Meta Imaging Series® Software
MetaMorph® Software
Visual Basic Reference Guide

Version 7.6.5 for
Microsoft Windows XP
and Microsoft Windows Vista
This document is provided to customers who have purchased Molecular Devices, Inc. ("Molecular Devices") equipment, software, reagents, and consumables to use in the operation of such Molecular Devices equipment, software, reagents, and consumables. This document is copyright protected and any reproduction of this document, in whole or any part, is strictly prohibited, except as Molecular Devices may authorize in writing.

Equipment, software, reagents, and consumables that may be described in this document are protected under one or more patents filed in the United States, Canada, and other countries. Additional patents are pending.

Software that may be described in this document is furnished under a license agreement. It is against the law to copy, modify, or distribute the software on any medium, except as specifically allowed in the license agreement. Furthermore, the license agreement may prohibit the software from being disassembled, reverse engineered, or decompiled for any purpose.

Portions of this document may make reference to other manufacturers and/or their products, which may contain parts whose names are registered as trademarks and/or function as trademarks of their respective owners. Any such usage is intended only to designate those manufacturers' products as supplied by Molecular Devices for incorporation into its equipment and does not imply any right and/or license to use or permit others to use such manufacturers' and/or their product names as trademarks.

Molecular Devices makes no warranties or representations as to the fitness of this equipment for any particular purpose and assumes no responsibility or contingent liability, including indirect or consequential damages, for any use to which the purchaser may put the equipment described herein, or for any adverse circumstances arising therefrom.

For research use only. Not for use in diagnostic procedures.

META IMAGING SERIES, METAMORPH, METAFLUOR and METAVUE are registered trademarks of Molecular Devices, Inc. These trademarks may not be used in any type of promotion or advertising without the prior written permission of Molecular Devices, Inc.

Equipment built by Molecular Devices, Inc.
1311 Orleans Drive, Sunnyvale, California, United States of America 94089.
Molecular Devices, Inc. is ISO 9001 registered.
© 2010 Molecular Devices, Inc.
All rights reserved.
Printed in the USA.
## Contents

### Chapter 1: Introduction
- Overview
- Conventions Used in This Manual
- The Run User Program Command
- Visual Basic and User Programs
- Creating a User Program
- Data Types and Arrays

### Chapter 2: Performing Serial and Digital I/O Communication
- Overview
- Performing Serial Data Transmission
- Communicating with a Digital I/O Device

### Chapter 3: Executing Commands and Journals
- Overview
- Executing Commands and Journals

### Chapter 4: Reading and Manipulating Images and Image Windows
- Overview
- Loading, Creating, Copying, and Closing Images
- Finding Loaded Images
- Manipulating Image Windows
- Reading and Using Image Properties

### Chapter 5: Adjusting Image Display
- Overview
- Updating the Image After Changing the Display
- Adjusting Brightness and Contrast
- Autoscaling 16-Bit Images
- Working with Look-up Tables and Palettes

### Chapter 6: Reading and Using Image Pixel Data
- Overview
- Applying Thresholding
- Reading and Manipulating Image Data

### Chapter 7: Working with Regions of Interest
- Overview
- Creating and Removing Regions
- Finding Regions
- Reading and Manipulating Region Properties
- Reading Image Data from Regions

### Chapter 8: Performing Morphometry
- Overview
- Configuring Measurement Preferences
- Configuring Object Measurements
- Configuring Classifier Filters
- Measuring All Objects in an Image
- Measuring Single Objects
Figures

Introduction

1.1 The Run User Program Dialog Box ................................................................. 6
1.2 Class Module UserMethods Section Code .................................................... 14

Reading and Manipulating Images and Image Windows

4.1 Image “Get” Function Programming Example ............................................... 56

Working with Regions of Interest

7.1 Region Property “Get” Function Programming Example .............................. 106
7.2 Region Data “Get” Function Programming Example ..................................... 112

Performing Morphometry

8.1 Single Object Data “Get” Function Programming Example .......................... 132
## Tables

### Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Creating a User Program with Visual Basic .NET 2005/2008</td>
<td>11</td>
</tr>
<tr>
<td>1.2</td>
<td>Creating a User Program with Visual Basic .NET 2002/2003</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>Creating a User Program with Visual Basic Version 5 or 6</td>
<td>15</td>
</tr>
<tr>
<td>1.4</td>
<td>Data Types and Arrays</td>
<td>18</td>
</tr>
</tbody>
</table>

### Performing Serial and Digital I/O Communication

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Syntax Rules for Serial Data Transmission</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>ASCII Control Codes</td>
<td>24</td>
</tr>
</tbody>
</table>
Chapter 1 – Introduction

1.1 Overview

Welcome to the Visual Basic Reference Guide for the MetaMorph\textsuperscript{®} software for image processing and analysis, and the Meta Imaging Series\textsuperscript{®} software system. This manual has been designed to serve as a working reference for the use of the extended set of Visual Basic programming functions by the Run User Program drop-in command in the MetaMorph\textsuperscript{®} software. The Run User Program command, located in the File menu, provides you with the ability to run your own Visual Basic programs from within the MetaMorph program. This will allow you to process and analyze images with greater flexibility.

The full power of Microsoft Visual Basic is available to you, with the ability to run conditional “If…Then…Else” routines, to create nested subroutines and loops, and to pass and return values with the imaging system. It is assumed that you already have a working knowledge of Visual Basic and its application. This manual describes the extended set of programming functions that MetaMorph provides, and is intended to serve as a supplement to any Visual Basic references that you may already have.

\textbf{Note:} The following versions of Microsoft Visual Basic can be used to create programs to run in version 7.0 of the MetaMorph software:

- Microsoft Visual Basic .NET 2008 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2005 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2003 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2002 Professional or Enterprise editions
- Microsoft Visual Basic 6.0 Professional or Enterprise editions
- Microsoft Visual Basic 5.0 Professional or Enterprise editions

Note that the Standard and Express editions of the above versions are NOT supported.
This Reference Guide starts with some fundamental concepts about Visual Basic user programs and the Run User Program command. The rest of this manual is devoted to a description of the extended set of functions available for use in MetaMorph. Each entry starts with a short description of what the function does. The full syntax of the expression is given, and the involved parameters and returns are explained. For many entries, a “See Also” list of related or otherwise relevant functions is provided. For the most part, these functions will be in the same section of the manual as the function being described. For those located elsewhere in the manual, the pertinent section will be given in parentheses.

The chapters in this guide are roughly arranged in the order that you might expect to use them through the course of an experiment and thereafter:

- Chapter 1 describes how to install the RUNUSER drop-in and create your own user programs with Microsoft Visual Basic.
- Chapter 2 describes functions that are used to communicate with serial or digital devices.
- Chapter 3 covers execution of commands and journals.
- Chapters 4 and 5 deal with the use of image windows and adjusting their display.
- Chapter 6 concerns thresholding and performing basic densitometric procedures.
- Chapter 7 describes how to work with regions of interest.
- And finally, Chapter 8 ends the manual with a description of the morphometric functions that are available.

Continued on next page
1.1 Overview, continued

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventions Used in This Manual</td>
<td>4</td>
</tr>
<tr>
<td>The Run User Program Command</td>
<td>6</td>
</tr>
<tr>
<td>Visual Basic and User Programs</td>
<td>8</td>
</tr>
<tr>
<td>Creating a User Program</td>
<td>11</td>
</tr>
<tr>
<td>Data Types and Arrays</td>
<td>18</td>
</tr>
</tbody>
</table>
1.2 Conventions Used in This Manual

Before we proceed, it may be useful first to define some of the terms used in this manual and to review some of the typographical conventions that are used.

Some definitions

**Array** A multidimensional (usually one- or two-dimensional) “table” that stores values. For example, you can define an array and read into it the X and Y coordinates of an object’s outline by using the `MorphGetVertexList` function. In Visual Basic, you can declare arrays of up to 60 dimensions. (See Section 1.6, *Data Types and Arrays*.)

**Boolean** In its widest sense, this is a logical operand such as AND, OR, or NOT. When used to describe a variable, a Boolean indicates a logical state, such as TRUE or FALSE. As integers, FALSE is represented by 0 and TRUE is represented by -1 or some other nonzero value.

**Double** A double-precision (64-bit) floating-point number that encodes the value of a variable. Numbers can range from approximately -1.8 X 10^{308} through -4.9 X 10^{324} for negative values and 4.9 X 10^{-324} through 1.8 X 10^{308} for positive values.

**Function** A programming “command” that carries out some procedure. For example, the `PrintMsg` function prompts MetaMorph to display a message box that you have previously configured.

**Handle** An index number used by the program to deal with and keep track of images, image windows, functions, regions of interest, and MetaDevices.

**Integer** In Visual Basic, a 16-bit binary signed (negative or positive) whole number that encodes the value of a variable. Integers can range from -32,767 through 32,767.

**Long** A 32-bit signed whole number that encodes the value of a variable. Numbers can range from -2,147,483,647 through 2,147,483,647.

**Parameter** A value or an identifying element that is passed by the user to the program, such as the handle of an image you want to measure, or a set of coordinates that you want to use to position an image window.

**Return value** A value or state returned by the program in response to a query. For example, the upper and lower limits of a threshold range, expressed as integers, are the return values for the `GetThresholdRange` function.

**Single** A single-precision (32-bit) floating point number that encodes the value of a variable. Numbers can range from approximately -3.4 X 10^{38} through -1.4 X 10^{45} for negative values and 1.4 X 10^{-45} through 3.4 X 10^{38} for positive values.

**String** A segment of alphanumeric text.

**Variable** A placeholder that represents something that is acted upon by a function. For example, the variable `nObjectID` is used to represent the object in an image that is to be deleted by the `MorphDeleteObject` function.

**Variant** A variable that is not explicitly assigned to a particular data type. Variants can store different types of data—integers, floating point numbers, or character strings—depending on the need at the time.

Continued on next page
The following table lists the typographical conventions used in the *Visual Basic Reference Guide*.

<table>
<thead>
<tr>
<th>This:</th>
<th>Represents…</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italics</em></td>
<td>Variables</td>
</tr>
<tr>
<td></td>
<td>Examples: <em>hImage, nXPos</em></td>
</tr>
<tr>
<td><em>Bold</em></td>
<td>Functions</td>
</tr>
<tr>
<td></td>
<td>Examples: <em>LoadImage, SetRegionSize</em></td>
</tr>
<tr>
<td>ALL CAPITALS</td>
<td>Returned or passed states or Booleans</td>
</tr>
<tr>
<td></td>
<td>Examples: <em>EXCLUSIVE, TRUE</em></td>
</tr>
</tbody>
</table>
1.3 The Run User Program Command

MetaMorph allows you to run Visual Basic functions by using the Run User Program drop-in command. As with all other drop-ins, you will need to use the Configure Drop-ins command in the Meta Imaging Series Administrator to load the RUNUSER drop-in prior to starting MetaMorph. This will place the Run User Program command in your File menu, below the Run Program command (don’t confuse the two!).

When you choose Run User Program from the File menu, a dialog box will appear which contains a drop-down list box, a text box, a check box, and four command buttons (see Fig. 1.1). This dialog box provides an interface from within MetaMorph for you to pass parameters to the user program which you will have written in Visual Basic.

Figure 1.1 The Run User Program Dialog Box

Program Name
Contains a list of all of your user programs that are currently registered with the system. The entries in the list will be descriptions that were entered in Visual Basic when you created your program. If there is no description, the project name entered in Visual Basic will appear instead.

Command Line
This is a text field which will be passed to your user program as the parameter for the Startup and DoCommand functions (see Section 1.4). You might use this field, for example, to specify the name of an image that your program will then load, convolve with an image filter, threshold, measure, log measurement data from, and close.

Continued on next page
1.3 The Run User Program Command, continued

Run User Program dialog box options
(continued)

Keep Program in Memory After Execution
This check box determines whether your user program will stay in memory after running, or if it will be unloaded when the routine is completed. If you select the check box, the program will stay in memory, and will therefore run more quickly on subsequent runs. If you clear this check box, the program will be unloaded after running. This may be useful when you are debugging your program, as you can leave both MetaMorph and Visual Basic running at the same time, alternating between running your program and editing it. Visual Basic would not be able to recompile your program if the check box were still selected.

