limits.

Notes This method is available only if the device supports the IMG_CAP_IMAGESCALE capability.

This PDL_IMAGE_SCALE_LIMITS structure describes the limits on image scale for a given imaging device. The PDL_IMAGE_SCALE_LIMITS structure is defined as follows:

```
typedef struct
PDL_IMAGE_SCALE_LIMITS
{
// Minimum horizontal scale.
int nMinHScale;
// Maximum horizontal scale.
int nMaxHScale;
// Step by which to increment.
int nIncHScale;
// Default horizontal scale.
int nDefHScale;
// Minimum vertical scale.
int nMinVScale;
```

Notes (cont.)

```
// Maximum vertical scale.
int nMaxVScale;
// Step by which to increment.
int nIncVScale;
// Default vertical scale.
int nDefVScale;
PDL_IMAGE_SCALE_LIMITS;
```

The *nMinHScale* and *nMaxHScale* members hold the minimum and maximum allowable horizontal scale factors for the imaging device. *nIncHScale* specifies the amount by which the horizontal scale factor can be incremented and decremented. *nDefHScale* always holds the default horizontal scale factor for the device. *nMinVScale*, *nMaxVScale*, *nIncVScale* and *nDefVScalet* describe the limits on vertical image scale.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the scaling
//limits.
PDL_IMAGE_SCALE_LIMITS Limits;
//Holds error text.
TCHAR szText[200];
//Get the limits on image scaling
```

Example (cont.)

```
if (pid->GetImageScaleLimits(
   &Limits) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetImageDims

Syntax

```
int SetImageDims(
   int nWidth,
   int nHeight
   );
```

Include File C_PicTool.h

Description Sets the current width and height of the output image.

Parameters

Name: nWidth

Description: An integer that specifies the width of the image.

Name: nHeight

Description: An integer that specifies the height of the image.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the output image size to 640
//x 480 pixels.
if (pid->SetImageDims(640, 480) <
   0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageDims

Syntax

```
int GetImageDims(
   int *pnWidth,
   int *pnHeight
   );
```

Include File C_PicTool.h

Description Returns the current width and height of the output image.

Parameters

Name: pnWidth

Description: A pointer to an integer variable that receives the current width of the image.

Name: pnHeight

Description: A pointer to an integer variable that receives the current height of the image.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variables to receive the image
//width and height
int nWidth, nHeight;
//Holds error text.
TCHAR szText[200];
//Get the current image
//dimensions.
if (pid->GetImageDims (&nWidth,
   &nHeight) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetImageWidth

Syntax

```
int SetImageWidth(
   int nWidth
   );
```

Include File C_PicTool.h

Description Sets the width of the output image.

Parameters

Name: nWidth

Description: An integer that specifies the image width.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the output image width to
//640 pixels.
if (pid->SetImageWidth(640) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageWidth

Syntax

```
int GetImageWidth(
   int *pnWidth
   );
```

Include File C_PicTool.h

Description Returns the current width of the output image.

Parameters

Name: pnWidth

Description: A pointer to an integer variable that receives the current image width.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the image
//width
int nWidth;
//Holds error text.
TCHAR szText[200];
//Get the current image width.
if (pid->GetImageWidth (&nWidth)
   < 0)
```

Example (cont.)

```
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetImageHeight

Syntax

```
int SetImageHeight(
   int nHeight
   );
```

Include File C_PicTool.h

Description Sets the height of the output image.

Parameters

Name: nHeight

Description: An integer that specifies the image height.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
```

Example (cont.)

```
//Set the output image height to
//480 pixels.
if (pid->SetImageHeight (480) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageHeight

Syntax

```
int GetImageHeight(
   int *pnHeight
   );
```

Include File C_PicTool.h

Description Returns the current height of the output image.

Parameters

Name: pnHeight

Description: A pointer to an integer variable that receives the current image height.

Notes None

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the image
//height
int nHeight;
//Holds error text.
TCHAR szText[200];
//Get the current image height.
if (pid->GetImageHeight (&nHeight)
   < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageDimsLimits

Syntax

```
int GetImageDimsLimits (
   PDL_IMAGE_DIMS_LIMITS* pLimits
   );
```

Include File C_PicTool.h

Description Returns the operating limits for the image dimensions.

Parameters

Name: pLimits

Description: A pointer to the PDL_IMAGE_DIMS_LIMITS structure that receives the image height, width, and type.

Notes This PDL_IMAGE_DIMS_LIMITS structure describes the limits on image dimensions for a given imaging device. The PDL_IMAGE_DIMS_LIMITS structure is defined as follows:

```
typedef struct
   PDL_IMAGE_DIMS_LIMITS
{
// Minimum image width.
int nMinWidth;
// Maximum image width.
int nMaxWidth;
//Width increment value.
int nIncWidth;
// Default image width.
int nDefWidth;
// Minimum image height.
int nMinHeight;
// Maximum image height.
int nMaxHeight;
// Height increment value.
int nIncHeight;
// Default image height.
int nDefHeight;

} PDL_IMAGE_DIMS_LIMITS;
```

The *nMinWidth* and *nMaxWidth* members hold the minimum and maximum allowable width (in pixels) of images produced by the device. *nIncWidth* specifies the amount by which the image width can be incremented and decremented. *nDefWidth* always holds the default image width for the device. *nMinHeight*, *nMaxHeight*, *nIncHeight* and *nDefHeight* describe the limits on image height.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the
//dimension limits
PDL_IMAGE_DIMS_LIMITS Limits;
//Holds error text.
TCHAR szText[200];
//Get the limits on image
//dimensions.
if (pid->GetImageDimsLimits(
   &Limits) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetImageType

Syntax

```
int SetImageType(
   int nImageType
   );
```

Include File C_PicTool.h

Description Sets the type of output image to use.

Parameters

Name: nImageType

Description: An integer that specifies the type of output image. Possible values are as follows:

- IMAGE_TYPE_08BIT_GS - An 8-bit grayscale image.
- IMAGE_TYPE_32BIT_GS - A 32-bit grayscale image.
- IMAGE_TYPE_16BIT_GS - A 16-bit grayscale image.
- IMAGE_TYPE_FLOAT_GS - A floating-point grayscale image.
- IMAGE_TYPE_24BIT_RGB - A 24-bit RGB color image.
- IMAGE_TYPE_BINARY - A binary image (one byte per pixel; pixel values can be 0 or 1).
- IMAGE_TYPE_24BIT_HSL - A 24-bit HSL color image.

Notes

This method is provided for backward compatibility with existing APIs.

An imaging device may support only a subset of the image types listed.

Notes (cont.)

The image type flags used with **GetImageType**, described on page 747, and **SetImageType**, described on page 744, have different numeric values than the flags used with **GetImageTypeEx**, described on page 751, and **SetImageTypeEx**, described on page 749. Thus, the flags for the *nImageType* parameter should be used with the **GetImageType** and **SetImageType** methods only.

Return Values

0	The method was successful.
< 0	An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the current image to 8-bit
//grayscale.
if (pid->SetImageType(
   IMAGE_TYPE_08BIT_GS) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

20

GetImageType

Syntax

```
int GetImageType(
   int *pnImageType
   );
```

Include File C_PicTool.h

Description Returns the current output image type.

Parameters

Name: pnImageType

Description: A pointer to the integer variable that receives the image type. Possible values for image type are as follows:

- IMAGE_TYPE_08BIT_GS - An 8-bit grayscale image.
- IMAGE_TYPE_32BIT_GS - A 32-bit grayscale image.
- IMAGE_TYPE_16BIT_GS - A 16-bit grayscale image.
- IMAGE_TYPE_FLOAT_GS - A floating-point grayscale image.
- IMAGE_TYPE_24BIT_RGB - A 24-bit RGB color image.
- IMAGE_TYPE_BINARY - A binary image (one byte per pixel; pixel values can be 0 or 1).
- IMAGE_TYPE_24BIT_HSL - A 24-bit HSL color image.

Notes This method is provided for backward compatibility with existing APIs.

Notes (cont.)

An imaging device may support only a subset of the image types listed.

The image type flags used with **GetImageType**, described on page 747, and **SetImageType**, described on page 744, have different numeric values than the flags used with **GetImageTypeEx**, described on page 751, and **SetImageTypeEx**, described on page 749. Thus, the flags for the *pnImageType* parameter should be used with the **GetImageType** and **SetImageType** methods only.

Return Values

0	The method was successful.
< 0	An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the image
//type
int nImageType;
//Holds error text.
TCHAR szText[200];
//Get the current image type.
if (pid->GetImageType
   (&nImageType) < 0)
```

Example (cont.)

```
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

SetImageTypeEx

Syntax

```
int SetImageTypeEx(
   int nImageTypeEx
   );
```

Include File C_PicTool.h

Description Specifies the type of output image to use.

Parameters

Name: nImageTypeEx

Description: An integer that specifies the type of output image. Possible values are as follows:

- IMG_TYPE_08BIT_GS - An 8-bit grayscale image.
- IMG_TYPE_32BIT_GS - A 32-bit grayscale image.
- IMG_TYPE_16BIT_GS - A 16-bit grayscale image.
- IMG_TYPE_FLOAT_GS - A floating-point grayscale image.
- IMG_TYPE_24BIT_RGB - A 24-bit RGB color image.
- IMG_TYPE_BINARY - A binary image (one byte per pixel; pixel values can be 0 or 1).

Description (cont.):

- IMAGE_TYPE_24BIT_HSL - A 24-bit HSL color image.

Notes

An imaging device may support only a subset of the image types listed.

The image type flags used with **GetImageType**, described on page 747, and **SetImageType**, described on page 744, have different numeric values than the flags used with **GetImageTypeEx**, described on page 751, and **SetImageTypeEx**, described on page 749. Thus, the flags for the *nImageTypeEx* parameter should be used with the **GetImageTypeEx** and **SetImageTypeEx** methods only.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Set the current image to 8-bit
//grayscale.
if (pid->SetImageTypeEx
   (IMG_TYPE_08BIT_GS) < 0)
```

Example (cont.)

```
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageTypeEx

Syntax

```
int GetImageTypeEx(
   int *pnImageTypeEx
   );
```

Include File C_PicTool.h

Description Returns the current output image type.

Parameters

Name: pnImageTypeEx

Description: A pointer to an integer variable that receives the current image type. Possible values for image type are as follows:

- IMG_TYPE_08BIT_GS - An 8-bit grayscale image.
- IMG_TYPE_32BIT_GS - A 32-bit grayscale image.
- IMG_TYPE_16BIT_GS - A 16-bit grayscale image.
- IMG_TYPE_FLOAT_GS - A floating-point grayscale image.
- IMG_TYPE_24BIT_RGB - A 24-bit RGB color image.

Description (cont.):

- IMG_TYPE_BINARY - A binary image (one byte per pixel; pixel values can be 0 or 1).
- IMG_TYPE_24BIT_HSL - A 24-bit HSL color image.

Notes

An imaging device may support only a subset of the image types listed.

The image type flags used with **GetImageType**, described on page 747, and **SetImageType**, described on page 744, have different numeric values than the flags used with **GetImageTypeEx**, described on page 751, and **SetImageTypeEx**, described on page 749. Thus, the flags for the *pnImageTypeEx* parameter should be used with the **GetImageTypeEx** and **SetImageTypeEx** methods only.

Return Values

0	The method was successful.
< 0	An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive the image
//type
int nImageTypeEx;
//Holds error text.
TCHAR szText[200];
//Get the current image type.
```

Example (cont.)

```
if (pid->GetImageTypeEx (
   &nImageTypeEx) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetImageTypeLimits

Syntax

```
int GetImageTypeLimits(
   PDL_IMAGE_TYPE_LIMITS* pLimits
   );
```

Include File C_PicTool.h

Description Returns the limits on the image types supported by the device.

Parameters

Name: pLimits

Description: A pointer to the PDL_IMAGE_TYPE_LIMITS structure that receives the image type limits.

Notes This PDL_IMAGE_TYPE_LIMITS structure describes the image types or formats that are supported by a given imaging device. The PDL_IMAGE_SCALE_LIMITS structure is defined as follows:

```
typedef struct
PDL_IMAGE_TYPE_LIMITS
{
// Holds supported image types.
int nImgTypes;
```

Notes (cont.)

```
// Holds default image type.
int nDefType;
} PDL_IMAGE_TYPE_LIMITS;
```

nImgTypes is a bitfield that contains one or more of the following image type flags:

- IMG_TYPE_08BIT_GS –8-bit grayscale.
- IMG_TYPE_16BIT_GS –16-bit grayscale.
- IMG_TYPE_32BIT_GS –32-bit grayscale.
- IMG_TYPE_FLOAT_GS –Floating-point grayscale.
- IMG_TYPE_24BIT_RGB –24-bit RGB.
- IMG_TYPE_24BIT_HSL –24-bit HSL.
- IMG_TYPE_BINARY –Binary image.

nDefType holds the default image type for the device.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Variable to receive limits on
//the image type
PDL_IMAGE_TYPE_LIMITS Limits;
//Holds error text.
TCHAR szText[200];
```

Example (cont.)

```
//Get the limits on image type.
if (pid->GetImageTypeLimits(
   &Limits) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

GetCompatibleImage

Syntax

```
int GetCompatibleImage(
   CcImage **ppImage
   );
```

Include File C_PicTool.h

Description Returns an image object that is compatible with the current output format (image type, width, and height) of the imaging device.

Parameters

Name: ppImage

Description: A pointer to a pointer to a CcImage object that will contain the newly created image object.

Notes You can use the returned image object in subsequent calls to **AcquireImage**, described on page 765.