When the Keep Program in Memory check box is selected, the Startup function will be called the first time your program is run after being loaded. Subsequent runs will call the DoCommand function. When this check box is cleared, the Shutdown function will be called after the program finishes. If this check box is cleared before you run the program for the first time, the Startup and Shutdown functions will be called each time you run the program (see Section 1.4 for more about these three functions).

Browse
If the program you want to run does not appear in the Program Name list, this command button will allow you to search your system for it. This button opens the Select Start File dialog box, which is a standard file-selection dialog box that has a Look In drop-down list, Up One Level icon button, File Name text box, and a table that displays the files in the current folder.

When you create a user program, Visual Basic registers it with the system when it is compiled, and it should then be available in the Program Name list. However, if you obtain a program that was compiled elsewhere, it will not have been registered on your system. The Browse command button will register the program and insert it in the Program Name list.

NOTE- this only works for user programs created with Visual Basic 6 and earlier.

Remove
Choosing this button will remove the currently highlighted user program from the Program Name list and unregisters it with the system.

NOTE- this only works for user programs created with Visual Basic 6 and earlier.

OK
Loads the program selected in the Program Name list, passes the parameter you specify in the Command Line text box, runs the user program, and closes the Run User Program dialog box.

Cancel
Cancels any changes made in the Run User Program dialog box and closes it.
1.4 Visual Basic and User Programs

A user program is essentially an ActiveX object that communicates with the MetaMorph application using Microsoft’s object linking and embedding (OLE) interface. However, it is not necessary to be familiar with the details of this communication when using Microsoft Visual Basic, the language supported by the MetaMorph software to write user programs.

Note: The following versions of Microsoft Visual Basic can be used to create programs to run in version 7.0 of the MetaMorph software:

- Microsoft Visual Basic .NET 2008 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2005 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2003 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2002 Professional or Enterprise editions
- Microsoft Visual Basic 6.0 Professional or Enterprise editions
- Microsoft Visual Basic 5.0 Professional or Enterprise editions

Note that the Standard and Express editions of the above versions are NOT supported.

Variables and functions required by the MetaMorph software

To be recognized by the MetaMorph software, a Visual Basic user program must contain three variables, `MM`, `gParentWnd` and `gUserID`, and three functions, `Startup`, `DoCommand`, and `Shutdown`. There are several other more technical requirements, but these will be covered in the discussion of the use of specific versions of Visual Basic (Section 1.5).

The two variables, `gParentWnd` and `gUserID`, are of type Long, and `MM` is a variant:

```
MM
```

`MM` is a variant that is used for all of your communication with the MetaMorph software. For example, to put the handle of the current image in the MetaMorph software into the variable `sourceImage`, you would use the following:

```
Public sourceImage As Long
MM.GetCurrentImage sourceImage
```

`gParentWnd`

This will have the handle of the MetaMorph program’s main window placed in it. It is primarily useful for languages other than Visual Basic which are not yet supported.

`gUserID`

This variable is for future expansion, so you won’t be using it immediately, but it must be present nonetheless.

Continued on next page
Variables and functions required by the MetaMorph software software
(continued)

The three functions are **Startup**, **DoCommand**, and **Shutdown**.

**Startup**(cmdLine As String) As Long (or Integer)

This function is called when the program needs to be loaded into memory to run. `cmdLine` will contain the text entered in the `Command Line` text box of the Run User Program dialog box. **Startup** should return a value of 0 on success and a nonzero value on failure. The return value is currently ignored in MetaMorph, but in the future it may be used.

**DoCommand**(cmdLine As String) As Long (or Integer)

This function is called when the program is run after it has already been loaded into memory. `cmdLine` will contain the text entered in the `Command Line` text box of the Run User Program dialog box. **DoCommand** should return a value of 0 on success and a nonzero value on failure. As with the **Startup** function return, this return value is currently ignored in MetaMorph, but in the future it may be used. If your program behaves the same way whether it is already loaded or not, you may want to have **Startup** simply call **DoCommand**, and put the code that does the work in **DoCommand**.

**Shutdown**( ) As Long (or Integer)

This function is called when the user program is unloaded from memory. **Shutdown** should return a value of 0 on success and a nonzero value on failure. The return value is currently ignored in MetaMorph but in the future it may be used.

Continued on next page
1.4 Visual Basic and User Programs, continued

Class modules

Note: the Class modules section below applies only to Visual Basic versions 5 and 6. It does NOT apply either version of Visual Studio .NET.

All of the above functions and variables must be in a Class Module named UserMethods.

To access the MM variable from other modules, you must declare a Public Variant in one of the other modules, and then in Startup and DoCommand, use Set to assign MM to that variable. For example, suppose your program consists of the Class Module UserMethods, and two regular Modules, Mod1 and Mod2:

Mod1:

```vbnet
Public pubMM As Variant

UserMethods:

Public MM as MMAppLib.UserCall

Function Startup(cmdLine As String) As Long
    DoCommand cmdLine
    End Function

Function DoCommand(cmdLine As String) As Long
    Set pubMM = MM
    Doit
    End Function
```

Mod2:

```vbnet
Function Doit
    Dim i As Long

    pubMM.GetCurrentImage i
    End Function
```

You see that pubMM is visible to the whole program. Set must be used for the assignment, since MM is a variant. If Set is not used, problems will occur.
1.5 Creating a User Program

Introduction

Note: The following versions of Microsoft Visual Basic can be used to create programs to run in MetaMorph 7.0:

- Microsoft Visual Basic .NET 2008 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2005 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2003 Professional or Enterprise editions
- Microsoft Visual Basic .NET 2002 Professional or Enterprise editions
- Microsoft Visual Basic 6.0 Professional or Enterprise editions
- Microsoft Visual Basic 5.0 Professional or Enterprise editions

Note that the Standard and Express editions of the above versions are NOT supported.

Note

One limitation of user programs written in Visual Basic versions 5 and 6 must be noted: because of the manner in which the programs run, the user will not be able to use non-modal dialog boxes. That is, you will be able to work in only one command dialog box at a time. Any attempt to run a program with non-modal dialog boxes will return an error message from the Visual Basic runtime controller.

Visual Studio .NET enables the use of non-modal dialog boxes; however, their use has not been tested with the MetaMorph software and they are not supported by Molecular Devices. Use at your own risk.

Creating a user program with Visual Basic .NET

To create the framework for a user program using Microsoft Visual Basic .NET, use one of the procedures presented in the following tables.

Table 1.1 Creating a User Program with Visual Basic .NET 2005 or 2008

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open Visual Studio and create a new project of type Visual Basic: Class Library.</td>
</tr>
</tbody>
</table>
| 2    | Select My Project in the Solution Explorer  
Select the App Tab and Press "Assembly Info" button  
Check off “MakeAssemblyCom – visible”  
Press OK |
| 3    | Select the References Tab and click "Add" button  
Select COM Tab. Find the MetaMorph Type Library and select it.  
Press OK  
- Adds MetaMorph to the Reference List |
| 4    | In the Solution Explorer, choose the project and right click on it  
Choose Add > New Item  
Choose “module” and click Add |
| 5    | Add the code in Figure 1.2 (below) to Module1.vb. |
| 6    | Add any code for your VB application. |
If deploying this user program to multiple computers with MetaMorph, you must perform the following:

1. In Solution Explorer, highlight the solution, right click on it and choose Add > New Project
2. Find and choose "set up project" and press OK
3. Right click on the set up project and choose Add > Project Output
4. Choose the name of your project from the drop down menu
5. Select Primary Out from the list in the window
6. Choose Configure > Release
7. Press OK
8. From the list of Detected Dependencies, choose “mmapp.exe” and right click on it. Select “Exclude”
9. Under the Build menu, select Build Solution. Once the program has been compiled without any errors, it is ready to be run from the MetaMorph application

**Note**

The default condition for .NET 2005/2008 for Assembly Name and Root Namespace MUST be the same as the project name. DO NOT CHANGE these names or this will not work with the MetaMorph application.

### Table 1.2 Creating a User Program with Visual Basic .NET 2002 or 2003

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 2    | Add the following lines to the AssemblyInfo.vb file:  

```vbnet
<Assembly: ComVisible(True)>  
<Assembly: ClassInterface(ClassInterfaceType.AutoDual)>
```

| 3    | From the Project menu, choose Properties, and go to the General pane in the window that appears (it should start with the General pane automatically). |
| 4    | Bring up the properties for the project, select **Common Properties, General**. Under Output Type, select **Class Library**. Under Root Namespace, type in the name you want to appear in MetaMorph in the Run User Program dialog for your project.  

Note that if you are running Visual Basic.NET Standard Edition, you will not have a Class Library option. In this case, close Visual Studio, and open the folder that contains your project. Find the file with the .vbproj extension and edit it using Notepad. Change the line that says  

```
OutputType = "WinEXE" to OutputType = "Library"
```

Change the line that says  

```
StartupObject = "YourApplicationName.YourFormName" to  
StartupObject = ""
```

Save the changes and close the file. Now start up Visual Studio again and continue with these steps. |
<p>| 5    | Bring up the properties for the project, select Configuration Properties, then Build. Check the <strong>Register for COM Interop</strong> checkbox. |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6</strong></td>
<td>On the Project menu, select Add Reference, select the COM pane, click the Browse button, and find your MMApp.exe file and select it. Then click OK to close the Add Reference window.</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>Add the code in Figure 1.2 (below) to Module1.vb.</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td>Add any code for your VB application.</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td>Under the Build menu, select Build Solution. Once the program has been compiled without any errors, it is ready to be run from MetaMorph</td>
</tr>
</tbody>
</table>
Option Strict Off
Option Explicit On

Public Interface IUserMethods
    Property mm() As MMApplLib.UserCall
    Property gParentWnd() As Integer
    Property gUserID() As Integer

    Function Startup(ByRef cmdLine As String) As Integer
    Function Docommand(ByRef cmdLin As String) As Integer
    Function Shutdown() As Integer
End Interface

<ComClass(UserMethods.ClassId, UserMethods.InterfaceId)> _
Public Class UserMethods
    Implements IUserMethods
    Private mygParentWnd As Integer
    Private mygUserID As Integer
    Public mymm As MMApplLib.UserCall

    Public Const ClassId As String = "832F34A5-5CF5-403f-B4A8-428C8351FD02"
    Public Const InterfaceId As String = "3D8B5BA4-FB8C-5ff8-8468-11BF6BD5CF91"

    Property mm() As MMApplLib.UserCall Implements IUserMethods.mm
        Get
            Return mymm
        End Get
        Set(ByVal Value As MMApplLib.UserCall)
            mymm = Value
        End Set
    End Property

    Property gParentWnd() As Integer Implements IUserMethods.gParentWnd
        Get
            Return mygParentWnd
        End Get
        Set(ByVal Value As Integer)
            mygParentWnd = Value
        End Set
    End Property

    Property gUserID() As Integer Implements IUserMethods.gUserID
        Get
            Return mygUserID
        End Get
        Set(ByVal Value As Integer)
            mygUserID = Value
        End Set
    End Property

    Public Function Startup(ByRef cmdLine As String) As Integer Implements IUserMethods.Startup
        Docommand(cmdLine)
    End Function

    Public Function Docommand(ByRef cmdLine As String) As Integer Implements IUserMethods.Docommand
    End Function

    Public Function Shutdown() As Integer Implements IUserMethods.Shutdown
    End Function
End Class
Creating a User Program, continued

To create the framework for a user program using Microsoft Visual Basic version 5.0 or 6.0, use the procedure presented in the following table.