Make sure that you free all image objects obtained through calls to this method.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Pointer to receive the image
//object.
CcImage *pImage;
//Holds error text.
TCHAR szText[200];
//Get an image object that is
//compatible with the current
//image output configuration of
//the device.
if (pid->GetCompatibleImage(
   &pImage) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
//Do something with the image and
//free when done
delete pImage;
```

SetImageAverage

Syntax

```
int SetImageAverage(
   int nImageCount
   );
```

Include File C_PicTool.h

Description Specifies the number of images that you want to acquire from the device and average when you call **AcquireImage**.

Parameters

Name: nImageCount

Description: The number of images that you want to acquire from the device and average. By default, a single image is acquired from the device.

Notes **AcquireImage**, described on page 765, always returns the average of the acquired images in a single image object.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Each time AcquireImage is
//called, acquire four images,
```

Example (cont.)

```
//average them, and return
//the result.
if (pid->SetImageAverage(4) < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

EnableTimeStamping

Syntax

```
int EnableTimeStamping(
   BOOL bEnable
   );
```

Include File C_PicTool.h

Description Enables or disables time stamping of images.

Parameters

Name: bEnable

Description: Specifies whether to enable or disable time stamping of images. If TRUE, time stamping is enabled. If FALSE, time stamping is disabled.

Notes When time stamping is enabled, a time stamp is added to the lower-left corner of all images that are acquired from the device. When time stamping is disabled, no time stamp is added.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Enable time stamping on the
//device.
if (pid->EnableTimeStamping (TRUE)
   < 0)
{
  //Get error text.
  pid -> GetErrorText(szText,
   200);
}
```

IsTimeStampingEnabled

Syntax `BOOL IsTimeStampingEnabled();`

Include File C_PicTool.h

Description Determines whether time stamping is currently enabled on the 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:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Is time stamping enabled?
if (pid->IsTimeStampingEnabled() )
{
  //Yes, time stamping is enabled.
}
else
{
  //No, time stamping is disabled.
}
```

StartStreaming

Syntax `int StartStreaming();`

Include File C_PicTool.h

Description Starts streaming on the device. Images are continuously streamed as they become available.

Parameters None

Notes This method is available only if the device supports the IMG_CAP_STREAMING capability. For some imaging devices, the use of streaming can considerably increase the rate at which images are acquired.

Once streaming has been started, a program must still call **AcquireImage**, described on page 765, to obtain images from the device.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Does the device support
//streaming?
if (pid->IsDeviceCapSupported(
   IMG_CAP_STREAMING) )
{
  //Yes, streaming is supported so
  //start it.
  if (pid->StartStreaming() < 0)
  {
    //Get error text.
    pid->GetErrorText(szText,
   200);
  }
}
```

StopStreaming

Syntax `int StopStreaming();`

Include File C_PicTool.h

Description Stops streaming on the device.

Parameters None

Notes This method is available only if the device supports the IMG_CAP_STREAMING capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Does the device support
//streaming?
if (pid->IsDeviceCapSupported(
   IMG_CAP_STREAMING) )
{
  //Yes, streaming is supported,
  //but is streaming already in
  //progress?
  if (pid->IsStreamingInProgress()
   < 0)
  {
    //Yes, it is, so stop it.
    if (pid->StopStreaming() < 0)
    {
      //Get error text.
      pid->GetErrorText(szText,
        200);
    }
  }
}
```

WaitForImage

Syntax `int WaitForImage();`

Include File C_PicTool.h

Description Waits for an image to become available when the device is running in streaming mode.

Parameters None

Notes This method is available only if the device supports the IMG_CAP_STREAMING capability.

Streaming must be running before you can call this method.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Start streaming
if (pid->StartStreaming() )
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
//Wait for an image to become
//available?
```

Example (cont.)

```
if (pid->WaitForImage() )
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

IsStreamingInProgress

Syntax `BOOL IsStreamingInProgress();`

Include File C_PicTool.h

Description Returns whether streaming is currently in progress on the device.

Parameters None

Notes None

Return Values

TRUE	Streaming is currently in progress on the device.
FALSE	Streaming is currently not in progress on the device.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Is streaming in progress?
if (pid->IsStreamingInProgress() )
{
  //Yes, streaming is in progress.
}
```

Example (cont.)

```
else
{
  //No, streaming is not in
  //progress.
}
```

AcquireImage

Syntax

```
int AcquireImage(
   CcImage *pImage
   );
```

Include File C_PicTool.h

Description Acquires an image from the device and returns it in the supplied image object.

Parameters

Name: pImage

Description: A pointer to an image object that was previously obtained through the **GetCompatibleImage** method, described on page 755.

Notes A program can obtain an image object for use with **AcquireImage**, described on page 765, by calling **GetCompatibleImage**. **GetCompatibleImage**, described on page 755, retrieves an image object that is compatible with the output configuration (image type, width, and height) of the device.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Pointer that receives image
//object.
CcImage *pImage;
//Holds error text.
TCHAR szText[200];
//Get a compatible image object.
if (pid->GetCompatibleImage(
   &pImage) < 0 )
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
//Acquire an image from the
//device
if (pid->AcquireImage(pImage) < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
//Do something with the image and
//free when done.
delete pImage;
```

TimedAcquireToAVI

Syntax

```
int TimedAcquireToAVI(
   CcAVI pAVI,
   int nImageCount,
   int nTimeDelay
   );
```

Include File C_PicTool.h

Description Acquires one or more images to an AVI file with an optional delay between consecutive images.

Parameters

Name: pAVI

Description: A pointer to an object of type CcAVI.

Name: nImageCount

Description: The number of images that you want to acquire to the AVI file. The value must be greater than or equal to one.

Name: nTimeDelay

Description: The delay between images, in milliseconds, that is used when generating the AVI file. This value must be greater or equal to zero.

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 (image type and dimensions) of the device.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//AVI object.
CcAVI Avi;
//Holds image type.
int nImageType;
//Holds image width and height.
int nWidth, nHeight;
//Holds error text.
TCHAR szText[200];
//Get the output image type.
if (pid->GetImageType(&nImageType)
   < 0 )
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
//Get the image dimensions
if (pid->GetImageDims(&nWidth,
   &nHeight) < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

Example (cont.)

```
//Create an AVI object that is
//compatible with the output
//configuration of the device.
AVI.Create ("C:\\AviFile.avi",
   nImageType, nWidth, nHeight);
//Acquire 40 images to the AVI
//file
if (pid->TimedAcquireToAvi(&Avi,
   40, 0) < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

TimedAcquireToDisc

Syntax

```
int TimedAcquireToDisc(
   LPCSTR szDir,
   LPCSTR szBaseFileName,
   int nImageCount,
   int nCountStart,
   int nTimeDelay
   );
```

Include File C_PicTool.h

Description Acquires one or more images to bitmap files.

Parameters

Name: szDir

Description: A NULL-terminated constant string that specifies the directory in which you want to place the generated bitmap files.

Name:	szBaseFileName
Description:	A NULL-terminated constant string that specifies the base file name to use for all generated bitmap files.
Name:	nImageCount
Description:	The number of images that you want to acquire and write to disk. The value must be greater than or equal to one.
Name:	nCountStart
Description:	The first counter number that you want to append to the base file name when you start generating the bitmap files.
Name:	nTimeDelay
Description:	The delay between images, in milliseconds, used when generating the bitmap files. The value must be greater than or equal to zero.

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, *nImageCount* = 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 method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Acquire 40 images to disk.
if (pid->TimedAcquireToDisc(
   "C:\Temp","MyBitmap",40,0,0)
   < 0)
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

TimedAcquireToMemory

Syntax

```
int TimedAcquireToMemory(
  LPCSTR szBaseImageName,
  int nImageCount,
  int nCountStart,
  int nTimeDelay,
  CcList *pImageList
  );
```

Include File C_PicTool.h

Description Acquires one or more images to image objects 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: nImageCount

Description: The number of images that you want to acquire from the device. This value must be greater than or equal to one. The maximum value is determined by the amount of memory available in your system.

Name: nCountStart

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.

Name: nTimeDelay

Description: The delay between images, in milliseconds, used when generating the bitmap files. The value must be greater than or equal to zero.

Name: pImageList

Description: A pointer to a CcList object that receives the image objects that are generated by this method. Any existing objects in the specified list are deleted. The value must not 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, *nImageCount* = 4, and *szBaseFileName* = "MyImage," the tool captures four images and saves them as MyImage4, MyImage5, MyImage6, and MyImage7.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example

The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//List to hold acquired images.
CcList *pList;
//Holds error text.
TCHAR szText[200];
//Acquire 40 images to disk at
//100 ms intervals. Images will be
//named Image0, Image1, Image2,
//and so on.
if (pid->TimedAcquireToMemory(
   "Image", 40, 0, 100, pList) < 0
   )
```

Example (cont.)

```
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

StartLiveVideo

Syntax

```
int StartLiveVideo(
   HWND hWindow);
```

Include File C_PicTool.h

Description Starts live video in the specified window.

Parameters

Name: hWindow

Description: A handle to the window in which to display the live video. The value cannot be NULL.

Notes Live video provides an application with the ability to view a live video image for the purpose of focusing cameras, and so on.

This method is available only if the device indicates support for the IMG_CAP_LIVEVIDEO capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Window to display live video.
HWND hWindow;
//Holds error text.
TCHAR szText[200];
//Fill in hWindow with a valid
//window handle
hWindow = <some window handle>;
//Start live video
if (pid->StartLiveVideo(hWindow) <
   0 )
{
  //Get error text.
  pid->GetErrorText(szText, 200);
}
```

StopLiveVideo

Syntax `int StopLiveVideo();`

Include File C_PicTool.h

Description Stops live video if it is currently running.

Parameters None

Notes This method is available only if the device indicates support for the IMG_CAP_LIVEVIDEO capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//See if live video is currently
//running
if (pid->IsLiveVideoRunning() )
{
  //Stop live video.
  if (pid->StopLiveVideo() < 0)
  {
    //Get error text.
    pid->GetErrorText(szText,
     200);
  }
}
```

IsLiveVideoRunning

Syntax `BOOL IsLiveVideoRunning();`

Include File C_PicTool.h

Description Determines whether live video is currently running.

Parameters None

Notes None

Return Values

TRUE Live video is currently running.

FALSE Live video is not currently running.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Is live video currently running
if (pid->IsLiveVideoRunning() )
{
  //Yes, live video is running.
}
else
{
  //No, live video is not
  //currently running.
);
```

SetDeviceConfig

Syntax

```
int SetDeviceConfig(
   LPSTREAM pStream
   );
```

Include File C_PicTool.h

Description Restores the device configuration for the currently selected video input source

Parameters

Name: pStream

Description: A pointer to a STREAM object that contains the device configuration to restore.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Stream object for saving
//configurations.
STREAM *pStream;
//Holds the video input source
//count.
int nCount;
//Initialize a stream that
//contains the device
//configuration
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
```

Example (cont.)

```
{
  //Yes. Does the device support a
  //separate configuration for
  //each source?
  if (pid->IsDeviceCapSupported(
   IMG_CAP_CFGPERSOURCE) )
  {
    //Yes. Get the number of
    //sources.
    if (pid->GetInputSourceCount
       (&nCount) < 0 )
    {
      //Handle error.
    }
  //Restore the device
  //configuration for each source.
  for (int i = 0; i < nCount; i++)
  {
    //Set the input source.
    if (pid->SetInputSource(i)< 0)
    {
      //Handle error.
    }
    //Restore device config for
    //current source.
    if (pid->SetDeviceConfig(
      pStream) < 0 )
    {
      //Handle error.
    }
  }
}
else
  {
    //No. Restore the
    //configuration for the
    //current source since
```

Example (cont.)

```
        //the same configuration is
        //shared by all sources.
        if (pid->SetDeviceConfig(
         pStream) < 0 )
        {
          //Handle error.
        }
      }
    }
    //Dispose of the stream object.
```

GetDeviceConfig

Syntax

```
int GetDeviceConfig(
   LPSTREAM pStream
   );
```

Include File C_PicTool.h

Description Returns the device configuration for the currently selected video input source.

Parameters

Name: pStream

Description: A pointer to a STREAM object that receives the device configuration.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Stream object for saving
//configurations.
STREAM *pStream;
//Holds the video input source
//count.
int nCount;
//Create a stream for saving the
//device configuration
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
{
  //Yes. Does the device support a
  //separate configuration
  //for each source?
  if (pid->IsDeviceCapSupported(
   IMG_CAP_CFGPERSOURCE) )
  {
    //Yes. Get the number of
    //sources.
    if (pid->GetInputSourceCount(
      &nCount) < 0 )
    {
      //Handle error.
    }
    //Save the device
    //configuration for each
    //source.
    for (int i = 0; i < nCount;
   i++)
```

Example (cont.)

```
        {
          //Set the input source.
          if (pid->SetInputSource
         (i) < 0 )
          {
            //Handle error.
          }
          //Save device config for
          //current source.
          if (pid->GetDeviceConfig(
            pStream) < 0 )
          {
            //Handle error.
          }
        }
      }
      else
      {
      //No. Save the configuration
      //for the current source
      //since the same
      //configuration is shared
      //by all sources.
      if (pid->GetDeviceConfig(
       pStream) < 0 )
        {
          //Handle error.
        }
      }
    }
    //Do something with the
    //configuration data in the
    //stream and dispose of the
    //stream object.
```

LoadDeviceConfig

Syntax

```
int LoadDeviceConfig(
   LPCSTR szFileName
   );
```

Include File C_PicTool.h

Description Loads the device configuration for the current input source from a file.

Parameters

Name: szFileName

Description: A NULL-terminated string that identifies the configuration file to load.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Holds configuration file
//extension.
TCHAR szExt[20];
//Holds configuration file name.
TCHAR szFileName[100];
```

Example (cont.)

```
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
{
  //Get the configuration file
  //extension.
  if (pid->GetDeviceConfigFileExt
   (szExt, 20) < 0 )
  {
    //Get error text and handle
    //error.
    pid->GetErrorText(szText, 200)
  }
  //Create configuration file name
  wsprintf(szFileName, "%s.%s",
   "MyConfig", szExt);
  //Load the device configuration
  //for the current source.
  if (pid->LoadDeviceConfig(
   szFileName) < 0 )
  {
    //Get error text and handle
    //error.
    pid->GetErrorText(szText, 200)
   }
}
```

SaveDeviceConfig

Syntax

```
int SaveDeviceConfig(
   LPCSTR szFileName
   );
```

Include File C_PicTool.h

Description Saves the device configuration for the current input source to a file.

Parameters

Name: szFileName

Description: A NULL-terminated string that identifies the configuration file to create.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

A program should call **GetDeviceConfigFileExt**, described on page 787, to retrieve the three-character configuration file extension that is associated with the device, and use this extension for all configuration files that are generated. This allows device configuration files to be uniquely identified.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Holds configuration file
//extension.
TCHAR szExt[20];
```

Example (cont.)

```
//Holds configuration file name.
TCHAR szFileName[100];
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
{
  //Get the configuration file
  //extension.
  if (pid->GetDeviceConfigFileExt(
   szExt, 20) < 0 )
  {
    //Get error text and handle
    //error.
    pid->GetErrorText(szText,200)
  }
  //Create configuration file name
  wsprintf(szFileName, "%s.%s",
   "MyConfig", szExt);
  //Save the device configuration
  //for the current source.
  if (pid->SaveDeviceConfig(
   szFileName) < 0 )
  {
    //Get error text and handle
    //error.
    pid->GetErrorText(szText,200)
  }
}
```

GetDeviceConfigFileExt

Syntax

```
int GetDeviceConfigFileExt(
   LPSTR szFileExt,
   int nBufSize
   );
```

Include File C_PicTool.h

Description Returns the three-character configuration file extension for the device.

Parameters

Name: szFileExt

Description: A character buffer that is large enough to hold the three-character extension (plus a NULL-termination character) that is returned by this method.

Name: nBufSize

Description: An integer that specifies the size, in characters, of the buffer that receives the device configuration file extension.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

As an example of using this method, assume that this method returned the characters "C52" for a DT3152 board. This extension could then be appended to a base file name to generate unique file names for the configuration.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Holds configuration file
//extension.
TCHAR szExt[20];
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
{
  //Get the configuration file
  //extension.
  if (pid->GetDeviceConfigFileExt(
   szExt, 20) < 0 )
  {
  //Get error text and handle
  //error.
  pid->GetErrorText(szText,200)
  }
}
```

GetDeviceConfigFileDesc

Syntax

```
int GetDeviceConfigFileDesc(
   LPSTR szFileDesc,
   int nBufSize
   );
```

Include File C_PicTool.h

Description Returns a short description of the configuration file for the device.

Parameters

Name: szFileDesc

Description: A character buffer that is large enough to hold the device configuration description that is returned by this method.

Name: nBufSize

Description: An integer that specifies the size, in characters, of the buffer that receives the device configuration description.

Notes This method is available only if the device supports the IMG_CAP_DEVICECONFIG capability.

The returned string is primarily intended for use in the Open/Save file dialog boxes. For example, a call to this method might return the characters “DT3152 Config Files” for a DT3152 board.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Holds configuration file
//description.
TCHAR szDesc[20];
//Does the device support
//configuration persistence?
if (pid->IsDeviceCapSupported(
   IMG_CAP_DEVICECONFIG) )
{
  //Get the configuration file
   description.
  if
   (pid->GetDeviceConfigFileDesc(
     szDesc, 20) < 0 )
  {
    //Get error text and handle
    //error.
    pid->GetErrorText(szText,200)
  }
}
```

ShowDeviceConfigDialog

Syntax

```
int ShowDeviceConfigDialog(
   HWND hParent
   );
```

Include File C_PicTool.h

Description Displays the configuration dialog box for the device.

Parameters

Name: hParent

Description: A handle to the window that serves as the parent of the device configuration dialog box.

Notes This method is available only if the device supports the IMG_CAP_CONFIGDIALOG capability.

The configuration dialog box allows you to configure the settings for a device. This dialog box is device-specific; therefore, different device or plug-in combinations may have different option dialog boxes.

Return Values

0 The method was successful.

< 0 An error occurred. Use **GetErrorText**, described on page 792, to return a description of the error.

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
```

Example (cont.)

```
//Holds configuration file
//description.
TCHAR szDesc[20];
//Holds error text.
TCHAR szText[200];
//Handle to the window.
HWND hParent;
//Fill in with a valid window
//handle.
HParent = <some window handle>;
//Does the device provide a
//configuration dialog box?
if (pid->IsDeviceCapSupported(
   IMG_CAP_CONFIGDIALOG) )
{
  //Display the dialog box.
  if (pid->ShowDeviceConfigDialog(
   hParent) < 0 )
  {
  //Get error text and handle
  //error.
  pid->GetErrorText(szText, 200)
  }
}
```

GetErrorText

Syntax

```
void GetErrorText(
   LPSTR szErrorText,
   int nBufSize
   );
```

Include File C_PicTool.h

Description Returns a description of the last error that occurred.

Parameters

Name: szErrorText

Description: A character buffer that receives the text associated with the last error generated. This value must not be NULL.

Name: nBufSize

Description: The size of the supplied character buffer.This value must not be zero.

Notes None

Return Values None

Example The following is a sample code fragment:

```
//Pointer to an image device
//object.
CcImageDevice *pid;
//Holds error text.
TCHAR szText[200];
//Get error text and handle error.
pid->GetErrorText(szText, 200)
```

21

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 DT Vision Foundry object) to the specified value for the given image (CcImage DT Vision Foundry 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 38.

Table 38: 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: A pointer to an image that is derived from the CcImage class on which to perform the pixel change operation.

Name: CRoi

Description: A pointer to the ROI object that defines the area in which to perform the operation.

Name: fNewValue

Description: The new grayscale value for the pixels inside the ROI. If *CImage* is a 24-bit RGB image, the area defined by the ROI is filled with the following colors: red = *fNewValue*, green = *fNewValue*, and blue = *fNewValue*.

Return Values

< 0 Method failed.

0 Method was successful.

Example

```
//Grayscale image object
CcGrayImage256* pImage;
//Rectangular ROI object
CcRoiRect* pRoi;
//Change tool API object
CcChange API;

//Use structured exception
//handling
_try
{
  //Create a new image object.
  //Exit on failure.
  if ( !(pImage =
   new CcGrayImage256) )
    return FALSE;

  //Create a new rectangular ROI
  //object
  if ( !(pRoi = new CcRoiRect) )
    return FALSE;

  //Configure ROI coordinates
  RECT rcBounds = { 50, 150, 150,
   50};

  //Set coordinates to ROI object
  if (pRoi->SetRoiImageCord(
   &rcBounds) < 0)
    return FALSE;

  //Open a 640x480, 8-bit
  //grayscale bitmap
  if (pImage->OpenBMPFile(
   "MyImage.bmp") < 0)
    return FALSE;
```

Example (cont.)

```
  //Change the rectangle to
  //grayscale value 128
  if (API.Change(pImage, pRoi,
   128) < 0)
    return FALSE;
}
_finally
{
  //Clean up before leaving
  if (pImage)
   delete pImage;
  if (pRoi)
   delete pRoi;
}
```


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: A pointer to the 24-bit RGB image on which to perform the pixel change operation.

Name: CRoi

Description: A pointer to the ROI object that defines the area in which to perform the operation.

Name: bRed

Description: The new red value for the RGB pixels in the ROI area.

Name: bGreen

Description: The new green value for the RGB pixels in the ROI area.

Name: bBlue

Description: The new blue value for the RGB pixels in the ROI area.

Return Values

< 0 Operation failed.

0 Successful.

Example

```
//24-bit RGB image object
Cc24BitRGBImage* pImage;
//Rectangular ROI object
CcRoiRect* pRoi;
//Change tool API object
CcChange API;

//Use structured exception
//handling
_try
{
  //Create a new image object.
  //Exit on failure.
  if ( !(pImage =
   new Cc24BitRGBImage) )
    return FALSE;
```

Example (cont.)

```
    //Create a new rectangular ROI
    //object
    if ( !(pRoi = new CcRoiRect) )
      return FALSE;

    //Configure ROI coordinates
    RECT rcBounds = { 50, 150, 150,
     50};

    //Set coordinates to ROI object
    if (pRoi->SetRoiImageCord(
     &rcBounds) < 0)
      return FALSE;

    //Open a 640x480, 24-bit
    //RGB bitmap
    if (pImage->OpenBMPFile(
     "MyImage.bmp")< 0)
      return FALSE;
    //Change the rectangle to
    //bright red
    if (API.ChangeRGB(pImage, pRoi,
     255, 0, 0) < 0)
      return FALSE;
  }
  _finally
  {
    //Clean up before leaving
    if (pImage)
     delete pImage;
    if (pRoi)
     delete pRoi;
  }
```

21

ChangeOverlay

Syntax

```
int ChangeOverlay(
 CcImage* CImage,
 CcRoiBase* CRoi,
 BYTE bValue);
```

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: A pointer to the image on which to perform the pixel change operation.

Name: CRoi

Description: A pointer to the ROI object that defines the area in which to perform the operation.

Name: bValue

Description: The fill color that is used to set all the pixels inside the specified ROI. The following values are supported:

- 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.

Description (cont.):

- OVERLAY_VIOLET –Sets the overlay to a transparent violet.
- OVERLAY_CYAN –Sets the overlay to a transparent cyan.
- OVERLAY_WHITE –Sets the overlay to a transparent white.

Return Values

< 0	Method failed.
0	Method was successful.

Example

```
//Grayscale image object
CcGrayImage256* pImage;
//Rectangular ROI object
CcRoiRect* pRoi;
//Change tool API object
CcChange API;

//Use structured exception
//handling
_try
{
  //Create a new image object.
  //Exit on failure.
  if ( !(pImage =
   new CcGrayImage256) )
    return FALSE;

  //Create a new rectangular ROI
  //object
  if ( !(pRoi = new CcRoiRect) )
    return FALSE;
```

Example (cont.)

```
    //Configure ROI coordinates
    RECT rcBounds = { 50, 150, 150,
     50};

    //Set coordinates to ROI object
    if (pRoi->SetRoiImageCord(
     &rcBounds) < 0)
      return FALSE;

    //Open a 640x480, 8-bit
    //grayscale bitmap
    if (pImage->OpenBMPFile(
     "MyImage.bmp")< 0)
      return FALSE;

    //Change the rectangle to red
    if (API.ChangeOverlay(pImage,
     pRoi, OVERLAY_RED) < 0)
      return FALSE;
  }
  _finally
  {
    //Clean up before leaving
    if (pImage)
     delete pImage;
    if (pRoi)
     delete pRoi;
  }
```

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.

```
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;
}
```

22

Using the Polar Unwrap Tool API

Overview of the Polar Unwrap Tool API

The API for the Polar Unwrap tool has one object only: the CcUnwrapper class. This class allows you to transform a section of an image from polar to rectangular coordinates.

The CcUnwrapper class uses a standard constructor and destructor and the class methods listed in Table 39.

Table 39: CcUnwrapper Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcUnwrappervoid);
	~CcUnwrapper(void);
CcUnwrapper Class Methods	int SetReferenceAngle(float fRefAngle);
	int GetReferenceAngle(float* pfRefAngle);
	int SetUnwrapAngle(float fUnwrapAngle);
	int GetUnwrapAngle(float* pfUnwrapAngle);
	int SetUnwrapDirection(BOOL bClockwise);
	int GetUnwrapDirection(BOOL bClockwise);
	int SetOutputScaleFactor(float fScalefactor);
	int GetOutputScaleFactor(float* pfScalefactor);
	int SetInputImage(CcImage* pImage);
	int SetInputRoi(CcRoiBase* pRoi);
	int SizeOutputImage(CcImage* pOutputImage);
	int Unwrap(CcImage* pOutputImage);

CcUnwrapper Methods

This section describes each method of the CcUnwrapper class in detail.

SetReferenceAngle

Syntax

```
int SetReferenceAngle(
   float fRefAngle);
```

Include File C_Unwrapper.h

Description Sets the reference angle for the polar unwrap operation.

Parameters

Name: fRefAngle

Description: The reference angle. Values range from 0° to 360°.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Error text buffer
TCHAR szText[500];

//Set the reference angle to
//45 degrees
if (Unwrapper.SetReferenceAngle(
   45) < 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
```

Example (cont.)

```
    //Report error
    ::MessageBox (NULL, szText,
     "Error", MB_OK);
}
```

GetReferenceAngle

Syntax

```
int GetReferenceAngle(
   float* pfRefAngle);
```

Include File C_Unwrapper.h

Description Returns the current reference angle for the polar unwrap operation.

Parameters

Name: pfRefAngle

Description: A pointer to a variable that contains the current reference angle.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Variable to receive the
//reference angle
float fRefAngle;
//Error text buffer
TCHAR szText[500];

//Get the current reference angle
if (Unwrapper.GetReferenceAngle(
   &fRefAngle) < 0)
```

Example (cont.)

```
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```

SetUnwrapAngle

Syntax

```
int SetUnwrapAngle(
   float fUnwrapAngle);
```


Include File C_Unwrapper.h

Description Sets the unwrap angle for the polar unwrap operation.

Parameters

Name: fUnwrapAngle

Description: The unwrap angle. Values range from 0° to 720°.

Return Values

< 0	Operation failed.
0	Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Error text buffer
TCHAR szText[500];
//Set the unwrap angle to 180
//degrees
if (Unwrapper.SetUnwrapAngle(180)<
   0)
```

Example (cont.)

```
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```

GetUnwrapAngle

Syntax

```
int GetUnwrapAngle(
   float* pfUnwrapAngle);
```

Include File C_Unwrapper.h

Description Returns the current unwrap angle for the polar unwrap operation.

Parameters

Name: pfUnwrapAngle

Description: A pointer to a variable that contains the current unwrap angle.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Variable to receive the
//unwrap angle
float fUnwrapAngle;
//Error text buffer
TCHAR szText[500];
```

Example (cont.)

```
//Get the current unwrap angle
if (Unwrapper.GetUnwrapAngle(
   &fUnwrapAngle)< 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```


SetUnwrapDirection

Syntax

```
int SetUnwrapDirection(
   BOOL bClockwise);
```

Include File C_Unwrapper.h

Description Sets the unwrap direction for the polar unwrap operation.

Parameters

Name: bClockwise

Description: The unwrap direction. If TRUE, all unwrap operations are performed in the clockwise (negative angular) direction. If FALSE, all unwrap operations are performed in the counterclockwise (positive angular) direction.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Error text buffer
TCHAR szText[500];
//Set the unwrap direction to
//clockwise
if (Unwrapper.SetUnwrapDirection
   (TRUE)< 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```

GetUnwrapDirection

Syntax

```
int GetUnwrapDirection(
   BOOL* pbClockwise);
```

Include File C_Unwrapper.h

Description Returns the current unwrap direction for the polar unwrap operation.

Parameters

Name: pbClockwise

Description: A pointer to a variable that contains the current unwrap direction.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Variable to receive the
//unwrap direction
BOOL fUnwrapDir;
//Error text buffer
TCHAR szText[500];
//Gets the current unwrap
//direction
if (Unwrapper.GetUnwrapDirection
   (&pUnwrapDir)< 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```


SetOutputScaleFactor

Syntax

```
int SetOutputScaleFactor(
   float fScaleFactor);
```

Include File C_Unwrapper.h

Description Sets the output scale factor for the polar unwrap operation.

Parameters

Name: fScaleFactor

Description: The scale factor to apply to the dimensions of the output image. Values range from 10% to 100%.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Error text buffer
TCHAR szText[500];
//Scale the output image by 50%
if (Unwrapper.SetOutputScaleFactor
   (50)< 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```

GetOutputScaleFactor

Syntax

```
int GetOutputScaleFactor(
   float* pfScaleFactor);
```

Include File C_Unwrapper.h

Description Returns the current scale factor for the polar unwrap operation.

Parameters

Name: pfScaleFactor

Description: A pointer to a variable that contains the current scale factor to apply to the dimensions of the output image.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Variable to receive the current
//scale factor
//Error text buffer
float fScaleFactor;
TCHAR szText[500];
//Get the current output scale
//factor
if (Unwrapper.GetOutputScaleFactor
   (&fScaleFactor)< 0)
{
  //Get error
  Unwrapper.GetErrorText(szText,
   500);
  //Report error
  ::MessageBox (NULL, szText,
   "Error", MB_OK);
}
```

22

SetInputImage

Syntax

```
int SetInputImage(
   CcImage* pImage);
```

Include File C_Unwrapper.h

Description Sets the input image for the polar unwrap operation.