Table 1.3 Creating a User Program with Visual Basic 5.0 or 6.0

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start the Microsoft Visual Basic program and create a new project by choosing New Project from the File menu.</td>
</tr>
<tr>
<td>2</td>
<td>For the type of project to create, select ActiveX DLL.</td>
</tr>
<tr>
<td>3</td>
<td>From the Project menu, choose Properties, and go to the General pane in the window that appears (it should start with the General pane automatically).</td>
</tr>
<tr>
<td>4</td>
<td>From the Startup Object drop-down list, select Sub Main.</td>
</tr>
<tr>
<td>5</td>
<td>In the Project Name text box, type the name you want to appear in MetaMorph when selecting a user program to run.</td>
</tr>
</tbody>
</table>
| 6    | For Project Description, type the description that you want to appear in MetaMorph when selecting a user program to run.  
**Note:** The project description is not currently recognized by MetaMorph when using VB 5.0/6.0. MetaMorph will display the Project Name in the Run User Program dialog box. |
| 7    | Choose OK. |
| 8    | From the View menu, choose Properties Window. |

Continued on next page
### 1.5 Creating a User Program, continued

Creating a user program with Visual Basic 5.0 or 6.0 or 6.0
(continued)

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 9    | In the Properties window that appears,  
• Set Instancing to 5 – MultiUse, and  
• Set (Name) to UserMethods. |
| 10   | From the Project menu, choose References. Then find *MetaMorph Type Library* in the list and select its check box. If you don’t see *MetaMorph Type Library*, choose the *Browse* button, select Files of Type Executable, navigate to your MetaMorph directory, and select MMAPP.EXE. |
| 11   | Once *MetaMorph Type Library* has been selected, you can choose Object Browser from the View menu and select MMAppLib from the top drop-down list. Then select *UserCall* from the Classes list and you will get a list of all the available functions in MetaMorph that you can call with the MM variable, along with their parameters. When you start to type the name of the object in the module window, a list will appear on the screen with all of the available functions for that object, along with their parameters. |
| 12   | In the Class Module UserMethods section, insert the code shown in Fig. 1.3, which follows this procedure. |
| 13   | From the Project menu, choose Add Module. |
| 14   | In the module you just inserted, add the following code:  
```
Sub Main()
End Sub
```

---

### Compiling the user program

That’s the framework for your program. You can now add your personal code to the program, as described in Section 1.4. When you have finished, you are ready to compile your program. From the File menu, choose *Make YourProjectName*. You can select a name and location for your compiled program, but note that the name you select in this dialog will not be the name that you will use to reference your program in MetaMorph. In MetaMorph, your program will be referenced by the name or description you entered in the *Project Name* text box (Step 5 in table 1.2). Once the program has been compiled without any errors, it is ready to be run from MetaMorph.

*Continued on next page*
1.5 Creating a User Program, continued

Figure 1.3 Class Module UserMethods Section Code

```
Option explicit

Public gUserID As Long
Public gParentWnd As Long
Public MM As MMAppLib.UserCall

Public Function Startup(cmdLine As String) As Long
End Function

Public Function DoCommand(cmdLine As String) As Long
End Function

Public Function Shutdown() As Long
End Function
```
1.6 Data Types and Arrays

Introduction

Visual Basic functions in MetaMorph frequently make use of arrays to handle image pixel data, such as edgelist coordinates and intensity or color values. There is a complex interplay between the size of an array, the image bit-depth, and the type of data (Byte, Integer, or Long) being passed. The number of elements in the array and the range of data that can be stored will be determined by both the bit-depth of the image (1, 8, 16, or 24) and by the data type.

The following table indicates the number of elements in the array as a function of the image depth, data depth, and data type.

Table 1.3 Data Types and Arrays

<table>
<thead>
<tr>
<th>Image Bit-Depth</th>
<th>Data Bit-Depth</th>
<th>Data Type</th>
<th># Elements in Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Byte</td>
<td>number of pixels / 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integer</td>
<td>number of pixels / 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long</td>
<td>number of pixels / 32</td>
</tr>
<tr>
<td>8</td>
<td>Byte</td>
<td></td>
<td>number of pixels(^1)</td>
</tr>
<tr>
<td>16</td>
<td>Integer</td>
<td></td>
<td>number of pixels(^2)</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td></td>
<td>number of pixels(^3)</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>same as for 1-bit images(^4)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Integer</td>
<td></td>
<td>number of pixels</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td></td>
<td>number of pixels(^3)</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>same as for 1-bit images(^4)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Byte</td>
<td></td>
<td>number of pixels(^5)</td>
</tr>
<tr>
<td>16</td>
<td>Integer</td>
<td></td>
<td>number of pixels</td>
</tr>
<tr>
<td>24</td>
<td>Long</td>
<td></td>
<td>number of pixels(^6)</td>
</tr>
</tbody>
</table>

Continued on next page
1.6 Data Types and Arrays, continued

Data Types and Arrays
(continued)

<table>
<thead>
<tr>
<th>Image Bit-Depth</th>
<th>Data Bit-Depth</th>
<th>Data Type</th>
<th># Elements in Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>1</td>
<td>same as for 1-bit images$^4$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Byte</td>
<td>number of pixels$^7$</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Integer</td>
<td>number of pixels$^7$</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Long</td>
<td>number of pixels</td>
</tr>
</tbody>
</table>

$^1$ Pixel values will be converted to 0 or 255.
$^2$ Pixel values will be converted to 0 or 65535.
$^3$ Each pixel’s values will be packed into three bytes, but each of the 32-bit elements in the array will contain four bytes of data such that element one will contain the three values (red, green, blue) of pixel one and the first value of pixel two, the second element will contain the next two values for pixel two and the first two values of pixel three, and so on. Data from 1-bit (binary) images will be expressed as values of 0,0,0 or 255,255,255. Data from 8-bit images will be expressed as a triplet of the pixel value (Value,Value,Value). Data from 16-bit images will be expressed as a triplet of the low byte value of the pixel (Low,Low,Low).
$^4$ Pixel values will be converted to 0 or 1.
$^5$ Only the low byte of each pixel will be stored.
$^6$ Only the low byte from each of the three values (red, green, blue) will be stored.
$^7$ Only the intensity value will be stored. This value will be a numeric average of the red, green, and blue intensities.
Chapter 2 – Performing Serial and Digital I/O Communication

2.1 Overview

Introduction
Peripheral devices can be controlled from within MetaMorph by a number of user program functions. This chapter describes two types of functions used for communication with a peripheral device—those that send and receive data streams over a serial port and those that control a digital device.

In this chapter
This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performing Serial Data Transmission</td>
<td>21</td>
</tr>
<tr>
<td>Communicating with a Digital I/O Device</td>
<td>26</td>
</tr>
</tbody>
</table>
2.2 Performing Serial Data Transmission

Introduction

You can control some devices such as VCRs by sending and receiving sequential streams of data over a serial port. A pair of Visual Basic functions, `SendSerialData` and `WaitForSerialData`, can perform the sending and receiving procedures. You will need to install the CUSTOMIO drop-in with the MetaMorph Drop-in Manager and to install and configure a Data Stream MetaDevice before you can take advantage of these two functions. This section describes the two functions, and provides a list of the syntax rules and a table of the serial command codes that are used for serial communication.

SendSerialData

Description

Carries out the Custom I/O: Send Serial Data command, which sends a sequential stream of data from the computer to another device via a serial port.

Syntax

```
SendSerialData(sData As String, lTimeout As Long, bSendWithEcho As Boolean)
As Long
```

Remarks

This function is a shortcut to the “Send Serial Data” function of the MetaMorph CUSTOMIO drop-in, which provides control over devices that use Data Stream MetaDevices. `SendSerialData` runs without displaying a dialog box. The CUSTOMIO drop-in must be loaded for this function to run.

Parameters

- `sData` Gives the string to be sent to the serial port.
- `lTimeout` Specifies a maximum time, in seconds, that MetaMorph should wait if no echo has been returned before continuing.
- `bSendWithEcho` Determines whether or not MetaMorph is to wait for an echo from the serial device before sending the next character and to warn you if the character is not received. If `bSendWithEcho` is set to TRUE, MetaMorph will wait until the device returns an echo, or until `lTimeout` seconds have transpired, whichever comes first. If `bSendWithEcho` is set to FALSE, MetaMorph will return immediately.

Example

```
' Send the string "TestString" to the serial port with a
' timeout of 5. Do not return until the string has been
' echoed or the timeout has expired.
MM.SendSerialData "TestString", 5, TRUE
```

See also:

`WaitForSerialData`

Continued on next page
2.2 Performing Serial Data Transmission, continued

WaitForSerialData

**Description**
Carries out the Custom I/O: Wait for Serial Data command, which waits for a sequential stream of data from another device by way of a serial port.

**Syntax**
```
WaitForSerialData(sData As String, lTimeout As Long, bLogData As Boolean, sLogFormat As String) As Long
```

**Remarks**
This function is a shortcut to the “Wait for Serial Data” function of the MetaMorph CUSTOMIO drop-in, which provides control over devices that use Data Stream MetaDevices. It runs without displaying a dialog. The CUSTOMIO drop-in must be loaded for this function to run.

**Parameters**
- **sData** Specifies the string from the serial port for which to wait.
- **lTimeout** Specifies the number of seconds *WaitForSerialData* will wait for the string before returning.
- **bLogData** Determines whether or not the received string will be written to a log file. If *bLogData* is set to TRUE and the string that is received matches the one specified in *sData*, the message that you specify with *sLogFormat* will be written to the log file (assuming it is open).
- **sLogFormat** Specifies a message that will be sent to an open log file if the string that is received matches the one specified in *sData* and *bLogData* has been set to TRUE.

**Example**
```
' Wait for 10 seconds for the string "ok" from the serial port. ' Don't log it to the data file.
MM.WaitForSerialData "ok", 10, FALSE, ""
```

**See also:**
SendSerialData

Continued on next page
Table 2.1 describes the syntax rules for the command codes used in serial communication. Be sure to consult Table 2.2 for the codes themselves.

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>&quot;Escape&quot; character, which is ASCII 027 in decimal.</td>
</tr>
<tr>
<td>^A thru ^Z</td>
<td>&quot;Control&quot; character, which is ASCII 001 for ^A through ASCII 26 for ^Z.</td>
</tr>
<tr>
<td>\c</td>
<td>Sends the character after the slash. In this example, the character &quot;c&quot; would be sent. Useful for sending ^, , or $ characters.</td>
</tr>
<tr>
<td>\ddd</td>
<td>Sends ASCII digits in decimal. EXAMPLE: \192.</td>
</tr>
<tr>
<td>\xddd</td>
<td>Sends ASCII digits in hexadecimal. EXAMPLE: \x27.</td>
</tr>
<tr>
<td>(ddddd)</td>
<td>Delays for specified number of milliseconds. EXAMPLE: (1000)</td>
</tr>
</tbody>
</table>

Continued on next page
The following table provides the code strings used in serial communication.