Parameters

Name: pImage

Description: A pointer to a variable that contains the input image to use for the polar unwrap operation. Currently, this tool supports only 8-bit grayscale images.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Pointer to a grayscale image
//object

//Use structured exception
//handling
_try
{
  //Create a new image object.
  //Exit on failure.
  if ( !(pImage = new
   CcGrayImage256) )
   return FALSE;

  //Open a 640x480 8-bit grayscale
  //bitmap
  if (pImage->OpenBMPFile(
   "MyImage.bmp") < 0)
  //Set the input image.
  if (Unwrapper.SetInputImage(
   pImage) < 0)
   return FALSE;
}
```

Example (cont.)

```
_finally
{
  //Clean up before leaving
  if (pImage)
    delete pImage;
}
```

SetInputRoi

Syntax

```
int SetInputRoi(
   CcRoiBase* pRoi);
```

Include File C_Unwrapper.h

Description Sets the input ROI for the polar unwrap operation.

Parameters

Name: pRoi

Description: A pointer to a variable that contains the input ROI to use for the polar unwrap operation. Currently, this tool supports only elliptical ROIs.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Pointer to an 8-bit grayscale
//image object
CcGrayImage256* pImage;
//Pointer to an elliptical ROI
CcRoiEllipse* pRoi;
```

Example (cont.)

```
//Use structured exception
//handling
_try
{
  //Create a new image object.
  //Exit on failure.
  if ( !(pImage = new
   CcGrayImage256) )
   return FALSE;

  //Open a 640x480 8-bit grayscale
  //bitmap
  if (pImage->OpenBMPFile(
   "MyImage.bmp") < 0)

  //Set the input image.
  if (Unwrapper.SetInputImage(
   pImage) < 0)
   return FALSE;

  //Create a new elliptical
  //ROI object
  if ( !(pRoi = new CcRoiEllipse))
   return FALSE;

  //Configure ROI coordinates
  RECT rcBounds = {50, 150, 150,
   50};

  //Set coordinates to ROI objects
  if (pRoi->SetRoiImageCord(
   &rcBounds) < 0 )
   return FALSE;
```

Example (cont.)

```
    //Set the input ROI
    if (Unwrapper.SetInputRoi(
     pRoi) < 0)
     return FALSE;
  }
  _finally
  {
    //Clean up before leaving
    if (pImage)
      delete pImage;
    if (pRoi)
      delete pRoi;
  }
```


SizeOutputImage

Syntax

```
int SetOutputImage(
   CcImage* pOutputImage);
```

Include File C_Unwrapper.h

Description Sets the size of the supplied output image to the dimensions required to receive the polar unwrap image. The appropriate size for the output image is determined by the radius of the input ROI, the current unwrap angle, and the current output image scale factor.

Parameters

Name: pOutputImage

Description: A pointer to a variable that contains the output image.

Return Values

< 0 Operation failed.

0 Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Pointer to an input image object
CcGrayImage256* pInputImage;
//Pointer to an output image
//object
CcGrayImage256* pOutputImage;
//Pointer to an elliptical ROI
CcRoiEllipse* pInputRoi;
//Use structured exception
//handling
_try
{
  //Create an input image object.

  //Exit on failure.
  if ( !(pInputImage = new
   CcGrayImage256) )
   return FALSE;

  //Create an output image object.
  //Exit on failure.
  if ( !(pOutputImage = new
   CcGrayImage256) )
   return FALSE;

  //Open a 640x480 8-bit grayscale
  //bitmap
  if (pInputImage->OpenBMPFile(
   "MyImage.bmp") < 0)
   return FALSE;

  //Set the input image.
  if (Unwrapper.SetInputImage(
   pInputImage) < 0)
   return FALSE;
```

Example (cont.)

```
//Create a new elliptical
//ROI object
if ( !(pInputRoi = new
 CcRoiEllipse))
 return FALSE;

//Configure ROI coordinates
RECT rcBounds = {50, 150, 150,
 50};

//Set coordinates to ROI objects
if (pInputRoi->SetRoiImageCord(
 &rcBounds) < 0 )
 return FALSE;

//Set the input ROI
if (Unwrapper.SetInputRoi(
 pInputRoi) < 0)
 return FALSE;

//Unwrap 360 degrees (the entire
//ellipse
Unwrapper.SetUnwrapAngle(360);

//Scale the output image by 50%
Unwrapper.SetOutputScaleFactor(
 50);

//Size the output image to the
//proper dimensions
if (SizeOutputImage(
 pOutputImage) < 0
 return FALSE;
```

Example (cont.)

```
  //Unwrap the image under the
  //input ROI
  if
   (Unwrapper.Unwrap(pOutputImage)
   < 0)
   return FALSE;
}
_finally
{
  //Clean up before leaving
  if (pOutputImage)
    delete pOutputImage;
  if (pInputImage)
    delete pInputImage;
  if (pInputRoi)
    delete pInputRoi;
}
```

Unwrap

Syntax

```
int Unwrap(
   CcImage* pOutputImage);
```

Include File C_Unwrapper.h

Description Unwraps the area of the input image under the input ROI using the currently specified unwrap angle, unwrap direction, and scale factor.

Parameters

Name: pOutputImage

Description: A pointer to a variable to receive the unwrapped image.

Return Values

< 0	Operation failed.
0	Operation was successful.

Example

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;
//Pointer to an input image object
CcGrayImage256* pInputImage;
//Pointer to an output image
//object
CcGrayImage256* pOutputImage;
//Pointer to an elliptical ROI
CcRoiEllipse* pInputRoi;

//Use structured exception
//handling
_try
{
  //Create an input image object.
  //Exit on failure.

  if ( !(pInputImage = new
   CcGrayImage256) )
   return FALSE;

  //Create an output image object.
  //Exit on failure.
  if ( !(pOutputImage = new
   CcGrayImage256) )
   return FALSE;

  //Open a 640x480 8-bit grayscale
  //bitmap
  if (pInputImage->OpenBMPFile(
   "MyImage.bmp") < 0)
   return FALSE;
```

Example (cont.)

```
//Set the input image.
if (Unwrapper.SetInputImage(
 pInputImage) < 0)
 return FALSE;

//Create a new elliptical
//ROI object
if ( !(pInputRoi = new
 CcRoiEllipse))
 return FALSE;

//Configure ROI coordinates
RECT rcBounds = {50, 150, 150,
 50};

//Set coordinates to ROI objects
if (pInputRoi->SetRoiImageCord(
 &rcBounds) < 0 )
 return FALSE;

//Set the input ROI
if (Unwrapper.SetInputRoi(
 pInputRoi) < 0)
 return FALSE;

//Start unwrapping at the 90
//degree position
Unwrapper.SetRefAngle(90);

//Unwrap 360 degrees (the entire
//ellipse
Unwrapper.SetUnwrapAngle(360);

//Scale the output image by 50%
Unwrapper.SetOutputScaleFactor(
 50);
```

Example (cont.)

```
  //Unwrap in the clockwise
  //direction
  Unwrapper.SetUnwrapDirection(
   TRUE);

  //Size the output image to the
  //proper dimensions
  if (SizeOutputImage(
   pOutputImage) < 0)
   return FALSE;

  //Unwrap the image under the
  //input ROI
  if
   (Unwrapper.Unwrap(pOutputImage)
   < 0)
   return FALSE;
}

_finally
{
  //Clean up before leaving
  if (pOutputImage)
    delete pOutputImage;
  if (pInputImage)
    delete pInputImage;
  if (pInputRoi)
    delete pInputRoi;
}
```

Example Program Using the Polar UnwrapTool API

This example opens an input image and input ROI, and unwraps the image under the input ROI in the clockwise direction starting at a reference angle of 90°. The entire ellipse is unwrapped ellipse (360°), and the resulting output image is scaled by 50%.

```
//Polar unwrap tool API object
CcUnwrapper Unwrapper;

//Pointer to an input image object
CcGrayImage256* pInputImage;

//Pointer to an output image object
CcGrayImage256* pOutputImage;

//Pointer to an elliptical ROI
CcRoiEllipse* pInputRoi;

//Use structured exception handling

_try
{
  //Create an input image object.
  //Exit on failure.
  if ( !(pInputImage = new CcGrayImage256) )
      return FALSE;

  //Create an output image object.
  //Exit on failure.
  if ( !(pOutputImage = new CcGrayImage256) )
      return FALSE;
```

```
//Open a 640x480 8-bit grayscale bitmap
if (pInputImage->OpenBMPFile("MyImage.bmp") < 0)
    return FALSE;

//Set the input image.
if (Unwrapper.SetInputImage(pInputImage) < 0)
    return FALSE;

//Create a new elliptical ROI object
if ( !(pInputRoi = new CcRoiEllipse))
    return FALSE;

//Configure ROI coordinates
RECT rcBounds = {50, 150, 150, 50};

//Set coordinates to ROI objects
if (pInputRoi->SetRoiImageCord(&rcBounds) < 0 )
    return FALSE;

//Set the input ROI
if (Unwrapper.SetInputRoi(pInputRoi) < 0)
    return FALSE;

//Start unwrapping at the 90 degree position
Unwrapper.SetRefAngle(90);

//Unwrap 360 degrees (the entire ellipse)
Unwrapper.SetUnwrapAngle(360);

//Scale the output image by 50%
Unwrapper.SetOutputScaleFactor(50);

//Unwrap in the clockwise direction
Unwrapper.SetUnwrapDirection(TRUE);
```

```
    //Size the output image to the proper dimensions
    if (SizeOutputImage( pOutputImage) < 0)
        return FALSE;

    //Unwrap the image under the input ROI
    if (Unwrapper.Unwrap(pOutputImage) < 0)
        return FALSE;

  }

  _finally
  {
    //Clean up before leaving
    if (pOutputImage)
      delete pOutputImage;
    if (pInputImage)
      delete pInputImage;
    if (pInputRoi)
      delete pInputRoi;
  }
```

23

Using the ROI Shape Fitter Tool API

Overview of the ROI Shape Fitter Tool API

The API for the ROI Shape Fitter tool has one object only: the CcShapeFitter class. The CcShapeFitter class is designed to work within DT Vision Foundry environment. It is useful as an edge preprocessor for the Gauge tool. The class can be used for fitting a straight line into an edge, fitting a circle into an arc or an enclosed contour, or locating the center of gravity for a given object. The input and output of this tool is an ROI. For example, given an arc, you can find the center of a circle passing through the arc; given an approximately round object, you can find the center of gravity and then measure the minimum and maximum distances (radius) from the center of gravity to the outline of the object.

Any output from the CcShapeFitter class is passed in the STRT structure. The *CRoiOut* variable is used to pass the result of a shape fit operation. It contains a pointer to the resulting ROI. Note that currently, the *fFitError* variable of this structure is not supported.

```
struct stRTTag
{
    CcRoiBase *CRoiOut;  //Output fitted ROI
    float fFitError;    //Fit error; not implemented
};
typedef struct stRTTag STRT;
```

The CcShapeFitter class uses a standard constructor and destructor and the class methods listed in Table 40.

Table 40: CcShapeFitter Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcShapeFitter(void);
	~CcShapeFitter(void);

Table 40: CcShapeFitter Object Methods (cont.)

Method Type	Method Name
CcShapeFitter Class Methods	BOOL SetInputImage(CcImage* CImageIn);
	BOOL SetInputRoi(CcRoiBase* InputRoi);
	RoiToLineRoi();
	RoiToEllipseRoi();
	RoiToPointRoi();
	STRT * GetResults();
	CcList * GetMethodList();

CcShapeFitter Methods

This section describes each method of the CcShapeFitter class in detail.

SetInputImage

Syntax

```
BOOL SetInputImage(
   CcImage* CImageIn);
```

Include File C_ShapeFitter.h

Description Specifies the input image.

Parameters

Name: CImageIn

Description: Pointer to a CcImage object.

Return Values

TRUE The input image was set successfully.

FALSE The input image was not valid and was not set.

Comments This method passes the input image to the shape fitter class so that when circles are fit to ROIs, only those within the image are generated.

Example None

SetInputRoi

Syntax

```
BOOL SetInputRoi(
   CcRoiBase * InputRoi);
```

Include File C_ShapeFitter.h

Description Specifies the input ROI.

Parameters

Name: InputRoi

Description: Pointer to a DT Vision Foundry ROI class that specifies the input ROI. It can be a line, rectangle, ellipse, freehand line, poly freehand, or freehand ROI. Point and poly line ROIs are not 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;
CcShapeFitter   CShapeFitter;

//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 ROI
bStatus=CShapeFitter.SetInputRoi(
     (CcRoiBase *)&CRoiLine);
```

RoiToLineRoi

Syntax

```
RoiToLineRoi(
   );
```

Include File C_ShapeFitter.h

Description For freehand line input ROIs only, generates a line ROI representing the least-squares based line fit to the input points represented by the input ROI.

Parameters None

Return Values These values are returned by the **GetResults** method, described on page 839.

A pointer to a line ROI.	A supported ROI was provided.
NULL	An unsupported ROI was provided.

Example The following is a sample code fragment:

```
CcShapeFitter    CShapeFitter;
STRT            *stResults;
CcRoiBase       *CRoiIn, CRoiOut;
//Fill the CRoiIn with appropriate
//data
. . . .
//Set the input
CShapeFitter.SetInputRoi(CRoiIn);
// Invoke the fitting method
CShapeFitter.RoiToLineRoi();

stResults=CShapeFitter.GetResults(
   );
CRoiOut = stResults->CRoiOut;
```

Example (cont.)

```
if (CRoiOut == NULL)
{
    Error("Failed to generate a
     new ROI!");
     return;
}
```

RoiToEllipseRoi

Syntax

```
RoiToEllipseRoi(
   );
```

Include File C_ShapeFitter.h

Description For freehand line and freehand input ROIs only, generates an ellipse ROI representing the least-squares-based circle fit to the input points that are represented by the input ROI.

Parameters None

Return Values These values are returned by the **GetResults** method, described on page 839.

A pointer to an ellipse ROI.	A supported ROI was provided.
NULL	An unsupported ROI was provided.

Example The following is a sample code fragment:

```
CcShapeFitter    CShapeFitter;
STRT            *stResults;
CcRoiBase       *CRoiIn, CRoiOut;
//Fill the CRoiIn with appropriate
//data
. . . .
//Set the input
CShapeFitter.SetInputRoi(CRoiIn);
```

23

Example (cont.)