### Table 2.2 ASCII Control Codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Key</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>^@</td>
<td>NUL</td>
<td>Null</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>^A</td>
<td>SOH</td>
<td>Start of Header</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>^B</td>
<td>STX</td>
<td>Start of Text</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>^C</td>
<td>ETX</td>
<td>End of Text</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>^D</td>
<td>EOT</td>
<td>End of Transmission</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>^E</td>
<td>ENQ</td>
<td>Inquiry</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>^F</td>
<td>ACK</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>^G</td>
<td>BEL</td>
<td>Bell</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>^H</td>
<td>BS</td>
<td>Backspace</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>^I</td>
<td>HT</td>
<td>Horizontal Tab</td>
</tr>
<tr>
<td>0A</td>
<td>10</td>
<td>^J</td>
<td>LF</td>
<td>Line Feed</td>
</tr>
<tr>
<td>0B</td>
<td>11</td>
<td>^K</td>
<td>VT</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>0C</td>
<td>12</td>
<td>^L</td>
<td>FF</td>
<td>Form Feed</td>
</tr>
<tr>
<td>0D</td>
<td>13</td>
<td>^M</td>
<td>CR</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>0E</td>
<td>14</td>
<td>^N</td>
<td>SO</td>
<td>Shift Out</td>
</tr>
<tr>
<td>0F</td>
<td>15</td>
<td>^O</td>
<td>SI</td>
<td>Shift In</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>^P</td>
<td>DLE</td>
<td>Data Link Escape</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>^Q</td>
<td>DC1</td>
<td>Device Control 1</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>^R</td>
<td>DC2</td>
<td>Device Control 2</td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>^S</td>
<td>DC3</td>
<td>Device Control 3</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>^T</td>
<td>DC4</td>
<td>Device Control 4</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>^U</td>
<td>NAK</td>
<td>Negative Acknowledge</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
<td>^V</td>
<td>SYN</td>
<td>Synchronous Idle</td>
</tr>
</tbody>
</table>

*Continued on next page*
### 2.2 Performing Serial Data Transmission, continued

#### ASCII control codes (continued)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Key</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>23</td>
<td>^W</td>
<td>ETB</td>
<td>End Transmission Block</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>^X</td>
<td>CAN</td>
<td>Cancel</td>
</tr>
<tr>
<td>19</td>
<td>25</td>
<td>^Y</td>
<td>EM</td>
<td>End of Medium</td>
</tr>
<tr>
<td>1A</td>
<td>26</td>
<td>^Z</td>
<td>SUB</td>
<td>Substitute</td>
</tr>
<tr>
<td>1B</td>
<td>27</td>
<td>ESC</td>
<td>Escape</td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>FS</td>
<td>File Separator</td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>29</td>
<td>GS</td>
<td>Group Separator</td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>30</td>
<td>RS</td>
<td>Record Separator</td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>31</td>
<td>US</td>
<td>Unit Separator</td>
<td></td>
</tr>
</tbody>
</table>
2.3 Communicating with a Digital I/O Device

Introduction

MetaMorph allows you to control digital devices by sending and receiving TTL-level voltage signals through a parallel port or with the use of a digital I/O board. To do so with a user program, you will need to install the CUSTOMIO drop-in with the MetaMorph Drop-in Manager and to install and configure a Digital I/O MetaDevice.

DIGetFirst

Description

Obtains the handle of the first Digital I/O MetaDevice in the current list of Digital I/O MetaDevices.

Syntax

DIGetFirst(hRetDevice As Long) As Long

Return values

hRetDevice Returns the handle of the first Digital I/O MetaDevice in the current list. If there are no Digital I/O MetaDevices, hRetDevice will return a value of -1.

Example

' 'dev' will hold the device handle
Dim dev As Long

' get the handle of the first device and put it in 'dev'
MM.DIGetFirst dev

See also: DIGetNext

DIGetIOStatus

Description

Determines whether a specified I/O line of the given device is an input line or an output line.

Syntax

DIGetIOStatus(hDevice As Long, nLineNumber As Integer, nRetInOut As Integer) As Long

Parameters

hDevice Specifies the handle of the device.

nLineNumber Specifies the number of the I/O line in question. This number will correspond to the pin number on the line’s connector.

Return values

nRetInOut Returns a value corresponding to whether the specified line is an input line or an output line. If it is an input line, a value of 0 will be returned. If it is an output line, a value of 1 will be returned.

Continued on next page
2.3 Communicating with a Digital I/O Device, continued

DIGetIOStatus
(continued)

Example
Dim dev As Long
Dim inout As Integer

' Get the IO status of line 2 of a device and place it in
' 'inout'. 'dev' must have been previously obtained using
' DIGetFirst or DIGetNext
MM.DIGetIOStatus dev, 2, inout

See also: DIGetFirst, DIGetLineCount, DIGetNext

DIGetLineCount

Description
Obtains the number of I/O lines for a given device.

Syntax
DIGetLineCount(hDevice As Long, nRetLines As Integer) As Long

Parameters
hDevice Specifies the handle of the device.

Return values
nRetLines Returns the number of I/O lines.

Example
Dim dev As Long
Dim numLines As Integer

' Get the number of lines on a device and place it in
' 'numlines'. 'dev' must have been previously obtained using
' DIGetFirst or DIGetNext
MM.DIGetLineCount dev, numLines

See also: DIGetFirst, DIGetNext

Continued on next page
2.3 Communicating with a Digital I/O Device, continued

**DIGetLineState**

**Description**
Obtains the state of a specified I/O line (Low vs. High).

**Syntax**

```
DIGetLineState(hDevice As Long, nLineNumber As Integer, nRetState As Integer) As Long
```

**Parameters**
- **hDevice** Specifies the handle of the device.
- **nLineNumber** Specifies the number of the I/O line in question. This number will correspond to the pin number on the line’s connector.

**Return values**
- **nRetState** Returns a value corresponding to the state of the I/O line. If the line is Low, a value of 0 will be returned. If the line is High, a value of 1 will be returned.

**Example**

```
Dim dev As Long
Dim state As Integer

' Get the line state of line 1 of a device and place it in 'state'. 'dev' must have been previously obtained using 'DIGetFirst or DIGetNext.
MM.DIGetLineState dev, 1, state
```

**See also:**
DIGetFirst, DIGetNext

---

**DIGetName**

**Description**
Obtains the name of the MetaDevice whose handle you pass to it.

**Syntax**

```
DIGetName(hDevice As Long, sDeviceName As String) As Long
```

**Parameters**
- **hDevice** Specifies the handle of the device.

**Return values**
- **sDeviceName** Returns the name of the selected MetaDevice. This will be given as a text string.

**Example**

```
Dim dev As Long
Dim name As String

' Get the name of a device and place it in 'name'. 'dev' must ' have been previously obtained using DIGetFirst or DIGetNext.
MM.DIGetName dev, name
```

**See also:**
DIGetFirst, DIGetNext

Continued on next page
2.3 Communicating with a Digital I/O Device, continued

**DIGetNext**

**Description**
Obtains the handle of the Digital I/O MetaDevice that follows the last one that was read (using either `DIGetNext` or `DIGetFirst`).

**Syntax**
```
DIGetNext(hRetDevice As Long) As Long
```

**Return values**
- `hRetDevice` Returns the handle of the Digital I/O MetaDevice that follows the last one read. If there are no more Digital I/O MetaDevices, `hRetDevice` will return a value of -1.

**Example**
```
Dim dev As Long

' Get the third device and place it in 'dev'.

' Get the first device
MM.DIGetFirst dev

' Get the device after the first device (the second device).
MM.DIGetNext dev

' Get the device after the second device.
MM.DIGetNext dev
```

**See also:**
- `DIGetFirst`

---

**SetDigitalIO**

**Description**
Carries out the Custom I/O: Set Digital I/O command, which controls a peripheral digital I/O device by sending signals to it from MetaMorph.

**Syntax**
```
SetDigitalIO(lBitsToSet As Long, lState As Long) As Long
```

**Remarks**
This function is a shortcut to the “Set Digital I/O” function of the MetaMorph CUSTOMIO drop-in. It runs without displaying a dialog box. The CUSTOMIO drop-in must be loaded for this function to run.

**Parameters**
- `lBitsToSet` Provides a 32-bit binary bit pattern that indicates which digital I/O lines will be active. For example, if bit 3 of `lBitsToSet` is set to 1, digital I/O line 3 will be enabled to write to the device. If it is set to 0, line 3 will be disabled from writing.

  - `lState` Provides a 32-bit binary bit pattern that sets the state (High vs. Low) of each I/O line identified by the bit pattern in `lBitsToSet`. To continue the preceding example, if bit 3 of `lState` is set to 0 and `lBitsToSet` has assigned a value of 1 to bit 3, this will set the line state for bit 3 to Low. If `lState` assigns a value of 1 to bit 3, this will set the line state for bit 3 to High. If `lBitsToSet` has assigned a value of 0 to bit 3, the setting of bit 3 for `lState` will be ignored, and the line state of bit 3 will be unaffected.

*Continued on next page*
2.3 Communicating with a Digital I/O Device, continued

SetDigitalIO (continued)

Example

' Set bits 3 and 4 of the DIO lines High. 24 in binary is 00011000 and 255 is 11111111. Since the lState argument had ones in all the first 8 lines, it will set to High whichever of those lines were selected by the lBitsToSet argument.

MM.SetDigitalIO 24, 255

See also: WaitForDigitalIO

WaitForDigitalIO

Description

Carries out the Custom I/O: Wait for Digital I/O command, which configures MetaMorph to wait for specific signals from a peripheral digital I/O device.

Syntax

WaitForDigitalIO(lBitsToRead As Long, lState As Long, lMilliseconds As Long) As Long

Remarks

This function is a shortcut to the “Wait for Digital I/O” function of the MetaMorph CUSTOMIO drop-in. It runs without displaying a dialog. The CUSTOMIO drop-in must be loaded for this function to run.

Parameters

lBitsToRead Provides a 32-bit binary bit pattern that indicates which bits in lState will be compared to data read from the I/O lines. For example, if bit 3 of lBitsToRead is set to 1, WaitForDigitalIO will not return until either digital I/O line 3 matches the state (High vs. Low) that has been set for bit 3 by lState, or until lMilliseconds have elapsed—whichever occurs first.

lState Provides a 32-bit binary bit pattern against which the state (High vs. Low) of each I/O line identified by the bit pattern in lBitsToRead will be compared. To continue the preceding example, if bit 3 of lState is set to 0 and lBitsToRead has assigned a value of 1 to bit 3, this will set the condition to wait until line 3 switches to Low. If lState assigns a value of 1 to bit 3, this will set the condition to wait for line 3 to switch to High.

lMilliseconds Specifies the number of milliseconds to wait before timing out.

Example

' Wait for bits 3, 4, and 7 to be High, Low, and Low, respectively. Timeout if this takes longer than 500 milliseconds. 98 is 10011000 in binary, and 128 is 10000000.

MM.WaitForDigitalIO 98, 128, 500

See also: SetDigitalIO
Introduction

Fundamental to the successful deployment of user programs is the ability to set variables and pass parameters to the functions, and then run the functions. Occasionally, you will need to determine a function’s handle. You may also want to use message boxes to provide feedback while running or troubleshooting the program. This chapter describes the functions you will need for performing all of these procedures.

WARNING

The parts of the interface that are accessed through the RunFunction, RunFunctionEx, and SetFunctionVariable functions are subject to change. Care should be taken when considering whether to use them. These functions should be used only when there is no alternative way to accomplish your task. Because of this, and because of the great number of functions and variables, not all of the functions and variables that can be accessed through the RunFunction and SetFunctionVariable functions have been documented. Information for individual functions and their variables will be provided by e-mail on a case-by-case basis. Send inquiries to: support.dtn@moldev.com.

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executing Commands and Journals</td>
<td>32</td>
</tr>
</tbody>
</table>
3.2 Executing Commands and Journals

GetFunctionHandle

Description Obtains the handle of a specified function.

Syntax

\[
\text{GetFunctionHandle}(\text{sName As String, lRetFunctionHandle As Long}) \text{ As Long}
\]

Remarks You can use the function handles returned by this function when calling \textbf{RunFunctionEx} and \textbf{SetFunctionVariable}.

Parameters

\textit{sName} Supplies the name of the function.

Return values

\textit{lRetFunctionHandle} Returns the function handle.

Example

\begin{verbatim}
' Run the MetaMorph Deinterlace function on the image 'src'. 
' Put the odd and even images in 'destOdd' and 'destEven'.
' 'src', 'destOdd', and 'destEven' should have been previously
' set to valid existing images.
Dim deinterHandle As Long
MM.GetFunctionHandle "Deinterlace", deinterHandle
MM.SetFunctionVariable deinterHandle, "imSource", src
MM.SetFunctionVariable deinterHandle, "imDestOdd", destOdd
MM.SetFunctionVariable deinterHandle, "imDestEven", destEven
MM.RunFunctionEx deinterHandle, 1

' Run Deinterlace again, this time using the current desktop
' image as the source. This time, use RunFunction to run the
' function instead of RunFunctionEx.
MM.GetCurrentImage src
MM.RunFunction "Deinterlace", 1
\end{verbatim}

See also: \textbf{RunFunctionEx}, \textbf{SetFunctionVariable}

PrintMsg

Description Displays a configured message in a diagnostic window in MetaMorph.

Syntax

\[
\text{PrintMsg(sMessage As String)} \text{ As Long}
\]

Remarks This function is useful for the purposes of debugging. You can print diagnostics in a message box without pausing execution. The \textbf{SetPrintMsgSizeAndPosition} function allows you to set properties of the window in which the messages appear.

\textit{Continued on next page}
3.2 Executing Commands and Journals, continued

PrintMsg
(continued)

Parameters

$sMessage$  Specifies the message that is to appear in the message box.

Example

' Print the string "Hello" in the print message window. Place
' the window in the upper left corner of the screen and
' make it 100 pixels wide and 30 high.
$MM.PrintMsg "Hello"
$MM.SetPrintMsgWindowSizeAndPosition 1, 1, 100, 30

See also:

SetPrintMsgSizeAndPosition

RunFunction

Description

Carries out the function named by the function name $sFunctionName$ as if you had
chosen it from a menu.

Syntax

$RunFunction(sFunctionName As String, nMode As Integer) As Long$

Remarks

This function differs from $RunFunctionEx$ in its use of the function’s name.
$RunFunctionEx$ uses the function’s handle.

Parameters

$sFunctionName$  Specifies the name of the function to be run.

$nMode$  Determines the execution mode. If $nMode$ is 0, the function will run
normally. If $nMode$ is 1, the function will run in journal playback mode—for most
functions this means no user interface will be displayed.

Example

' Run the MetaMorph Deinterlace function on the image 'src'.
$MM.GetCurrentImage src$
$MM.RunFunction "Deinterlace", 1

Continued on next page
3.2 Executing Commands and Journals, continued

---

RunFunctionEx

Description
Carries out the function named by the function handle `lFunctionHandle` as if you had chosen it from the menu.

Syntax
```
RunFunctionEx(lFunctionHandle As Long, nMode As Integer) As Long
```

Remarks
This function differs from `RunFunction` in its use of the function’s handle. `RunFunction` uses the function’s name. Function handles are obtained for use with `RunFunctionEx` by calling `GetFunctionHandle`.

Parameters
- `lFunctionHandle` Specifies the handle of the function to be run.
- `nMode` Determines the execution mode. If `nMode` is 0, the function will run normally. If `nMode` is 1, the function will run in journal playback mode—for most functions this means no user interface will be displayed.

Example
' Run the MetaMorph Deinterlace function on the image 'src'.
' Put the odd and even images in 'destOdd' and 'destEven'.
' 'src', 'destOdd', and 'destEven' should have been previously 'set to valid existing images.
Dim deinterHandle As Long
MetaMorph.GetFunctionHandle "Deinterlace", deinterHandle
MetaMorph.SetFunctionVariable deinterHandle, "imSource", src
MetaMorph.SetFunctionVariable deinterHandle, "imDestOdd", destOdd
MetaMorph.SetFunctionVariable deinterHandle, "imDestEven", destEven
MetaMorph.RunFunctionEx deinterHandle, 1

See also: `GetFunctionHandle`

---

Continued on next page
RunJournal

Description
Runs the specified journal.

Syntax
RunJournal(sPath As String) As Long

Remarks
sPath should contain the full path of the journal along with the extension, “.jnl”.

Parameters
sPath  Specifies the path and name of the journal to be run.

Example
' Run a journal named 'closeall.jnl' that resides in the directory C:\MM\Journals.
MM.RunJournal "c:\mm\journals\closeall.jnl"

SetFunctionVariable

Description
Sets the value of a variable in a MetaMorph function.

Syntax
SetFunctionVariable(lFunctionHandle As Long, sVariableName As String, value As Variant) As Long

Remarks
You can obtain the handle of the function whose variable you want to set by using GetFunctionHandle.

Parameters
lFunctionHandle  Gives the handle of the function whose variable you want to set.

sVariableName  Gives the name of the function’s variable that you want to set.

value  Specifies the new value of the variable—this must be of the same data type as the variable you are setting.

Continued on next page
3.2 Executing Commands and Journals, continued

SetFunctionVariable
(continued)

Example

' Run the MetaMorph Deinterlace function on the image 'src'.
' Put the odd and even images in 'destOdd' and 'destEven'.
' 'src', 'destOdd', and 'destEven' should have been previously
' set to valid existing images.
Dim deinterHandle As Long
MM.GetFunctionHandle "Deinterlace", deinterHandle
MM.SetFunctionVariable deinterHandle, "imSource", src
MM.SetFunctionVariable deinterHandle, "imDestOdd", destOdd
MM.SetFunctionVariable deinterHandle, "imDestEven", destEven
MM.RunFunctionEx deinterHandle, 1

' Run Deinterlace again, this time using the current desktop
' image as the source. This time, use RunFunction to run the
' function instead of RunFunctionEx.
MM.GetCurrentImage src
MM.RunFunction "Deinterlace", 1

See also: GetFunctionHandle

SetPrintMsgWindowPositionAndSize

Description
Sets the position and size of the window used by PrintMsg to display messages.

Syntax

SetPrintMsgWindowPositionAndSize(nXPos As Integer, nYPos As Integer, nXSize As Integer, nYSize As Integer) As Long

Parameters

nXPos Specifies the X-coordinate for placement of the upper left corner of the
message box.

nYPos Specifies the Y-coordinate for placement of the upper left corner of the
message box.

nXSize Specifies the width of the message box, in pixels.

nYSize Specifies the height of the message box, in pixels.

Example

' Print the string "Hello" in the print message window. Place
' the window in the upper left corner of the screen and
' make it 100 pixels wide and 30 high.
MM.PrintMsg "Hello"
MM.SetPrintMsgWindowPositionAndSize 1, 1, 100, 30

See also: PrintMsg

Continued on next page
3.2 Executing Commands and Journals, continued

SetMMVariable

Description
Sets the value of a MetaMorph custom variable or certain writable built in variables.

Syntax
SetMMVariable VariableName, value

Parameters
VariableName can be any existing or undefined custom variable, or a built in variable name, such as $Camera.Digital.Exposure$. If it is an undefined custom variable, that variable will be created. value can be a string or a number. If it is not the same type as an existing variable, it will not be set.

Examples
mm.SetMMVariable "MyComment", "Experiment number 7"
mm.SetMMVariable "$Camera.Digital.Exposure", 1250

See also:
GetMMVariable

GetMMVariable

Description
Gets the value of an existing MetaMorph custom or built in variable.

Syntax
GetMMVariable VariableName, var

Parameters
VariableName can be any existing custom variable or a built in variable. var is a VB variable. If the type of var does not match the type of the MetaMorph variable, it will try to convert the type before it assigns it. So, for example, the number 123 assigned to a string will end up as “123”.

Examples
Dim planeNum as Integer
mm.GetMMVariable "$Image.ActivePlane", planeNum
Dim comment as String
mm.GetMMVariable "MyComment", comment

See also:
SetMMVariable
Chapter 4 – Reading and Manipulating Images and Image Windows

4.1 Overview

Introduction
This chapter deals with the functions that you need for managing image windows and their properties. These functions will be central to any user program that you create.

In this chapter
This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loading, Creating, Copying, and Closing Images</td>
<td>39</td>
</tr>
<tr>
<td>Finding Loaded Images</td>
<td>46</td>
</tr>
<tr>
<td>Manipulating Image Windows</td>
<td>49</td>
</tr>
<tr>
<td>Reading and Using Image Properties</td>
<td>53</td>
</tr>
</tbody>
</table>
4.2 Loading, Creating, Copying, and Closing Images

Introduction

Image loading, copying, saving, and closing are vital steps in any user program that you will run in MetaMorph. This section describes the functions that are used for these procedures. In particular, you will make frequent use of the **LoadImage**, **SaveImage**, and **CloseImage** functions.

CloneImage

**Description**

Creates a copy of a specified image.

**Syntax**

```vbscript
CloneImage(hImage As Long, hRetNewImage As Long) As Long
```

**Remarks**

This function differs from **CopyImage** in that **CopyImage** requires the existence of a destination image that will be overwritten. The information that will be copied by **CloneImage** includes the image data, the annotation, zoom factor, calibration setting, thresholding, position, wavelength, scaling, and image times.

**Parameters**

- `hImage`: Specifies the handle of the image to be copied.

**Return values**

- `hRetNewImage`: Returns the handle for the new image.

**Example**

```vbscript
' Create a copy of the current desktop image and place it in ' 'dest'
Dim src As Long, dest As Long
MM.GetCurrentImage src
MM.CloneImage src, dest
```

**See also:**

**CopyImage**, **GetCurrentImage** (Section 4.3), **GetImage** (Section 4.3)
4.2 Loading, Creating, Copying, and Closing Images, continued

**CloseImage**

**Description**
Closes a specified image and removes it from the desktop.

**Syntax**
`CloseImage(hImage As Long) As Long`

**Remarks**
If the image contains unsaved information, you will be prompted before the image is closed. In this regard, `CloseImage` differs from `ForceCloseImage`, which will close a modified image even if it has not been saved first. This function does not affect any disk-based copies of the image.

**Parameters**
- `hImage` Specifies the handle of the image to be closed.

**Example**
```
' Close the current desktop image
Dim src As Long
MM.GetCurrentImage src
MM.CloseImage src
```

**See also:**
`ForceCloseImage`, `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3)

---

**CopyImage**

**Description**
Copies a source image, `hSourceImage`, over a destination image, `hDestImage`.

**Syntax**
`CopyImage(hSourceImage As Long, hDestImage As Long) As Long`

**Remarks**
This function differs from `CloneImage` in that `CopyImage` requires the existence of a destination image, `hDestImage`, that will be overwritten. Only the image data, annotation, Z-position, wavelength, and image times are copied.

**Parameters**
- `hSourceImage` Specifies the handle of the source image to be copied.
- `hDestImage` Specifies the handle of the destination image to be overwritten.

**Example**
```
' Copy the current desktop image to an existing image named
' "Temp"
Dim src As Long, dest As Long
MM.GetCurrentImage src
MM.GetNamedImage "Temp", dest
MM.CopyImage src, dest
```

**See also:**
`CloneImage`, `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3)

*Continued on next page*
4.2 Loading, Creating, Copying, and Closing Images, continued

CopyImagePlane

Description
Copies a specified source image plane to the current plane in a specified destination image.

Syntax
CopyImagePlane(hSourceImage As Long, hDestImage As Long, nSrcPlane As Integer) As Long

Remarks
Plane numbering starts at 0. The information that will be copied includes the image data, the annotation, zoom factor, calibration setting, thresholding, position, wavelength, scaling, and image times.

Parameters
hSourceImage  Specifies the handle of the source image.

hDestImage  Specifies the handle of the destination image into which the plane will be copied.

nSrcPlane  Specifies the number of the plane that will be copied.

Example
' Copy the first plane of the current image to the current plane of an existing image named "Stack".
Dim src As Long, dest As Long
MM.GetCurrentImage src
MM.GetNamedImage "Stack", dest
MM.CopyImagePlane src, dest, 0

See also:
GetCurrentImage (Section 4.3), GetImage (Section 4.3)

CreateImage

Description
Creates a new image, using a specified name, size, and bit-depth.

Syntax
CreateImage(width As Integer, height As Integer, depth As Integer, name As String, hRetNewImage As Long) As Long

Remarks
The SetDisplayedImagesWhenCreated function sets whether or not images will be automatically drawn on the screen when they are created. If you set its bState parameter to FALSE, the image will not appear on the screen when CreateImage is called, although the image will still be created. This may be useful for performing intermediate work on a temporary work image. In this case, you will need to use ShowImage to force the undisplayed image to be displayed.

Continued on next page
4.2  Loading, Creating, Copying, and Closing Images, continued

CreateImage (continued)

**Parameters**
- `width`  Specifies a width (X-axis size), in pixels, for the new image.
- `height` Specifies a height (Y-axis size), in pixels, for the new image.
- `depth`  Specifies a bit-depth for the new image. This must be 1, 8, 16, or 24.
- `name`   Specifies a name for the new image. If an image with that name already exists, “-1” will be appended to the name of the new image.