```
// Invoke the fitting method
CShapeFitter.RoiToEllipseRoi();

stResults=CShapeFitter.GetResults(
   );
CRoiOut = stResults->CRoiOut;
if (CRoiOut == NULL)
{
Error("Failed to generate a new
            ROI!");
     return;
}
```

RoiToPointRoi

Syntax

```
RoiToPointRoi(
   );
```

Include File C_ShapeFitter.h

Description For line, rectangle, ellipse, poly freehand, and freehand input ROIs only, generates a point ROI that represents the center of gravity (for enclosed input ROIs) or the middle point (for line input ROIs).

Parameters None

Return Values These values are returned by the **GetResults** method, described on page 839.

A pointer to a point ROI.	A supported ROI was provided.
NULL	An unsupported ROI was provided.

Example The following is a sample code fragment:

```
CcShapeFitter    CShapeFitter;
STRT            *stResults;
CcRoiBase       *CRoiIn, CRoiOut;

//Fill the CRoiIn with appropriate
//data
. . . .
//Set the input
CShapeFitter.SetInputRoi(CRoiIn);

//Invoke the fitting method
CShapeFitter.RoiToPointRoi();
stResults=CShapeFitter.GetResults(
   );
CRoiOut = stResults->CRoiOut;
if (CRoiOut == NULL)
{
Error("Failed to generate a new
         ROI!");
     return;
}
```


GetResults

Syntax

```
STRT * GetResults(
   );
```

Include File C_ShapeFitter.h

Description Returns a pointer to the results structure. It can be invoked after executing one of the fitting methods.

Parameters None

Return Values

A pointer to the results structure containing the output ROI.	Successful.
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcShapeFitter     CShapeFitter;
STRT             *stResults;
CcRoiBase        *CRoiIn, CRoiOut;

//Fill the CRoiIn with appropriate
//data
. . . .
//Set the input
CShapeFitter.SetInputRoi(CRoiIn);

// Invoke the fitting method
CShapeFitter.RoiToPointRoi();

stResults =
   CShapeFitter.GetResults();
CRoiOut = stResults->CRoiOut;
if (CRoiOut == NULL)
{
     Error("Failed to generate a
   new
       ROI!");
     return;
}
```

GetMethodList

Syntax

```
CcList * GetMethodList(
   );
```

Include File C_ShapeFitter.h

Description Returns a pointer to the list of fitter method pointers. This list provides a way to associate text names of the fitter methods with pointers to these methods so that you can invoke the fitter methods based on their text names. The text names are defined at the top of the C_ShapeFitter.h header file.

Parameters None

Return Values These values are returned by the **GetResults** method, described on page 839.

A pointer to the list containing the ROI fitting methods.	Successful.
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcShapeFitter    CShapeFitter;
STRT            *stResults;
CcRoiBase       *CRoiIn, CRoiOut;
CcList          *MethodList;
CcFitterMethod  *CFitterMethod;
//Fill the CRoiIn with appropriate
//data
. . . .
//Set the input
CShapeFitter.SetInputRoi(CRoiIn);
```

Example (cont.)

```
// Get the list of fitter methods
MethodList=CShapeFitter.
   GetMethodList ();
// Get the method pointer from the
//list based on the name
CFitterMethod=(CcFitterMethod *)
   TheList->GetViaName(“Circle”);
if (CFitterMethod == NULL)
{
    Error("Invalid method name.");
   return;
}

// Invoke the fitting method
(CShapeFitter.*CFitterMethod->
     ShapeFitterMethod)();

stResults =
   CShapeFitter.GetResults();
CRoiOut = stResults->CRoiOut;
if (CRoiOut == NULL)
{
Error("Failed to generate a new
   ROI!");
return;
}
```

24

Using the Search Tool API

Overview of the Search Tool API

The CcSearch Geometrically Enhanced Grayscale Correlation class contains all the methods required to configure and execute a template matching operation.

The capabilities provided by these methods include the ability to set a feature image (an image containing the feature that will be searched for), the ability to specify both a search level and a correlation type (such as standard grayscale correlation or geometrically enhanced grayscale correlation), the ability to search an inspection image for the current feature image, the ability to specify the maximum number of matches that should be found in the inspection image, and the ability to retrieve match metrics (such as x- and y-position and match score) for each match resulting from the most recently performed template matching operation.

For proper operation, you must specify the following parameters, in the order shown, before calling the **Search** method:

1. Inspection image using **SetInspectionImage**, described on page 849.
2. Inspection ROI using **SetInspectionROI**, described on page 850.
3. Feature image using **SetFeatureImage**, described on page 848.
4. Mask image (if you are using a feature-type search) using **SetMaskImage**, described on page 851, or **GuessMaskImage**, described on page 877.
5. Search type using **SetSearchType**, described on page 857.
6. Search level using **SetSearchLevel**, described on page 855.
7. Any other search parameters, such as the subpixel flag, the maximum number of matches, and so on.

SearchTypeEnum Enumeration

The SearchTypeEnum enumeration defines the valid search types (correlation methods) that you can use to perform a template matching operation.

```
SearchTypeEnum
typedef enum tagSearchTypeEnum
{
   GrayScaleNormDirect     = 1,
   GrayScaleNormFft        = 3,
   GrayScaleNormDirectEx   = 5,
   GrayScaleNormFftEx      = 7,
   GrayScaleNormDirectMMX = 9,
   Feature              = 777,
} SearchTypeEnum;
```

The search component supports normalized grayscale correlation, which can be further categorized according to its invariance to changes in lighting conditions. The light invariant correlation types are identified with an Ex postfix.

MatchRecord Type

MatchRecord is the primary structure through which match information is retrieved from the search component:

```
MatchRecord

    typedef struct tagMatchRecord
{
    float fXPos;
    float fYPos;
    float fMetric;
    BOOL bValid;

} MatchRecord;
```

The *fXPos* and *fYPos* members identify the location of the upper-left corner of the feature match in the inspection image. *fXPos* and *fYPos* are given in cartesian coordinates, where (0,0) corresponds to the lower-left corner of the inspection image. The *fMetric* member indicates the strength of a feature match and can range from 0.0 to 1.0. The *bValid* member indicates whether or not the metric contained in *fMetric* is above (TRUE) or below (FALSE) the match score threshold set by **SetScoreThresh**.

Class Method Summary

The CcSearch class uses a standard constructor and destructor and the class methods listed in Table 41.

Table 41: CcSearch Object Methods

Method Type	Method Name
Constructor & Destructor Methods	CcSearch(void);
	~CcSearch(void);
CcSearch Class Methods	int SetFeatureImage(CcImage *pFeatureImage);
	int SetInspectionImage(CcImage *pInspectionImage);
	int SetInspectionRoi(CcRoiBase *pInspectionRoi);
	int SetMaskImage(CcImage *pMaskImage);
	int SetMaxNumMatches(int iMaxNumMatches);
	int SetNumPoints(int iNumPoints);
	int SetSearchLevel(int iSearchLevel);
	int SetSearchType(SearchTypeEnum SearchType);
	int SetScoreThresh(float fScoreThresh);
	int SetSubpixelFlag(bool bSubpixel);

Table 41: CcSearch Object Methods (cont.)

Method Type	Method Name
CcSearch Class Methods (cont.)	bool SaveCatalog(char *cName);
	bool LoadCatalog(char *cName);
	int Search(void);
	CcImage* GetFeatureImage(void);
	int GetMaxNumMatches();
	int GetValidNumMatches();
	CcImage* GetMaskImage(void);
	int GetMaxMatch(MatchRecord * pMatchRecord);
	int GetMinMatch(MatchRecord * pMatchRecord);
	int GetMatch(int iMatchIndex, MatchRecord * pMatchRecord);
	int GetSearchTime();
	int GetSearchLevel(void);
	int GetSearchType();
	bool GetSubpixelFlag(void);
	float GetScoreThresh(void);
	CcRoiBase* GuessMaskImage(void);

24

CcSearch Methods

This section describes each method of the CcSearch class in detail.

SetFeatureImage

Syntax

```
int SetFeatureImage(
   CcImage *pFeatureImage);
```

Include File C_Search.h

Description Specifies the current feature image (the image containing the feature to search for in the inspection image).

Parameters

Name: pFeatureImage

Description: A pointer to an object of type CcImage, which contains the feature image. This image should be of type IMAGE_TYPE_08BIT_GS (8-bit grayscale), IMAGE_TYPE_24BIT_RGB (24-bit RGB color), or IMAGE_TYPE_24BIT_HSL (24-bit HSL color).

Notes You must set the inspection image using **SetInspectionImage** and the inspection ROI using **SetInspectionROI** before calling this method. Refer to page 849 and page 850 for more information.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
//Search object instance.
CcImage *pFeatureImage;
// Pointer to grayscale image
// object.
pFeatureImage = new
   CcGrayImage256;
Result = SearchObj.SetFeatureImage
     (&pFeatureImage);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetInspectionImage

Syntax

```
int SetInspectionImage(
   CcImage *pInspectionImage);
```

Include File C_Search.h

Description Specifies the inspection image in which to search.

Parameters

Name: pInspectionImage

Description: A pointer to an object of type CcImage, which contains the feature image. This image should be of type IMAGE_TYPE_08BIT_GS (8-bit grayscale), IMAGE_TYPE_24BIT_RGB (24-bit RGB color), or IMAGE_TYPE_24BIT_HSL (24-bit HSL color).

Notes Ensure that you set the inspection image before setting any other search parameters.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
//Search object instance.
CcImage *pInspectionImage;
// Pointer to grayscale image
// object.
Result = SearchObj.
   SetInspectionImage (
   pInspectionImage);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetInspectionROI

Syntax

```
int SetInspectionROI(
   CcRoiBase *pInspectionRoi);
```

Include File C_Search.h

Description Specifies the inspection ROI in which to search.

Parameters

Name: pInspectionRoi

Description: A pointer to an object of type CcRoiBase, which contains the inspection ROI.

Notes Ensure that you specify the inspection image using **SetInspectionImage** (described on page 849) before calling this method.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
//Search object instance.
CcRoiBase *pInspectionRoi;
Result = SearchObj.
   SetInspectionRoi (
   pInspectionRoi);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```


SetMaskImage

Syntax

```
int SetMaskImage(
   CcImage *pMaskImage);
```

Include File C_Search.h

Description Specifies the mask image.

Parameters

Name: pMaskImage

Description: A pointer to an object of type CcImage, which contains the mask image. This image should be of type IMAGE_TYPE_BINARY (binary image), IMAGE_TYPE_08BIT_GS (8-bit grayscale), IMAGE_TYPE_16BIT_GS (16-bit grayscale), or IMAGE_TYPE_32BIT_GS (32-bit grayscale).

Notes This method is used only by the feature-type search. Black pixels in the mask image designate the pixel locations in the feature image that are used during the search operation.

Ensure that you specify the following parameters before calling this method:

1. Inspection image using **SetInspectionImage**, described on page 849,
2. Inspection ROI using **SetInspectionROI**, described on page 850, and
3. Feature image using **SetFeatureImage, described** on page 848.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
//Search object instance.
CcImage *pMaskImage;
Result = SearchObj.SetMaskImage(
   pMaskImage);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetMaxNumMatches

Syntax

```
int SetMaxNumMatches(
   int iMaxNumMatches);
```

Include File C_Search.h

Description Specifies the maximum number of matches that you want the software to look for during the search.

Parameters

Name: iMaxNumMatches

Description: The maximum number of feature matches that you want the software to locate during the search. The value must be a positive integer.

Notes You must set the search type using **SetSearchType** before calling this method. Refer to page 857 for more information on setting the search type.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
int       Result, iMaxNumMatches;

// Find 10 matches.

iMaxNumMatches = 10;
Result = SearchObj.
   SetMaxNumMatches(
   iMaxNumMatches);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetNumPoints

Syntax

```
int SetNumPoints(
   int iNumPoints);
```

Include File C_Search.h

Description Specifies the number of points to use in subpixel analysis.

Parameters

Name: iNumPoints

Description: The maximum number of points to use in subpixel analysis. Currently, this number must be 9. Future versions of this tool may support additional values.

Notes None

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
int       Result, iNumPoints;

// Set the number of points to 9.

iNumPoints = 9;
Result = SearchObj.SetNumPoints(
   iNumPoints);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetSearchLevel

Syntax

```
int SetSearchLevel(
   int iSearchLevel);
```

Include File C_Search.h

24

Description Specifies the search level that you want the software to apply to both the feature and inspection images before the template matching operation is performed.

Parameters

Name: iSearchLevel

Description: The search level (or down-sampling factor) by which the feature and inspection images should be down-sampled before the template matching operation is performed. The value can range from 0 to 4, where 0 is the most coarse and 4 is the finest search level.

Down-sampling decreases the resolution of the feature and inspection images and, therefore, increases the execution speed of the core correlation algorithms that are used in the search component. However, this increase in execution speed is realized at the expense of accuracy since the down-sampling process reduces the amount of information in the feature and inspection images.

Notes Ensure that you set the following parameters before calling this method:

1. Inspection image using **SetInspectionImage**, described on page 849.
2. Inspection ROI using **SetInspectionROI**, described on page 850.
3. Feature image using **SetFeatureImage,** described on page 848.

Notes (cont.)

4. Mask image (if you are using a feature-type search) using **SetMaskImage**, described on page 851, or **GuessMaskImage**, described on page 877.
5. Search type using **SetSearchType**, described on page 857.

Return Values

0	Successful.
< 0	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int        Result;
int        iSearchLevel;
// Use a search level of 3 in
// the search.
iSearchLevel = 3;
Result = SearchObj.SetSearchLevel(
     iSearchLevel);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetSearchType

Syntax

```
int SetSearchType(
   SearchTypeEnum SearchType);
```

Include File C_Search.h

Description Specifies the correlation type that you want to use in the template matching operation.

Parameters

Name: SearchType

Description: The correlation technique to use in the template matching operation. Valid values for this parameter are defined by the SearchTypeEnum enumeration, described on page 845.

Notes Ensure that you specify the following parameters before calling this method:

1. Inspection image using **SetInspectionImage**, described on page 849.
2. Inspection ROI using **SetInspectionROI**, described on page 850.
3. Feature image using **SetFeatureImage**, described on page 848.
4. Mask image (if you are using a feature-type search) using **SetMaskImage**, described on page 851, or **GuessMaskImage**, described on page 877.

Return Values

0	Successful.
< 0	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int Result;

// Use light invariant normalized
// grayscale correlation.

Result = SearchObj.SetSearchType(
      NormGrayScaleEx);
if (Result < 0)
{
// Operation failed – handle error
}
```

SetScoreThresh

Syntax

```
int SetScoreThresh(
   float fScoreThresh);
```

Include File C_Search.h

Description Specifies the lower bound of the range of match scores that the software should consider as valid.

Parameters

Name: fScoreThresh

Description: The lower bound of the range of match scores that the software should consider as valid. A match with a score that falls below *fScoreThresh* is not considered a match.

Description (cont): All match scores in a given result set (those resulting from a search operation) are normalized to values between 0.0 and 1.0, where the best match has the value closest to 1.0. For example, if *fScoreThresh* = 0.5, only the matches with scores between 0.5 and 1.0 are reported as valid.

When retrieved from the result set by calls to **GetMatch**, valid matches have the *bValid* member of the MatchRecord structure set to TRUE.