**Return values**
- `hRetNewImage` Returns the handle of the new image.

**Example**

```
' Create a 512 x 480 pixel 8-bit image named "myimage" and ' place it in 'im'.
Dim im As Long
MM.CreateImage 512, 480, 8, "myimage", im
```

**See also:**  SetDisplayImagesWhenCreated, ShowImage

---

ForceCloseImage

**Description**
Closes a specified image and removes it from the desktop.

**Syntax**

```
ForceCloseImage(hImage As Long) As Long
```

**Remarks**
If the image has not been saved, it will be deleted anyway, and you will lose any unsaved information. In this regard, `ForceCloseImage` differs from `CloseImage`, which will prompt you to save a modified image before closing it. This function does not affect any disk-based copies of the image.

**Parameters**
- `hImage`  Specifies the handle of the image to be closed.

**Example**

```
' Close the current desktop image even if it has not yet been ' saved to disk.
Dim im As Long
MM.GetCurrentImage  im
MM.ForceCloseImage  im
```

**See also:**  CloseImage, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

*Continued on next page*
4.2  Loading, Creating, Copying, and Closing Images, continued

LoadImage

Description
Loads a specified image file.

Syntax
LoadImage(sFileName As String, hRetImage As Long) As Long

Parameters
sFileName  Specifies the name of the image to be loaded. This should contain the complete path of the image.

Return values
hRetImage  Returns the handle of the loaded image.

Example
' Load the image stack Nerve.stk which is stored in the directory C:\MM\Images, and place it in nerveIm.
MM.LoadImage "c:\mm\images\nerve.stk", nerveIm

See also:
GetCurrentImage (Section 4.3), GetImage (Section 4.3)

SaveImage

Description
Saves the specified image to a file on the hard disk.

Syntax
SaveImage(hImage As Long, sFileName As String, bPartial As Boolean, nFormat As Integer) As Long

Remarks
nFormat is a numeric value which denotes the format in which the image is to be saved. This value may be any of the following:

1  Image-1 File Type (*.img)
2  Windows File Type (*.bmp)
3  TIFF File Type (*.tif)
4  Stack File Type (*.stk)
5  MRC-500 File Type (*.pic)
6  Photometrics CC200 File Type (*.cc2)
7  Hamamatsu File Type (*.ham)
8  RGB TIFF File Type (*.tif)
9  WinView File Type (*.spe)
10  Argus File Type (*.ham)

Continued on next page
4.2 Loading, Creating, Copying, and Closing Images, continued

SaveImage (continued)

Parameters

\textit{hImage} \quad \text{Specifies the handle of the image to be saved.}

\textit{sFileName} \quad \text{Gives the name to be used for the image being saved. This should include the full path.}

\textit{bPartial} \quad \text{For an image with an active region of interest, this Boolean logic variable specifies whether only the image inside the region is to be saved (TRUE) or if the whole image is to be saved regardless of the presence of an active region (FALSE). If there is no active region, the whole image will be saved.}

\textit{nFormat} \quad \text{Supplies a numeric value that indicates what format the image is to be saved. (See Remarks.)}

Example

\begin{verbatim}
' Save the current desktop image as a TIFF file named ' Newimage.tif in the directory C:\MM\Images.
Dim currentIm As Long
MM.GetCurrentImage currentIm
MM.SaveImage currentIm, "c:\mm\images\newimage.tif", FALSE, 3
\end{verbatim}

See also: \texttt{GetCurrentImage} (Section 4.3), \texttt{GetImage} (Section 4.3)

SetDisplayImagesWhenCreated

Description

Configures whether or not images are to be drawn automatically on the screen when they are created.

Syntax

\texttt{SetDisplayImagesWhenCreated}(bState As Boolean) As Long

Remarks

\texttt{SetDisplayedImagesWhenCreated} sets \texttt{Use ShowImage} to force an undisplayed image to be displayed.

Parameters

\textit{bState} \quad \text{This Boolean logic variable specifies whether created images are to be displayed (TRUE) or kept hidden (FALSE). If \textit{bState} is set to FALSE when \texttt{CreateImage} or other functions are called which create an image, the image will be created, but will not appear on the screen. This is useful for performing intermediate work on a temporary work image which you don’t necessarily want to see. If \textit{bState} is set to TRUE, new images will be displayed upon creation.}

Continued on next page
4.2 Loading, Creating, Copying, and Closing Images, continued

SetDisplayImagesWhenCreated
(continued)

Example

' Create an image and do some operations on it. Don't let the
' user see the image until all the operations are done.
MM.SetDisplayImagesWhenCreated FALSE
Dim im As Long
MM.CreateImage 512, 480, 8, "temp", im

' Do some operations on 'im'
' Cause 'im' to appear on the screen
MM.ShowImage im, 236

' Reset state so images are once again displayed as soon as
' they are created
MM.SetDisplayImagesWhenCreated TRUE

See also: CreateImage

ShowImage

Description
Causes the specified image to be displayed, if it is not already being displayed.

Syntax
ShowImage(hImage As Long, nPalEntries As Integer) As Long

Parameters
hImage  Specifies the handle of the image to be displayed.
nPalEntries  Gives the number of palette entries to use in displaying the image: 2, 4,
8, 16, 32, 64, 128, or 236.

Example

' Create an image and do some operations on it. Don't let the
' user see the image until all the operations are done.
MM.SetDisplayImagesWhenCreated FALSE
Dim im As Long
MM.CreateImage 512, 480, 8, "temp", im

' Do some operations on 'im'
' Cause 'im' to appear on the screen
MM.ShowImage im, 236

' Reset state so images are once again displayed as soon as
' they are created
MM.SetDisplayImagesWhenCreated TRUE

See also: GetCurrentImage, GetImage
4.3 Finding Loaded Images

Introduction

The next set of functions are crucial for control of the procedures being carried out by your user program. These functions are used for finding an image’s handle, the “indexing tag” that both Visual Basic and MetaMorph use to keep track of the image, as well as for determining how many images are loaded and whether a handle is currently in use. The GetCurrentImage and GetImage functions, which obtain an image’s handle, are particularly important, and you will see these two functions referenced frequently in the “See Also” lists throughout this manual.

GetCurrentImage

Description

Returns the handle of the currently active image.

Syntax

GetCurrentImage(hRetImage As Long) As Long

Remarks

A handle is a unique number used by MetaMorph to keep track of the appropriate image while working with it. This number is used extensively as a variable by most of the MetaMorph Visual Basic functions in this manual. The current (active) image is the one at the front of all the loaded images.

Return values

hRetImage  Returns the handle of the current image.

Example

' Get the current desktop image and put it in 'im'
Dim im As Long
MM.GetCurrentImage  im

See also:

GetImage

GetImage

Description

All images loaded in MetaMorph have a unique index number, from 0 to \((n-1)\), where \(n\) is the number of loaded images. GetImage returns the handle of the image denoted by nIndex.

Syntax

GetImage(nIndex As Integer, hRetImage As Long) As Long

Remarks

The image handle is returned in nRetImage. If an error occurs (for example, if nIndex is not a valid number), nRetImage will contain 0.

Parameters

nIndex  Gives the index number of the desired image. (See Description.) If an error occurs (for example, if nIndex is not a valid number), MetaMorph will return a handle of “0”.

Continued on next page
4.3 Finding Loaded Images, continued

GetImage (continued)

Return values  

\texttt{hRetImage} Returns the handle of the specified image. If an error occurs (for example, if \texttt{hRetImage} is not a valid number), MetaMorph will return a handle of “0”.

Example  

' Print out the names of all loaded images in the message window  
Dim \textit{im} As Long  
Dim \textit{i} As Integer  
Dim \textit{name} As String  

For \textit{i} = 0 To \textit{MM.GetNumberOfImages}  
    \textit{MM.GetImage} \textit{i}, \textit{im}  
    \textit{MM.GetImageName} \textit{im}, \textit{name}  
    \textit{MM.PrintMsg} \textit{name}  
Next \textit{i}

See also: \texttt{GetCurrentImage}

GetNumberOfImages

Description  

Returns the number of images loaded in MetaMorph.

Syntax  

\textit{GetNumberOfImages}() As Long

Remarks  

A value of 0 will be returned if there are no images or if an error occurred.

Return values  

This function returns a value corresponding to the number of images that are currently loaded. If an error has occurred, or if there are no images currently loaded, a value of 0 will be returned.

Example  

' Print out the names of all loaded images in the message window  
Dim \textit{im} As Long  
Dim \textit{i} As Integer  
Dim \textit{name} As String  

For \textit{i} = 0 To \textit{MM.GetNumberOfImages}  
    \textit{MM.GetImage} \textit{i}, \textit{im}  
    \textit{MM.GetImageName} \textit{im}, \textit{name}  
    \textit{MM.PrintMsg} \textit{name}  
Next \textit{i}
4.3 Finding Loaded Images, continued

IsValidImage

Description Tests whether or not the image handle you pass is valid.

Syntax `IsValidImage(hImage As Long) As Long`

Parameters  
  `hImage` Gives the handle of the image being tested. If the handle value that you pass is not valid, `IsValidImage` will return a value of “0”.

Return values `hImage` Returns a value of 0 if the handle you pass is not valid.

Example

```
' Determine if the image handle 'im' denotes a valid image.
' 'im' is an image handle that was initialized previously.
If MM.IsValidImage(im) Then
    MM.PrintMsg "image is valid"
Else
    MM.PrintMsg "Image is not valid"
End If
```

See also: `GetCurrentImage`, `GetImage`
4.4 Manipulating Image Windows

Introduction

This section describes the functions that are used for controlling the appearance of your image windows. The following functions are used for changing the size and position of image windows, as well as minimizing (shrinking to a desktop icon) and maximizing (restoring to full size).

GetImageWindowPosition

Description

Obtains the on-screen position of the specified image.

Syntax

GetImageWindowPosition(hImage As Long, nRetXPos As Integer, nRetYPos As Integer) As Long

Parameters

hImage  Specifies the handle of the image whose position you want to know.

Return values

nRetXPos  Gives the X-coordinate of the upper left corner of the image.
nRetYPos  Gives the Y-coordinate of the upper left corner of the image.

Example

' Print the size and position of the window holding the current image
Dim x As Integer, y As Integer, dx As Integer, dy As Integer
Dim im As Long

MM.GetCurrentImage im
MM.GetImageWindowSize im, dx, dy
MM.GetImageWindowPosition im, x, y
MM.PrintMsg "The current image is at " + Str(x) + ", " + Str(y)
MM.PrintMsg "The size of the current image is " + Str(dx) + 
" X " + Str(dy)

See also:  GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetImageWindowSize, SetImageWindowPosition

GetImageWindowSize

Description

Obtains the width and height of the specified image.

Syntax

GetImageWindowSize(hImage As Long, nRetXSize As Integer, nRetYSize As Integer) As Long

Parameters

hImage  Specifies the handle of the image whose size you want to know.

Continued on next page
4.4 Manipulating Image Windows, continued

GetImageWindowSize (continued)

Return values

$nRetXSize$  Gives the width, in pixels, of the image.

$nRetYSize$  Gives the height, in pixels, of the image.

Example

' Print the size and position of the window holding the current image

Dim $x$ As Integer, $y$ As Integer, $dx$ As Integer, $dy$ As Integer
Dim im As Long

MM.GetCurrentImage im
MM.GetImageWindowSize im, $dx$, $dy$
MM.GetImageWindowPosition im, $x$, $y$
MM.PrintMsg "The current image is at " + Str($x$) + ", " + Str($y$)
MM.PrintMsg "The size of the current image is " + Str($dx$) + ", " + Str($dy$)

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetImageWindowPosition, SetImageWindowSize

MaximizeImageWindow

Description

Expands the window of the specified image to full-size. If the image has been minimized (shrunk to a desktop icon), it will expand the window to its normal size.

Syntax

MaximizeImageWindow(hImage As Long) As Long

Parameters

$hImage$  Specifies the handle of the image you want to maximize.

Example

' Maximize the current image
Dim im As Long
MM.GetCurrentImage im
MM.MaximizeImageWindow im

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), MinimizeImageWindow

Continued on next page
4.4 Manipulating Image Windows, continued

MinimizeImageWindow

Description
Shrinks the window of the specified image to a desktop icon.

Syntax
MinimizeImageWindow(hImage As Long) As Long

Parameters
hImage  Specifies the handle of the image you want to minimize.

Example
' Minimize the current image
Dim im As Long
MM.GetCurrentImage im
MM.MinimizeImageWindow im

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), MaximizeImageWindow

SetImageWindowPosition

Description
Sets the position of the window for the specified image.

Syntax
SetImageWindowPosition(hImage As Long, nXPos As Integer, nYPos As Integer)

Remarks
This function dictates the position of the upper left corner of the image window.

Parameters
hImage  Specifies the handle of the image whose position you want to set.
nXPos  Specifies the X-coordinate of the upper left corner of the image window.
nYPos  Specifies the Y-coordinate of the upper left corner of the image window.

Example
' Make the window holding the current image 100 X 100 and put
' it in the upper left corner of the screen
Dim im As Long
MM.GetCurrentImage im
MM.SetImageWindowPosition im, 0, 0
MM.SetImageWindowSize im, 100, 100

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetImageWindowSize

Continued on next page
4.4 Manipulating Image Windows, continued

SetImageWindowSize

**Description**
Sets the size of the window for the specified image.

**Syntax**
```
SetImageWindowSize(hImage As Long, nXSize As Integer, nYSize As Integer) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image whose size you want to set.
- `nXSize` Specifies the width, in pixels, of the image.
- `nYSize` Specifies the height, in pixels, of the image.

**Example**
```
' Make the window holding the current image 100 X 100 and put
' it in the upper left corner of the screen
Dim im As Long
MM.GetCurrentImage im
MM.SetImageWindowPosition im, 0, 0
MM.SetImageWindowSize im, 100, 100
```

**See also:**
- `GetCurrentImage` (Section 4.3),
- `GetImage` (Section 4.3),
- `SetImageWindowPosition`
4.5 Reading and Using Image Properties

Introduction

It is frequently necessary or useful to change certain characteristics about an image, such as its dimensions, bit-depth, or name. The functions that follow can be used to read or configure such image properties, as well as to read or create an annotation, change zoom factor, or switch to a different plane in a stack. One programming example is given for all of the image “Get” functions (see Figure 4.1 on page 56).

GetActivePlane

Description

Obtains the number of the currently active plane in the specified image.

Syntax

GetActivePlane(hImage Long, nRetPlaneNumber As Integer) As Long

Parameters

hImage Specifies the handle of the image in which the plane resides.

Return values

nRetPlaneNumber Returns the number of the active plane.

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetNumberOfPlanes, SetActivePlane

GetDepth

Description

Obtains the bit-depth of a specified image.

Syntax

GetDepth(hImage As Long, nRetDepth As Integer) As Long

Parameters

hImage Specifies the handle of the image.

Return values

nRetDepth Returns the bit-depth—1, 8, 16, or 24.

See also: GetCurrentImage (Section 4.3), GetHeight, GetImage (Section 4.3), GetWidth

GetHeight

Description

Obtains the height (Y-axis size) of a specified image. (Note: This may differ from the height of the image window.)

Syntax

GetHeight(hImage As Long, nRetHeight As Integer) As Long

Parameters

hImage Specifies the handle of the image.
4.5 Reading and Using Image Properties, continued

GetHeight (continued)

Return values

\( nRetHeight \) Returns the Y-axis size, in pixels, of the image.

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetImageWindowSize, GetWidth

GetImageAnnotation

Description

Obtains the annotation of the specified image.

Syntax

\[
\text{GetImageAnnotation}(hImage \text{ As Long, planeNumber As Integer, sRetAnnotation As String}) \text{ As Long}
\]

Remarks

Returns the annotation of the given plane of the given image. The annotation is returned in \( sRetAnnotation \).

Parameters

\( hImage \) Specifies the handle of the image.

\( planeNumber \) Specifies the number of the plane in a stack whose annotation you want to obtain. If the image is a single plane, this variable must be 0.

Return values

\( sRetAnnotation \) Returns the annotation text.

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetImageAnnotation

GetImageName

Description

Obtains the name of a specified image.

Syntax

\[
\text{GetImageName}(hImage \text{ As Long, sRetName As String}) \text{ As Long}
\]

Parameters

\( hImage \) Specifies the handle of the image.

\( sRetName \) Returns the name of the image.

Return values

\( sRetName \) Returns the name of the image.

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetImageName

Continued on next page
### 4.5 Reading and Using Image Properties, continued

#### GetNumberOfPlanes

**Description**
Obtains the number of planes in a specified image or stack of images.

**Syntax**
```
GetNumberOfPlanes(hImage As Long, nRetNumberOfPlanes As Integer) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image.

**Return values**
- `nRetNumberOfPlanes` Returns the number of planes

**See also:**
- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetActivePlane`

#### GetWidth

**Description**
Obtains the width (X-axis size) of the specified image. (Note: This may differ from the width of the image window.)

**Syntax**
```
GetWidth(hImage As Long, nRetWidth As Integer) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image.

**Return values**
- `nRetWidth` Returns the X-axis size, in pixels, of the image.

**See also:**
- `GetCurrentImage` (Section 4.3), `GetHeight`, `GetImage` (Section 4.3), `GetImageWindowSize`

#### GetZoom

**Description**
Obtains the zoom factor level of the specified image.

**Syntax**
```
GetZoom(hImage As Long, nRetZoom As Integer) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image.

**Return values**
- `nRetZoom` Returns the zoom level, expressed as a percentage of the full size (100 = normal size).

**See also:**
- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetZoom`

Continued on next page
Figure 4.1  Image “Get” Function Programming Example

' Print information about the current image
Dim im As Long
MM.GetCurrentImage im

Dim name As String
MM.GetImageName im, name

Dim w As Integer, h As Integer, d As Integer
MM.GetWidth im, w
MM.GetHeight im, h
MM.GetDepth im, d

MM.PrintMsg "The image " + name + " is " + Str(w) + " X " + Str(h)
MM.PrintMsg "and has a depth of " + Str(d)

Dim nPlanes As Integer
Dim curPlane As Integer
MM.GetNumberOfPlanes im, nPlanes
MM.GetActivePlane im, curPlane
MM.PrintMsg "It has " + Str(nPlanes) + " planes, and the current plane is " + Str(curPlane)

Dim ann As String
MM.GetImageAnnotation im, curPlane, ann
MM.PrintMsg "The annotation of the current plane is " + ann

Dim z As Integer
MM.GetZoom im, z
MM.PrintMsg "and the zoom is " + Str(z)

Continued on next page
4.5 Reading and Using Image Properties, continued

SetActivePlane

Description
Selects a plane in a specified image stack and makes it the active plane.

Syntax
SetActivePlane(hImage As Long, planeNumber As Integer) As Long

Remarks
Planes are numbered starting at 0.

Parameters
hImage  Specifies the handle of the image.
planeNumber  Specifies the number of the plane to be made active.

Example
' Set the active plane of the current image to 3 if it has
' that many planes
Dim im As Long
MM.GetCurrentImage im
Dim nPlanes As Integer
MM.GetNumberOfPlanes im, nPlanes
If nPlanes > 3 Then
    MM.SetActivePlane im, 3
Else
    PrintMsg "Error! Image only has " + Str(nPlanes) + " planes"
End If

See also:  GetActivePlane, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

SetImageAnnotation

Description
Assigns an annotation to an image or stack plane.

Syntax
SetImageAnnotation(hImage As Long, planeNumber As Integer, annotation As String) As Long

Parameters
hImage  Specifies the handle of the image.
planeNumber  Specifies the number of the plane to be made active.
annotation  Provides the annotation text to be assigned to the selected image or plane.

Example
' Set the annotation of plane 0 of the current image
Dim im As Long
MM.GetCurrentImage im
MM.SetActiveAnnotation im, 0, "This is the first plane"

See also:  GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetImageAnnotation
4.5  Reading and Using Image Properties, continued

SetImageName

Description  Assigns a new name to a selected image.

Syntax  

\[ \text{SetImageName}(hImage \text{ As Long}, \text{name As Long}) \text{ As Long} \]

Parameters  

- \( hImage \)  Specifies the handle of the image.
- \( \text{name} \)  Specifies the name for the image.

Example  

' Set the name of the current image to "Fred"
Dim im As Long
MM.GetCurrentImage im
MM.SetImageName im, "Fred"

See also:  GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetImageName

SetImageTimestamp

Description  Sets the timestamp (either creation time or last saved time) for a given image to the specified date and time.

Syntax  

\[ \text{SetImageTimestamp}(hImage \text{ As Long}, \text{nStampType As Integer}, \text{nMonth As Integer}, \text{nDay As Integer}, \text{nYear As Integer}, \text{nHours As Integer}, \text{nMinutes As Integer}, \text{nSeconds As Integer}, \text{nMilliseconds As Integer}) \]

Parameters  

- \( hImage \)  Specifies the handle of the image.
- \( \text{nStampType} \)  Specifies which timestamp will be set, 0 for creation time and non-0 for last saved time.
- \( \text{nMonth} \)  Specifies the month, with 1 signifying January, 6 signifying June, etc.
- \( \text{nDay} \)  Specifies the date, from 1 to 31.
- \( \text{nYear} \)  Specifies the year, using four digits.
- \( \text{nHours} \)  Specifies the hour, from 0 to 23, counting forward from midnight.
- \( \text{nMinutes} \)  Specifies the minute, from 0 to 59.
- \( \text{nSeconds} \)  Specifies the seconds, from 0 to 59.
- \( \text{nMilliseconds} \)  Specifies the milliseconds, from 0 to 999.

Continued on next page
4.5  Reading and Using Image Properties, continued

SetImageTimestamp
(continued)

Example

' Set the creation timestamp of the current image to 2:33:20.68

Dim im As Long

MM.GetCurrentImage im

MM.SetImageTimestamp im, 0, 11, 3, 2001, 14, 33, 20, 680

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3)

SetZoom

Description
Sets the zoom factor of a specified image.

Syntax

SetZoom(hImage As Long, zoom As Integer, nXCenter As Integer, nYCenter As Integer) As Long

Remarks
Full size is assigned a numeric value of 100. Numbers smaller than 100 will make the
image smaller, and those greater than 100 will make the image larger. If the image
becomes larger than its display window, nXCenter and nYCenter give the pixel
coordinates at which to center the image.

Parameters

hImage    Specifies the handle of the image.
zoom     Specifies the zoom level, as a percent of the full-sized image.
nXCenter  Specifies the X-coordinate for the pixel at which the zoomed image is to
          be centered.
nYCenter  Specifies the Y-coordinate for the pixel at which the zoomed image is to
          be centered.

Example

' Set the zoom of the current image to 200 and center the image
' at 100, 100

Dim im As Long

MM.GetCurrentImage im

MM.SetZoom im, 200, 100, 100

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetZoom
Chapter 5 – Adjusting Image Display

5.1 Overview

Introduction

Image display can be modified in a number of ways—you can increase image contrast, you can brighten or darken the image, you can adjust the grayscale or color resolution by altering the number of entries in the image’s palette, or you can modify the values in the image’s look-up table. This chapter covers the functions you will need to use to accomplish these tasks.

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating the Image After Changing the Display</td>
<td>61</td>
</tr>
<tr>
<td>Adjusting Brightness and Contrast</td>
<td>62</td>
</tr>
<tr>
<td>Autoscaling 16-Bit Images</td>
<td>66</td>
</tr>
<tr>
<td>Working with Look-up Tables and Palettes</td>
<td>71</td>
</tr>
</tbody>
</table>
5.2 Updating the Image After Changing the Display

Introduction
The first function in this chapter is of such vital importance that we feel it warrants a separate section of its own. Any time you execute a function that makes a change to an image, such as when you create a region of interest, alter the brightness or contrast, switch look-up tables, or assign a set of intensity values to a row or column of pixels, you will need to update the image display. The **UpdateDisplay** function carries out this operation.

**UpdateDisplay**

**Description**
Redraws any parts of the image that need updating. This function should be called after pixel modification operations to cause those changes to be drawn.