Notes You must set the search level using **SetSearchLevel** before calling this method. Refer to page 855 for more information on setting the search level.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
float     fScoreThresh;
int       Result;
// Report only matches with a
// score of 0.5 or greater.
fScoreThresh = 0.5;
Result=SearchObj.SetScoreThresh(
  fScoreThresh);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SetSubpixelFlag

Syntax

```
int SetSubpixelFlag(
   bool bSubpixel);
```

Include File C_Search.h

Description Specifies whether or not to use subpixel analysis.

Parameters

Name: bSubpixel

Description: Determines whether subpixel analysis is used. If TRUE, subpixel analysis is used. If FALSE, subpixel analysis is not used.

Notes You must set the search level using **SetSearchLevel** before calling this method. Refer to page 855 for more information on setting the search level.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int Result;

Result=SearchObj.SetSubpixelFlag(
   TRUE);
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

SaveCatalog

Syntax

```
bool SaveCatalog(
   char *cName);
```

Include File C_Search.h

Description Saves the specified catalog.

Parameters

Name: cName

Description: A pointer to the file name of the catalog to save.

Notes The following parameters are saved when this method is called:

- Feature image,
- Mask image,
- Search type,
- Match score threshold,
- Search level,
- Maximum number of matches,
- Subpixel analysis flag,
- Color plane access for color images.

Ensure that you explicitly set these parameters; otherwise, you may save their default values and not the values you intended to save.

Return Values

TRUE Successful.

FALSE Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
char myfile[256];
bool Result;

Result=SearchObj.SaveCatalog(
   myfile);
if (Result = FALSE)
{
    // Operation failed.
    // Handle error.
}
```

LoadCatalog

Syntax

```
bool LoadCatalog(
   char *cName);
```

Include File C_Search.h

Description Loads the specified catalog.

Parameters

Name: cName

Description: A pointer to the file name of the catalog to load.

Notes The following parameters are loaded when this method is called:

- Feature image,
- Mask image,
- Search type,
- Match score threshold,

Notes (cont.)

- Search level,
- Maximum number of matches,
- Subpixel analysis flag,
- Color plane access for color images.

Return Values

TRUE Successful.

FALSE Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
char myfile[256];
bool Result;

Result=SearchObj.LoadCatalog(
   myfile);
if (Result = FALSE)
{
    // Operation failed.
    // Handle error.
}
```

Search

Syntax `int Search(void);`

Include File C_Search.h

Description Searches the specified inspection image and ROI for the feature image specified by **SetFeatureImage**.

Parameters None.

Return Values

0 Successful.

< 0 Unsuccessful.

Notes Invoke this method after all of the following parameters have been specified (in the order specified):

1. Inspection image using **SetInspectionImage**, described on page 849.
2. Inspection ROI using **SetInspectionROI**, described on page 850.
3. Feature image using **SetFeatureImage**, described on page 848.
4. Mask image (if you are using a feature-type search) using **SetMaskImage**, described on page 851, or **GuessMaskImage**, described on page 877.
5. Search type using **SetSearchType**, described on page 857.
6. Search level using **SetSearchLevel**, described on page 855.
7. Any other search parameters, such as the subpixel flag, the maximum number of matches, and so on.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.

// Load image data ...
```

Example (cont.)

```
Result = SearchObj.Search();
if (Result < 0)
{
    // Operation failed.
    // Handle error.
}
```

GetFeatureImage

Syntax

```
CcImage* GetFeatureImage(void);
```

Include File C_Search.h

Description Returns the currently specified feature image, which was set using **SetFeatureImage** or by using **LoadCatalog**.

Parameters None

Return Values

A pointer to the currently specified feature image (of type CcImage).	This image must be of type IMAGE_TYPE_08BIT_GS (8-bit grayscale), IMAGE_TYPE_24BIT_RGB (24-bit RGB color), or IMAGE_TYPE_24BIT_HSL (24-bit HSL color).
NULL	Unsucessful.

Notes You can use this method to retrieve the feature/catalog image after loading the catalog from disk.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
CcImage *Result;
//Get the currently specified
//feature image.
```

Example (cont.)

```
Result = SearchObj.
   GetFeatureImage();
```

GetMaxNumMatches

Syntax

```
int GetMaxNumMatches(
   void);
```

Include File C_Search.h

Description Returns the maximum number of matches that you want the software to locate during the search (this value is set by **SetMaxNumMatches**).

Parameters None

Return Values

The maximum number of matches that you want the software to locate during the search.	Successful.
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int iNumMatches
// Get the number of matches found
// during the last search.

iNumMatches = SearchObj.
   GetMaxNumMatches();
```

GetValidNumMatches

Syntax

```
int GetValidNumMatches(
   );
```

Include File C_Search.h

Description Returns the number of valid matches in the current search result set (the number of matches that have a score that is greater than the current match score threshold).

Parameters None

Return Values

The number of matches that have a score that is greater than the current match score threshold.	Successful.
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int iValMatches;

// Get the number of matches found
//during the last search.

iValMatches = SearchObj.
   GetValidNumMatches();
```

GetMaskImage

Syntax

```
CcImage* GetMaskImage(
   void);
```

Include File C_Search.h

Description Returns the currently specified mask image, which was set using **SetMaskImage** or by using **LoadCatalog**.

Parameters None

Return Values

A pointer to the currently specified mask image (of type CcImage).	This image must be of type IMAGE_TYPE_BINARY (binary image), IMAGE_TYPE_08BIT_GS (8-bit grayscale), IMAGE_TYPE_16BIT_GS (16-bit grayscale), or IMAGE_TYPE_32BIT_GS (32-bit grayscale).
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
CcImage *Result;
// Retrieve the currently
// specified mask image.
Result = SearchObj.
   GetMaskImage();
```


GetMaxMatch

Syntax

```
int GetMaxMatch(
   MatchRecord *pMatchRecord);
```

Include File C_Search.h

Description

Returns the match with the highest match score in the match set generated during the last search operation (when **Search** was last called).

Parameters

Name: pMatchRecord

Description: The match metrics, such as x- and y-position, for the match with the maximum match score in the current result set.

Return Values

0 Successful.

< 0 Unsuccessful.

Example

The following is a sample code fragment:

```
CcSearch     SearchObj;
// Search object instance.
MatchRecord MatchRecord;
int Result;
// Retrieve the match with the
// maximum match score.
Result = SearchObj.GetMaxMatch(
  &MatchRecord);
if (Result > 0)
{
    // Process the data contained
    // in MatchRecord.
}
```

GetMinMatch

Syntax

```
int GetMinMatch(
   MatchRecord *pMatchRecord);
```

Include File C_Search.h

Description Returns the match, in the match set generated during the last search operation (when **Search** was last called), with the lowest match score that is still above the match threshold specified in **SetScoreThresh**.

Parameters

Name: pMatchRecord

Description: The match metrics, such as x- and y-position, for the match with the lowest match score that is still above the match threshold specified in **SetScoreThresh**.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch     SearchObj;
// Search object instance.
MatchRecord MatchRecord;
int Result;
// Retrieve the match with the
// lowest valid match score.
Result = SearchObj.GetMinMatch(
    &MatchRecord);
if (Result > 0)
{ // Process the data contained
    // in MatchRecord.}
```

24

GetMatch

Syntax

```
int GetMatch(
   int iMatchIndex,
   MatchRecord *pMatchRecord);
```

Include File C_Search.h

Description Returns a specific match record from the current result set.

Parameters

Name: iMatchIndex

Description: The match record in the current result set that you want to return. The value can range from 0 to *n*, where *n* is the value returned by **GetMaxNumMatches** (the total number of matches in the current result set).

Name: pMatchRecord

Description: The match metrics, such as x- and y-position, for the match identified by *nMatchIndex*.

Return Values

0 Successful.

< 0 Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch     SearchObj;
// Search object instance.
MatchRecord MatchRecord;
int Result, iMatchIndex;
iMatchIndex = 2;

// Retrieve the third match
// result.
```

Example (cont.)

```
Result = SearchObj.GetMatch(
 iMatchIndex, &MatchRecord);
if (Result > 0)
{
    // Process the data contained
    // in MatchRecord.
}
```

GetSearchTime

Syntax

```
int GetSearchTime(
   );
```

Include File C_Search.h

Description Returns the execution time, in milliseconds, of the last template matching operation.

Parameters None

Return Values

The execution time, in milliseconds, of the last template matching operation.	Successful.
NULL	Unsuccessful.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int Result;
// Get the execution time of the
// last search.
Result = SearchObj.
   GetSearchTime();
```

GetSearchLevel

Syntax

```
int GetSearchLevel(
   void);
```

Include File C_Search.h

Description Returns the current search level, which was set using **SetSearchLevel**.

Parameters None

Return Values

The current search level. Values can range from 0 to 4, where 0 is the most coarse and 4 is the finest search level.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int Result;

// Get the current search level
Result = SearchObj.
   GetSearchLevel();
```

GetSearchType

Syntax

```
int GetSearchType(
   );
```

Include File C_Search.h

Description Returns the current search type, which was set using **SetSearchType**.

Parameters None

Return Values

The current search type.	Valid values for this parameter are defined by the SearchTypeEnum enumeration, described on page 845.

Notes None

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
int Result;

// Get the current search type
Result = SearchObj.
   GetSearchType();
```

GetSubpixelFlag

Syntax

```
bool GetSubpixelFlag(
   void);
```


Include File C_Search.h

Description Returns whether or not subpixel analysis was specified using **SetSubpixelFlag**.

Parameters None

Return Values

TRUE	Subpixel analysis was specified.
FALSE	Subpixel analysis was not specified.

Notes None

Example

The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
bool Result;

// Determine whether subpixel
// analysis was specified.
Result = SearchObj.
   GetSubpixelFlag();
```

GetScoreThresh

Syntax

```
float GetScoreThresh(
   void);
```

Include File

C_Search.h

Description

Returns the current score threshold, which was set using **SetScoreThresh**.

Parameters

None

Return Values

The current score threshold value.	The lower bound of the range of match scores that the software should consider as valid. A match with a score that falls below *fScoreThresh* is not considered a match.

Notes

All match scores in a given result set (those resulting from a search operation) are normalized to values between 0.0 and 1.0, where the best match has the value closest to 1.0. For example, if *fScoreThresh* = 0.5, only the matches with scores between 0.5 and 1.0 are reported as valid.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
float Result;

// Get the score threshold
// that was set.
Result = SearchObj.
   GetScoreThresh();
```

GuessMaskImage(void)

Syntax

```
CcRoiBase* GuessMaskImage(
   void);
```

Include File C_Search.h

Description Automatically guesses and sets the mask image; this method can be used instead of **SetMaskImage**. It returns the ROI of a guessed search object. This ROI encloses the region where the object you are searching for was guessed to exist.

Parameters None

Return Values

Pointer to an ROI of type CcRoiBase that encloses the guessed object.

Notes You must set the feature image using **SetFeatureImage**, described on page 848, before calling this method.

You are responsible for freeing the returned ROI.

Example The following is a sample code fragment:

```
CcSearch SearchObj;
// Search object instance.
CcRoiBase* Result;

// Returns the mask image ROI
Result = SearchObj.
   GuessMaskImage();
```

25

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 42.

Table 42: 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 InitializeComPortEx(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);

Table 42: CcSerialIO Object Methods (cont.)

Method Type	Method Name
CcSerialIO Class Methods (cont).	int ReadComPort(char* cPrefix,CcNumber* CNumber, char* cSuffix);
	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. If there is an existing COM port handle, this method frees the handle, then creates a new handle to the COM port; therefore, any data in the existing COM port is lost.

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.

See also **InitializeComPortEx**, described on page 887.

Return Values

-1	Unsuccessful.
The value of the active COM port.	Successful.

InitializeComPortEx

Syntax

```
int InitializeComPortEx(
  BOOL bAsync = FALSE,
  BOOL bPurge = FALSE);
```

Include File

C_Serial.h

Description

Initializes the active COM port using the existing COM port handle; therefore, any data in the existing COM port is saved.

Parameters

Name: bAsync

Description: Set this parameter to TRUE if you want to perform asynchronous reads/writes on the existing COM port. Set this parameter to FALSE to perform synchronous reads/writes on the existing COM port.

Name: bPurge

Description: Set this parameter to TRUE to clear/purge the associated buffer for the existing 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.

See also **InitializeComPort**, described on page 886.

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: DT Vision Foundry String object that is receiving the data that is entering the COM port.

Name: CNumber

Description: DT Vision Foundry 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 DT Vision Foundry 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.

25

Notes (cont.)

- *stComConfig* –Is a standard Windows COMMCONFIG structure. For more information, refer to the Windows SDK documentation.
- *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 DT Vision Foundry String object that contains the data to send out the COM port.

Name: CNumber

Description: Pointer to a DT Vision Foundry 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 DT Vision Foundry 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(1) != TRUE)
return(-1);

   //Initialize COM Port
   CCom.InitializeComPort(FALSE, FALSE);

   //Write Text out Port
   CCom.WriteComPort (“Prefix”, “Data”, “Suffix”);

   //Return OK
   return(0);
}
```

26

Using the Sound Tool API

Overview of the Sound Tool API

The API for the Sound tool uses DT Vision Foundry API objects. CcWAV uses a standard constructor and destructor and the class methods listed in Table 43.

Table 43: 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 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 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);
}
```

27

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 44.

Table 44: 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.

28

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 45.

Table 45: 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 ThresholdHSL(
  Cc24BitHSLImage* 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 *iRed*, *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;
}
```

29

Creating DT Vision Foundry Tools

Introduction

This chapter describes the operation of the DT Vision Foundry 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 *DT Vision Foundry User's Manual*.

What is a Tool?

The DT Vision Foundry 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 DT Vision Foundry package includes an example change tool (located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 DT Vision Foundry 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.

DT Vision Foundry Messages

All tools communicate with the DT Vision Foundry main application using a standard set of DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry messages in detail.

Note: All messages are defined in the DT_MGS.H header file, which is located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Include, by default. Structures used by these messages are defined in the DT_STR.H header file also located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Include, by default.

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 DT Vision Foundry 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 46.

Table 46: 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*

Table 46: Request Messages (cont.)

Request Message	Returned Information	Return Type
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
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*.

29

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 starting on page 11.

If the viewport is not displaying an image, this message returns NULL.

Return Values

NULL	Unsuccessful.
CImage –a pointer to a DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry 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.

The ROI returned is derived from an DT Vision Foundry 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 starting on page 11.

Return Values

NULL	Unsuccessful.
A pointer to a DT Vision Foundry 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 DT Vision Foundry 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.

Notes (cont.)

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.

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 starting on page 11.

Return Values

NULL	Unsuccessful.
A pointer to a DT Vision Foundry 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 DT Vision Foundry 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.

Notes (cont.)

- ROI_PLINE –Poly Line.
- ROI_FLINE –Freehand Line.
- ROI_RECT –Rectangle.
- ROI_ELLIPSE –Ellipse.
- ROI_FREEHAND –Freehand.
- ROI_PFREEHAND –Poly Freehand.

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.

29

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 DT Vision Foundry 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 seeChapter 2 starting 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 CcDTTool::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++)
```

Example (cont.)

```
{
::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 starting 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 (located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 CcDTTool::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 DT Vision Foundry notification messages start with the prefix HLN_.

The notification messages are briefly described in Table 47.

Table 47: 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 47: 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 47: 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 CcDTTool::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 CcDTTool::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).

29

HLN_DELETING_IMAGE_OBJECT

Syntax

```
//******** H L  N O T I F Y *****//
LRESULT CcDTTool::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).

29

HLN_ROI_TYPE_CHANGE

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcDTTool::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 CcDTTool::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.

29

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 CcDTTool::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 CcDTTool::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.

29

HLN_ROI_ACTIVATED

Syntax

```
//***** H L  N O T I F Y ********//
LRESULT CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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*.

29

HLN_DELETED_CALIBRATION_OBJECT

Syntax

```
//**** H L  N O T I F Y *********//
LRESULT CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 starting 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 DT Vision Foundry command messages start with the prefix: HLC_.

The command messages are briefly described in Table 48.

Table 48: 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 48: 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.
HLC_ROI_DELETE_ALL	Deletes all the ROIs in the given viewport.

Table 48: Command Messages (cont.)

Command Message	Description of Message
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_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 48: 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.

DT Vision Foundry 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.

Notes (cont.) 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*.

DT Vision Foundry 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

29

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 DT Vision Foundry 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.

29

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, and 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 (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

DT Vision Foundry 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 color table is used only for binary, 8-bit, 32-bit, and floating-point grayscale images.

DT Vision Foundry 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 DT Vision Foundry 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.

Notes (cont.) You can use the DT Vision Foundry 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.

Each viewport in the DT Vision Foundry 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.

Notes (cont.)

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 DT Vision Foundry 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.

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.

Notes (cont.)

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 DT Vision Foundry 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_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 DT Vision Foundry 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.
- 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 CcDTTool::OnPositionViewport(
   )
{
 RECT stPos;

stPos.top = 10;
stPos.bottom = 310;
```

Example (cont.)

```
 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 DT Vision Foundry 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 CcDTTool::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 DT Vision Foundry 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 DT Vision Foundry. 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 starting 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 (located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 CcDTTool::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 DT Vision Foundry point and click script messages start with the prefix: HLS_.

The point and click script messages are briefly described in Table 49.

Table 49: Point and Click Script Messages

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 49: 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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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.

This message along with HLS_SUPPLY_SCRIPT_STRUCT_SIZE provides a mechanism for upgrading script structures.

Notes (cont.)

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 CcDTTool::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.

29

HLS_DELETING_SCRIPT_STRUCT

Syntax

```
//******* H L  SCRIPT ******//
LRESULT CcDTTool::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 CcDTTool::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 CcDTTool::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 CcDTTool::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 the example change tool that is included in the DT Vision Foundry package (located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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.

This example consists of the following main tasks, which are described in the following subsections:

- Create a base tool.
- Register the tool with DT Vision Foundry.
- 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 DT Vision Foundry package (located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 DT Vision Foundry tools. All DT Vision Foundry 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 DT Vision Foundry 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 DT Vision Foundry 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 2000 or Windows XP and load the example change tool's project workspace file from within the Microsoft Visual Studio. The name of this file is DT_Change.dsw; it is located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, by default).

   Note: If you need more help, refer to your Visual C/C++ documentation.

   ---

2. If you did not install the DT Vision Foundry application using the install program, change the include path for all DT Vision Foundry include files to C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Include.

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 DT Vision Foundry 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 DT Vision Foundry

After building the tool, you need to register the tool with the DT Vision Foundry main application using the DTTools.ini initialization file. The DTTools.ini file is a standard Windows initialization file that must be located in the same directory as the DT Vision Foundry application (DTVF.exe). Both DTTools.ini and DTVF.exe are located in C:\Program Files\Data Translation\DT Vision Foundry\BIN, by default. Do not put the DTTools.ini initialization file in the Windows system directory. You edit the DTTools.ini file using any text editor, such as Notepad.

Each tool should have an entry similar to the following:

```
[Tool1]
LOCATION=..\Tools\DT_Arith\DT_Arith.dll
TOOLBAR=Image Processing
AUTOSTART=NO
```

The tool numbers must be unique and sequential starting from 1.

For the LOCATION entry, you can specify either a path relative to the location of the DT Vision Foundry application (DTVF.exe) or an absolute path. For example, the following entries are both supported:

```
LOCATION=..Tools\DT_Arith\DT_Arith.dll
LOCATION=D:\Program Files\Data Translation\
  DT Vision Foundry\Tools\DT_Arith\DT_Arith.dll
```

If you do not include a TOOLBAR entry, the tool is placed in the Miscellaneous toolbar.

The AUTOSTART entry determines whether you want DT Vision Foundry to automatically start the tool at program startup. If you do not include an AUTOSTART entry, DT Vision Foundry assumes that AUTOSTART=NO.

If you are compiling a debug version of your custom tool, you need to run the debug version of DT Vision Foundry (DTVFD.exe) and edit its associated DTTools.ini file (DTToolsD.ini). These files are located in C:\Program Files\Data Translation\DT Vision Foundry\BIN, by default.

For example, if you tool is named MyTool.dll, it is located in C:\DTVF, and you have 20 other tools in your system, add the following lines to the end of the DTTools.ini file:

```
[TOOL21]
LOCATION=C:\DTVF\MYTOOL.DLL
```

After saving the DTTools.ini file, you can start DT Vision Foundry. Your new tool appears in both the Tools menu and the Miscellaneous toolbar.

Customizing the Look of Your Tool

Now that you have the example change tool up and running with DT Vision Foundry, you need to customize the example change 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 DT Vision Foundry 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 toolbars and when the tool is iconized. The icons that appear in the toolbars are the bitmaps in the RC file named "Pressed" and "Unpressed." They represent how the icons in the toolbars 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 DT Vision Foundry 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:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\ Examples\ Tools\Change, 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 DT Vision Foundry, you can now run DT Vision Foundry 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 DT Vision Foundry.

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:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, by default. The code has error checking and variable declaration removed to simplify the code and amplify the main idea of how to use DT Vision Foundry 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 CcDTTool::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 DT Vision Foundry 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 DT Vision Foundry. It should work the same way as the DT Vision Foundry Pixel Change tool, but not as fast.

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 DT Vision Foundry notification messages.

Note: The code for this is section of the program is located in C:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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 CcDTTool::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 CcDTTool::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 997. 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 DT Vision Foundry. 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/machine vision 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:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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.

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:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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.

Speeding Up the Execution of a Tool

You can use an image processing and machine vision 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 DT Vision Foundry

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. DT Vision Foundry 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 DT Vision Foundry

After you have derived an algorithm, you will most likely want to run it repeatedly. If you are running a machine vision 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. DT Vision Foundry 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 50, (where 5 is the center pixel).

Table 50: 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:\Program Files\Data Translation\DT Vision Foundry\C++ Devel\Examples\Tools\Change, 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( );
}
```

A

Vendor-Specific Properties and Values

The **SetDeviceProperty** and **GetDeviceProperty** methods of the Picture tool, described starting on page 712, require values for the *nPropId* and *nValue* parameters.

The values for the parameters when using the MACH I Series, which include the DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series boards, are listed in Table 51 on page 1093.

The values for the parameters when using the MACH II Series, which currently includes the DT3162 board only, are listed in Table 52 on page 1112.

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards

nPropID	Description	Boards Supported	nValue
ActiveLine Count	Sets and returns the height of the active video area.	DT3152 DT3152-LS DT3157	For DT3152 and DT3152-LS, 1 to 4096. For DT3157, 1 to 4096 in single-channel mode, or 2 to 4096 in dual-channel mode.
ActivePixel Count	Sets and returns the width of the active video area.	DT3152 DT3152-LS DT3157	For DT3152 and DT3152-LS, 4 to 4096. For the DT3157, 4 to 4096 in single-channel mode, or 32 to 1024 in dual-channel mode.
BackPorch Start	Sets and returns the start of the back porch.	DT3152 DT3152-LS	0 to 4095
BlueOffset	Sets and returns the blue offset value.	DT3154	0 to 305,550 µV
Blue Reference	Sets and returns the blue reference value.	DT3154	338,000 to 1,199,966 µV

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
Brightness	Sets and returns the brightness level.	DT3153 DT3120 DT3130 Series	0 to 255, where 0 represents the least brightness and 255 represents the most brightness.
ClampEnd	Sets and returns the clamp end position.	DT3152 DT3152-LS	0 to 4095
ClampStart	Sets and returns the clamp start position.	DT3152 DT3152-LS	0 to 4095
Clock Frequency	Sets and returns the frequency of the internal pixel clock.	DT3152 DT3152-LS DT3157	1000 to 20,000,000 Hz
ClockSource	Sets and returns the pixel clock source.	DT3152 DT3152-LS DT3157	**ExternalClock** (clocking is generated by an outside source), or **InternalClock** (clocking is generated by the frame grabber board).
Clock Transition	Sets and returns the edge at which an external pixel clock pulse occurs.	DT3152 DT3152-LS DT3157	**OnHighToLow** (increment clock on a falling edge), or **OnLowToHigh** (increment clock on a rising edge).

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
Contrast	Sets and returns the contrast level.	DT3153 DT3120 DT3130 Series	0 to 511, where 0 represents the least contrast and 511 represents the most contrast.
DigitalCamera Type	Sets and returns the type of the currently attached digital camera.	DT3157	**CAM_10BIT** (10-bit video from one digital input line), **CAM_12BIT** (12-bit video from one digital input line), **CAM_14BIT** (14-bit video from one digital input line), **CAM_16BIT** (16-bit video from one digital input line), **CAM_8BIT_DUAL** (8-bit video from two digital input lines), or **CAM_8BIT_SINGLE** (8-bit video from one digital input line).
EnableExpose Pulse	Enables and disables the generation of an exposure pulse to a digital camera.	DT3157	**True** (the exposure pulse is enabled), **False** (the exposure pulse is disabled).
ExposePulse Duration	Sets and returns the exposure pulse width.	DT3157	82 µs to 1.33 s

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
ExposePulse Polarity	Sets and returns the active state of the exposure pulse (active high or active low).	DT3157	**ActiveHigh** (the exposure pulse is active when the signal is high), or **ActiveLow** (the exposure pulse is active when the signal is low).
FirstActive Line	Sets and returns the top-most pixel in the active video area.	DT3152 DT3152-LS DT3157 DT3120 DT3130 Series	For the DT3152, DT3152-LS, and DT3157, 0 to 4095. For the DT3120 and DT3130 Series, 0 to 255.
FirstActive Pixel	Sets and returns the left-most pixel in the active video area.	DT3152 DT3152-LS DT3157 DT3120 DT3130 Series	For the DT3152 and DT3152-LS, 0 to 4095. For the DT3175, 4 to 4095. For the DT3120 and DT3130 Series, 0 to 255.

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
FrameLeft	Returns the position of the first pixel in the active frame buffer.	DT3152 DT3152-LS DT3153 DT3154 DT3157 DT3120 DT3130 Series	For the DT3152, DT3152-LS, and DT3157, 0 to 4095. For the DT3153 and DT3154, 0 to 767 (50 Hz) or to 636 (60 Hz). For the DT3120 and DT3130 Series, 0 to 763 (50 Hz) or to 635 (60 Hz).
FrameTop	Returns the position of the first line in the active frame buffer.	DT3152 DT3152-LS DT3153 DT3154 DT3157 DT3120 DT3130 Series	For the DT3152, DT3152-LS, and DT3157, 0 to 4095. For the DT3153, DT3120, and DT3130 Series, 0 to 575 (50 Hz) or to 479 (60 Hz). For the DT3154, 0 to 572 (50 Hz) or to 476 (60 Hz).

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
FrameType	Returns the type of frame (or field) that you want to acquire in the active frame buffer.	All MACH I Series boards	For the DT3152, DT3152-LS, and DT3154 **InterlacedStartOnEven** (acquire interlaced frames, starting with the next even field of an interlaced frame), **InterlacedStartOnNext** (acquire interlaced frames, starting with the next field (of any kind) of an interlaced frame), **InterlacedStartOnOdd** (acquire interlaced frames, starting with the next odd field of an interlaced frame), or **NonInterlaced** (acquire noninterlaced frames, starting with the next field of a noninterlaced frame).

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
FrameType (cont.)	Returns the type of frame (or field) that you want to acquire in the active frame buffer.	All MACH I Series boards	For DT3153 and DT3155, **InterlacedStartOnEven** (acquire interlaced frames, starting with the next even field of an interlaced frame), **InterlacedStartOnNext** (acquire interlaced frames, starting with the next field (of any kind) of an interlaced frame), or **InterlacedStartOnOdd** (acquire interlaced frames, starting with the next odd field of an interlaced frame). For DT3157, **DefaultFrame** (acquire fields/frames of the default type).

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
FrameType (cont.)	Returns the type of frame (or field) that you want to acquire in the active frame buffer.	All MACH I Series boards	For the DT3120 and DT3130 Series, **InterlacedStartOnEven** (acquire interlaced frames, starting with the next even field of an interlaced frame), **InterlacedStartOnNext** (acquire interlaced frames, starting with the next field (of any kind) of an interlaced frame), **InterlacedStartOnOdd** (acquire interlaced frames, starting with the next odd field of an interlaced frame), **EvenField** (acquire even fields, starting with the next even field), or **OddField** (acquire odd fields, starting with the next odd field).

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
Gain	Sets and returns the gain.	DT3152 DT3152-LS	GAIN_OF_1, GAIN_OF_2, GAIN_OF_4, or GAIN_OF_POINT_5 (gain of 0.5).
GreenOffset	Sets and returns the green offset value.	DT3154	0 to 305,550 μV
Green Reference	Sets and returns the green reference value.	DT3154	338,000 to 1,199,966 μV
Horizontal Frequency	Sets and returns the frequency of the horizontal sync signal for Sync Master mode.	DT3152 DT3152-LS DT3157	1 to 2,000,000 Hz.
Horizontal PulseWidth	Sets and returns the width of the horizontal sync signal for Sync Master mode.	DT3152 DT3152-LS DT3157	250 to 950,000,00 ns
Horizontal Transition	Sets and returns the edge at which the horizontal sync signal occurs for variable-scan video signals.	DT3152 DT3152-LS	**OnHighToLow** (horizontal transitions occur on a falling edge), or **OnLowToHigh** (horizontal transitions occur on a rising edge).
HSyncInsert Pos	Sets and returns the horizontal sync insert position.	DT3152 DT3152-LS DT3157	The percentage of total pixels per line, multiplied by 100.

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
HSyncSearch Pos	Sets and returns the horizontal sync search position.	DT3152 DT3152-LS DT3157	The percentage of total pixels per line, multiplied by 100.
Hue	Sets and returns the hue level.	DT3153 DT3120 DT3130 Series	0 to 255, where 0 represents red and 255 represents purple.
InputFilter	Sets and returns the type of input filter used.	DT3152 DT3152-LS DT3155	For the DT3152 and DT3152-LS, **AC_50Hz_Filter** (AC coupled with 50 Hz (4.43 MHz) filter), **AC_60Hz_Filter** (AC coupled with 60 Hz (3.58 MHz) filter), **AC_No_Filter** (AC coupled with no filter), or **DC_No_Filter** (DC coupled with no filter). For the DT3155, **AC_50Hz_Filter** (AC coupled with 50 Hz (4.43 MHz) filter), **AC_60Hz_Filter** (AC coupled with

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
InputFilter (cont.)	Sets and returns the type of input filter used.	DT3152 DT3152-LS DT3155	60 Hz (3.58 MHz) filter), or **AC_No_Filter** (AC coupled with no filter).
InputLUT	Sets and returns the ILUT to use.	DT3152 DT3152-LS DT3154 DT3155 DT3157	For the DT3152, DT3152-LS, DT3155, and DT3157, one ILUT is available (0). For the DT3154, six ILUTs are available (0 for RVID0, 1 for GVID0, 2 for BVID0, 3 for RVID1, 4 for GVID1, or 5 for BVID1).
MultTrigger Mode	Sets and returns the trigger mode for a multiple-frame acquisition.	All MACH I Series boards	For the DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3120, and DT3130 Series, **TriggerForEach** (a separate trigger starts the acquisition of each frame in a series of multiple frames), or **TriggerToStart** (a

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
MultTrigger Mode (cont.)	Sets and returns the trigger mode for a multiple-frame acquisition.	All MACH I Series boards	single trigger starts the acquisition of a series of multiple frames). For the DT3157, **TriggerToStart** (a single trigger starts the acquisition of a series of multiple frames).
MultTrigger Transition	Sets and returns the edge at which an external trigger occurs for a multiple-frame acquisition.	All MACH I Series boards	**OnHighToLow** (trigger on a falling edge), or **OnLowToHigh** (trigger on a rising edge).
MultTrigger Type	Sets and returns the trigger type (software or external) for a multiple-frame acquisition.	All MACH I Series boards	**ExternalTrigger** (the frame grabber board is triggered through a dedicated external line), or **SoftwareTrigger** (external triggers are disabled. The frame grabber board is triggered through a software command).

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
Offset	For monochrome frame grabber boards, sets and returns the offset voltage.	DT3152 DT3152-LS DT3155	For the DT3152 and DT3152-LS, -1,075,200 to 1,066,800 µV. For the DT3155, -306,000 to -1.275 µV.
Phase	Sets and returns the phase between the horizontal and vertical sync signals for Sync Master mode.	DT3152 DT3152-LS DT3157	The percent of the total line that the vertical sync is shifted relative to the horizontal sync, multiplied by 100.
RedOffset	Sets and returns the red offset value.	DT3154	0 to 305,550 µV
RedReference	Sets and returns the red reference value.	DT3154	338,000 to 1,199,966 µV
Reference	For monochrome frame grabber boards, sets and returns the reference voltage.	DT3152 DT3152-LS DT3155	For the DT3152 and DT3152-LS, 0 to 1,275,000 µV. For the DT3155, 45,100 to 1,007,725 µV.
SyncMaster Enabled	Enables and disables Sync Master mode.	DT3152 DT3152-LS DT3153 DT3157	**True** (Sync Master mode is enabled), or **False** (Sync Master mode is disabled).

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
SyncSentinel	Enables and disables the Sync Sentinel.	DT3152 DT3152-LS DT3154 DT3155 DT3157	**True** (Sync Sentinel is enabled), or **False** (Sync Sentinel is disabled).
SyncSource	Sets and returns the sync source for composite video signals.	DT3152 DT3152-LS DT3153 DT3154 DT3155 DT3120 DT3130 Series	For the DT3152, DT3152-LS, and DT3155, **Channel0**, **Channel1**, **Channel2**, or **Channel3**. For the DT3153, DT3120, and DT3130 Series, **CurrentSource** (the channel currently being digitized). For the DT3154, **Channel0** (RVID0), **Channel1** (GVID0), **Channel2** (BVID0), **Channel3** (RVID1), **Channel4** (GVID1), **Channel5** (BVID1), or **ExternalLine** (an external sync source).

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
SyncThresh	Sets and returns the sync threshold for composite video signals.	DT3152 DT3152-LS DT3154 DT3155	For the DT3152, DT3152-LS, DT3155, 50, 75, 100, or 125 mV. For the DT3154, 50 or 125 mV.
TotalLinesPer Field	Sets and returns the size of the entire area between two consecutive vertical sync signals.	DT3152 DT3152-LS DT3157	1 to 4096
TotalPixels PerLine	Sets and returns the size of the entire area between two consecutive horizontal sync signals.	DT3152 DT3152-LS DT3157	4 to 4096
Trigger Transition	Sets and returns the edge at which an external trigger occurs for a single-frame acquisition.	DT3152 DT3152-LS DT3153 DT3154 DT3155 DT3120 DT3130 Series	**OnHighToLow** (trigger on a falling edge) or **OnLowToHigh** (trigger on a rising edge)

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
TriggerType	Sets and returns the trigger type (software or external) for a single-frame acquisition.	All MACH I Series boards	**ExternalTrigger** (the frame grabber board is triggered through a dedicated external line), or **SoftwareTrigger** (external triggers are disabled. The frame grabber board is triggered through a software command).
USaturation	Sets and returns the U-saturation level.	DT3153 DT3120 DT3130 Series	0 to 511, where 0 represents all white and 511 represents pure color (green and red) with no white.
Vertical Frequency	Sets and returns the frequency of the vertical sync signal for Sync Master mode.	DT3152 DT3152-LS DT3157	For the DT3152 and DT3152-LS, 1 to 200,000 Hz. For the DT3157, 0.00024 to 488.28 Hz.
VerticalPulse Width	Sets and returns the width of the vertical sync signal for Sync Master mode.	DT3152 DT3152-LS DT3157	For the DT3152 and DT3152-LS, 250 to 950,000,000 ns. For the DT3157, 500 to 950,000,000 ns.

A

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
Vertical Transition	Sets and returns the edge at which the vertical sync signal occurs for variable-scan video signals.	DT3152 DT3152-LS	**OnHighToLow** (vertical transitions occur on a falling edge), or **OnLowToHigh** (vertical transitions occur on a rising edge).
VideoSignal Type	Sets and returns the type of video signal (composite, variable-scan, Y/C, or RGB).	All MACH I Series boards	For the DT3152 and DT3152-LS, **CompositeVideo** (horizontal and vertical syncs are combined into one signal), or **VarScanVideo** (variable-scan video; horizontal and vertical syncs are on separate signals). For the DT3153, DT3120, and DT3130 Series, **CompositeVideo** (horizontal and vertical syncs are combined into one signal), or **YCVideo** (luminance (Y) information and

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
VideoSignal Type (cont.)	Sets and returns the type of video signal (composite, variable-scan, Y/C, or RGB).	All MACH I Series boards	color (C) information are stored separately). For the DT3154, **RGBVideo** (red, green and blue image data reside on separate signals). For the DT3155, **CompositeVideo** (horizontal and vertical syncs are combined into one signal). For the DT3157, **VarScanVideo** (variable-scan video; horizontal and vertical syncs are on separate signals).
VSaturation	Sets and returns the V-saturation level.	DT3153 DT3120 DT3130 Series	0 to 511, where 0 represents all white and 511 represents pure color (blue and green) with no white .

Table 51: Property IDs for MACH I Series (DT3152, DT3152-LS, DT3153, DT3154, DT3155, DT3157, DT3120, and DT3130 Series) Boards (cont.)

nPropID	Description	Boards Supported	nValue
VSyncInsert Pos	Sets and returns the vertical sync insert position.	DT3152 DT3152-LS DT3157	The percentage of the total lines per field, multiplied by 100.
VSyncSearch Pos	Sets and returns the vertical sync search position.	DT3152 DT3152-LS DT3157	The percentage of the total lines per field, multiplied by 100.

Table 52: Prop IDs for MACH II Series (DT3162) Boards

nPropID	Description	nValue
AcquireType	Sets and returns the type of frames/fields to acquire.	**Progressive** (acquire progressive scans - noninterlaced frames - starting with the next field), **Interlaced** (acquire interlaced frames - the starting field depends on the value set for FirstActiveLine and RoiTop), **InterlacedEvenFieldOnly** (acquire even fields of interlaced frames, starting with the next even field), **InterlacedOddFieldOnly** (acquire odd fields of interlaced frames, starting with the next odd field).
ActiveLineCount	Sets and returns the number of lines per frame in the active video area.	1 to 2047, in increments of 1
ActiveLUT	Sets and returns the look-up table (LUT) that you want to use.	0 or 1
ActivePixelCount	Sets and returns the number of pixels per line in the active video area.	1 to 2047, in increments of 1
Brightness	Sets and returns the brightness level for the incoming video.	0 to 255, where decreasing values make the digitized video darker and increasing values make the digitized video lighter.

Table 52: Prop IDs for MACH II Series (DT3162) Boards (cont.)

nPropID	Description	nValue
ClampEnd	Sets and returns the clamp end position.	0 to 4095, in increments of 1
ClampStart	Sets and returns the clamp start position.	0 to 4095, in increments of 1
Contrast	Sets and returns the contrast level for the incoming video.	0 to 99, where decreasing values contract the grayscale range of the digitized video and increasing values stretch the grayscale range of the digitized video.
ExposeEnabled	Enables and disables the expose/reset pulse.	**True** (the expose/reset pulse is enabled), or **False** (the expose/reset pulse is disabled).
ExposePolarity	Sets and returns the polarity of the expose/reset pulse	**ActiveHigh** (high-going pulse), or **ActiveLow** (low-going pulse).
ExposeStartLine	Sets and returns the start position of the expose/reset pulse.	1 to 65,535, in increments of 1
ExposeStopLine	Sets and returns the stop position of the expose/reset pulse.	1 to 65,535, in increments of 1
FirstActiveLine	Sets and returns the position of the beginning of the active video signal within the field.	0 to 2047, in increments of 1
FirstActivePixel	Sets and returns the position of the beginning of the active video signal on the line.	0 to 4095, in increments of 1

Table 52: Prop IDs for MACH II Series (DT3162) Boards (cont.)

nPropID	Description	nValue
HSyncInPolarity	Sets and returns the polarity of the horizontal (line) sync of an incoming variable-scan video signal.	**ActiveHigh** (high-going horizontal sync), or **ActiveLow** (low-going horizontal sync).
HSyncOutPulse Width	Sets and returns the width of the outgoing horizontal sync pulse.	1 to 4095, in increments of 1
LineFrequency	Sets and returns the frequency of a horizontal video line (how often the line occurs).	1 to 65,535, in increments of 1
PixelClock Source	Sets and returns the pixel clock source.	**InternalClock** (clocking is generated by the device), **ExternalSource1** (clocking is generated by an external TTL signal provided to the device), or **ExternalSource2** (clocking is generated by an external differential signal provided to the device).
RoiHeight	Sets and returns the number of lines in the ROI that you want to save.	1 to 2048, in increments of 1
RoiLeft	Sets and returns the position of the first pixel of the ROI that you want to save, relative to the active video area.	0 to 2039, in increments of 1

Table 52: Prop IDs for MACH II Series (DT3162) Boards (cont.)

nPropID	Description	nValue
RoiTop	Sets and returns the position of the first line of the ROI that you want to save, relative to the active video area.	0 to 2047, in increments of 1
RoiWidth	Sets and returns the number of pixels in each line of the ROI that you want to save.	8 to 2048, in increments of 8
StrobeEnabled	Enables and disables the strobe output pulse.	**True** (the strobe pulse is enabled), or **False** (the strobe pulse is disabled).
StrobePolarity	Sets and returns the polarity of the strobe output pulse.	**ActiveHigh** (high-going pulse), or **ActiveLow** (low-going pulse).
StrobeStartLine	Sets and returns the start position of the strobe output pulse.	1 to 65,535, in increments of 1
StrobeStopLine	Sets and returns the stop position of the strobe output pulse.	1 to 65,535, in increments of 1

Table 52: Prop IDs for MACH II Series (DT3162) Boards (cont.)

nPropID	Description	nValue
SyncInSource	Sets and returns the source of the incoming sync signal.	**InternalSync** (the horizontal and vertical syncs are extracted from the incoming video signal as is the case for composite signals), or **ExternalSync** (the horizontal and vertical syncs come from the external horizontal and vertical sync input lines as is the case for variable-scan signals).
SyncOutEnabled	Enables/disables the outgoing sync signal.	**True** (the outgoing sync signal is enabled), or **False** (the outgoing sync signal is disabled).
SyncOutPolarity	Sets and returns the polarity of the outgoing sync pulses.	**ActiveHigh** (high-going pulses.) or **ActiveLow** (low-going pulses).
TotalLinesPer Frame	Sets and returns the total number of lines in a frame of video.	1 to 4095, in increments of 1
TotalPixelsPer Line	Sets and returns the total number of pixels in a single horizontal line of video (the number of pixels that are not black).	1 to 4095, in increments of 1
TriggerTransition	Sets and returns the transition type for an external line used to trigger an acquisition.	**OnHighToLow** (trigger on a falling edge), or **OnLowToHigh** (trigger on a rising edge).

Table 52: Prop IDs for MACH II Series (DT3162) Boards (cont.)

nPropID	Description	nValue
TriggerType	Sets and returns the trigger type.	**ExternalForEach** (a separate external trigger starts the acquisition of each frame/field in a series of frames/fields), **ExternalToStart** (a single external trigger starts the acquisition of all frames/fields in a series of frames/fields), or **Internal** (external triggers are disabled - the acquisition of frames/fields is triggered by calling the AcquireMem method).
VSyncDelay	Sets and returns the number of lines between the end of the expose/reset pulse and the beginning of the vertical sync.	1 to 65,535, in increments of 1
VSyncInPolarity	Sets and returns the polarity of the vertical (field) sync of an incoming variable-scan video signal.	**ActiveHigh** (high-going vertical sync), or **ActiveLow** (low-going vertical sync).
VSyncOutPulse Width	Sets and returns the width of the outgoing vertical sync pulse.	1 to 4095, in increments of 1

Index

Symbols

Numerics

A

B

D

G

I

K

L

M

N

O

P

R

S

U

V

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.

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.

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.