**Syntax**
```
UpdateDisplay(hImage As Long) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image to be updated.

**Example**
```
' Draw a white box on the current image and then update the display so the changed pixels are shown
Dim i As Integer
Dim im As Long
Dim data(20) As Byte
For i = 1 To 20
    data(i) = 255
Next i
MM.GetCurrentImage im
For i = 1 To 20
    MM.WriteRow im, 0, i, 20, 8, data
Next i
MM.UpdateDisplay im
```

**See also:**
- **GetCurrentImage** (Section 4.3), **GetImage** (Section 4.3)
5.3 Adjusting Brightness and Contrast

Introduction

The image display adjustments you are most often likely to perform are those which modify an image’s brightness and contrast. Brightness settings can range from 0 to 100, with 50 representing the brightness value of the original, unaltered image. Contrast settings range from 50 to 100. (MetaMorph can only *increase* image contrast from the original.) The following functions are used for reading the current settings and changing them to suit your needs. If you need to adjust the brightness or contrast of a 16-bit image, you will need to use the functions described in Section 5.4.

AutoEnhance

Description

Automatically enhances the brightness and contrast of a specified image.

Syntax

```
AutoEnhance(hImage As Long) As Long
```

Parameters

- `hImage` Specifies the handle of the image to be autoenhanced.

Example

```
' Autoenhance the image contrast then fix it
Dim im As Long
MM.GetCurrentImage im
MM.AutoEnhance im
MM.FixImage im
```

See also:

- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetBrightness`, `SetContrast`

FixImage

Description

Makes changes to the brightness and/or contrast of an image permanent.

Syntax

```
FixImage(hImage As Long) As Long
```

Parameters

- `hImage` Specifies the handle of the image in which brightness and/or contrast changes are to be made permanent.

Example

```
' Autoenhance the image contrast then fix it
Dim im As Long
MM.GetCurrentImage im
MM.AutoEnhance im
MM.FixImage im
```

See also:

- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetBrightness`, `SetContrast`

Continued on next page
5.3 Adjusting Brightness and Contrast, continued

GetBrightness

Description
Obtains the brightness setting of a specified image.

Syntax
GetBrightness(hImage As Long, nRetBrightness As Double) As Long

Parameters
hImage Specifies the handle of the image whose brightness setting you want to obtain.

Example
' Double the brightness and contrast of the current image
Dim im As Long
Dim bright As Double
Dim contrast As Long
MM.GetCurrentImage im
MM.GetBrightness im, bright
MM.GetContrast im, contrast
MM.SetBrightness im, bright * 2
MM.SetContrast im, contrast * 2

Return values
nRetBrightness Returns the brightness setting value.

See also:
GetContrast, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

GetContrast

Description
Obtains the contrast setting of a specified image.

Syntax
GetContrast(hImage As Long, nRetContrast As Double) As Long

Parameters
hImage Specifies the handle of the image whose contrast setting you want to obtain.

Return values
nRetContrast Returns the contrast setting value.

Example
' Double the brightness and contrast of the current image
Dim im As Long
Dim bright As Double
Dim contrast As Long
MM.GetCurrentImage im
MM.GetBrightness im, bright
MM.GetContrast im, contrast
MM.SetBrightness im, bright * 2
MM.SetContrast im, contrast * 2

See also:
GetBrightness, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

Continued on next page
5.3 Adjusting Brightness and Contrast, continued

ResetContrast

Description
Sets the contrast and brightness of a specified image to the settings it had when last saved or “fixed.”

Syntax
ResetContrast(hImage As Long) As Long

Parameters
hImage Specifies the handle of the image whose brightness and/or contrast settings you want to reset.

Example
' Reset the contrast of the current image
Dim im As Long
MM.GetCurrentImage im
MM.ResetContrast im

See also:
FixImage, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

SetBrightness

Description
Sets the brightness of an image to a specified intensity value.

Syntax
SetBrightness(hImage As Long, brightness As Double) As Long

Parameters
hImage Specifies the handle of the image whose brightness you want to set.
brightness Specifies a new intensity setting.

Example
' Double the brightness and contrast of the current image
Dim im As Long
Dim bright As Double
Dim contrast As Long
MM.GetCurrentImage im
MM.GetBrightness im, bright
MM.GetContrast im, contrast
MM.SetBrightness im, bright * 2
MM.SetContrast im, contrast * 2

See also:
AutoEnhance, FixImage, GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetContrast

Continued on next page
### 5.3 Adjusting Brightness and Contrast, continued

**SetContrast**

**Description**
Sets the contrast of an image to a specified setting.

**Syntax**

```vba
SetContrast(hImage As Long, contrast As Double) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image whose contrast you want to set.
- `contrast` Specifies a new contrast setting.

**Example**

```vba
' Double the brightness and contrast of the current image
Dim im As Long
Dim bright As Double
Dim contrast As Long
MM.GetCurrentImage im
MM.GetBrightness im, bright
MM.GetContrast im, contrast
MM.SetBrightness im, bright * 2
MM.SetContrast im, contrast * 2
```

**See also:**

- AutoEnhance, FixImage, GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetBrightness
5.4 Autoscaling 16-Bit Images

Introduction
Sixteen-bit images require a different approach when it comes to adjusting brightness and contrast. Because of the greater range of intensity values that are involved, autoscaling a 16-bit image relies on a different set of functions from those described in the preceding section.

GetAutoScale

Description
Obtains the current autoscaling state of a specified 16-bit image.

Syntax
GetAutoScale(hImage As Long, bRetOnOff As Boolean) As Long

Parameters
hImage Specifies the handle of the image whose autoscaling state you want to obtain.

Return values
bRetOnOff Returns the current autoscaling state. A value of TRUE means it is currently enabled, and FALSE means it is currently disabled.

Example
' Get the autoscale properties of the current image

Dim im As Long
Dim onoff As Boolean
Dim min As Integer
Dim max As Integer

MM.GetCurrentImage im
MM.GetAutoScale im, onoff
MM.GetMinScale im, min
MM.GetMaxScale im, max

MM.PrintMsg "The min and max scale of the current image is " + Str(min) + " and " + Str(max)
If onoff = TRUE Then
    MM.PrintMsg "The autoscaling of the current image is ON"
Else
    MM.PrintMsg "The autoscaling of the current image is OFF"
End If

See also:
GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetAutoScale

Continued on next page
### GetMaxScale

**Description**
Obtains the maximum scaling value of a specified 16-bit image.

**Syntax**

```
GetMaxScale(hImage As Long, nRetMaxScale As Integer) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image whose maximum scaling value you want to obtain.

**Return values**

- `nRetMaxScale` Returns the maximum scaling grayscale value.

**Example**

```
' Get the autoscale properties of the current image

Dim im As Long
Dim onoff As Boolean
Dim min As Integer
Dim max As Integer

MM.GetCurrentImage im
MM.GetAutoScale im, onoff
MM.GetMinScale im, min
MM.GetMaxScale im, max

MM.PrintMsg "The min and max scale of the current image is " + Str(min) + " and " + Str(max)
If onoff = TRUE Then
    MM.PrintMsg "The autoscaling of the current image is ON"
Else
    MM.PrintMsg "The autoscaling of the current image is OFF"
End If
```

**See also:**

- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `GetMinScale`
5.4 Autoscaling 16-Bit Images, continued

GetMinScale

Description
Obtains the minimum scaling value of a specified 16-bit image.

Syntax
GetMinScale(hImage As Long, nRetMinScale As Integer) As Long

Parameters
hImage Specifies the handle of the image whose minimum scaling value you want to obtain.

Return values
nRetMinScale Returns the minimum scaling grayscale value.

Example
' Get the autoscale properties of the current image
Dim im As Long
Dim onoff As Boolean
Dim min As Integer
Dim max As Integer
MM.GetCurrentImage im
MM.GetAutoScale im, onoff
MM.GetMinScale im, min
MM.GetMaxScale im, max

MM.PrintMsg "The min and max scale of the current image is "
   + Str(min) + " and " + Str(max)
If onoff = TRUE Then
   MM.PrintMsg "The autoscaling of the current image is ON"
Else
   MM.PrintMsg "The autoscaling of the current image is OFF"
End If

See also:
GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetMaxScale

SetAutoScale

Description
Enables or disables autoscaling for the specified 16-bit image.

Syntax
SetAutoScale(hImage as Long, bOnOff As Boolean) As Long

Parameters
hImage Specifies the handle of the image for which you want to enable or disable autoscaling.

bOnOff Sets the scaling state. If you set bOnOff to TRUE, autoscaling will be enabled. To disable autoscaling, set bOnOff to FALSE.
5.4 Autoscaling 16-Bit Images, continued

SetAutoScale (continued)

Example

' Enable autoscaling for the current image and set the ' range from 0 to 4095
Dim im As Long
MM.GetCurrentImage im
MM.SetAutoScale im, TRUE
MM.SetMinScale im, 0
MM.SetMaxScale im, 4095

See also: GetAutoScale, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

SetMaxScale

Description
Sets the maximum value for the grayscale range being used to autoscale a specified 16-bit image.

Syntax
SetMaxScale(hImage As Long, nMaxScale As Integer) As Long

Parameters
hImage  Specifies the handle of the image for which you want to set the maximum autoscaling value.
nMaxScale  Defines the maximum grayscale value for the autoscaling range.

Example

' Enable autoscaling for the current image and set the ' range from 0 to 4095
Dim im As Long
MM.GetCurrentImage im
MM.SetAutoScale im, TRUE
MM.SetMinScale im, 0
MM.SetMaxScale im, 4095

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetAutoScale, SetMinScale

Continued on next page
5.4 Autoscaling 16-Bit Images, continued

SetMinScale

Description
Sets the minimum value for the grayscale range being used to autoscale a specified 16-bit image.

Syntax
SetMinScale(hImage As Long, nMinScale As Integer) As Long

Parameters
hImage Specifies the handle of the image for which you want to set the minimum autoscaling value.
nMinScale Defines the minimum grayscale value for the autoscaling range.

Example
' Enable autoscaling for the current image and set the range from 0 to 4095
Dim im As Long
MM.GetCurrentImage im
MM.SetAutoScale im, TRUE
MM.SetMinScale im, 0
MM.SetMaxScale im, 4095

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetAutoScale, SetMaxScale
5.5 Working with Look-up Tables and Palettes

Introduction

A look-up table, or LUT, is a “table” of values that translates each original grayscale value in an image to another intensity or to a color. This allows you to change the display of an image without affecting its data. The functions in this section can be used for making changes to the current LUT or for switching to another LUT. In addition, you can change the number of grayscale levels or colors that are displayed by changing the number of palette entries with the SetNumPaletteEntries function.

GetLut – for Visual Basic 6 & earlier


Description

Reads the elements from the given look-up table (LUT) into the red, green, and blue arrays that are passed.

Syntax

GetLut(hImage As Long, lutNumber As Integer, startElement As Integer, nElements As Integer, red() As Byte, green() As Byte, blue() As Byte) As Long
GetLutEx2(hImage As Long, lutNumber As Integer, startElement As Integer, nElements As Integer, red() As Byte, green() As Byte, blue() As Byte) As Long

Remarks

The LUT that you want to read is specified by lutNumber. Use one of the following constants:

0 = Monochrome
1 = Pseudocolor
2 = Red
3 = Green
4 = Blue

or use a user LUT number from 5 to 15.

Parameters

hImage Specifies the handle of the image for which you want to read the LUT values.

lutNumber Specifies the LUT to be read. (See Remarks.)

startElement Indicates the number of the element in the LUT where reading is to start.

nElements Specifies the number of LUT elements to read. This will determine the size of the red(), green(), and blue() arrays (see Return values).

Return values

red() Passes the values of the elements from the red channel of the LUT. The size of this array will be determined by the number of elements in the look-up table.

green() Passes the values of the elements from the green channel of the LUT. The size of this array will be determined by the number of elements in the look-up table.

blue() Passes the values of the elements from the blue channel of the LUT. The size of this array will be determined by the number of elements in the look-up table.

Continued on next page
### GetLut (continued)

**Example**

```
' Read the first user lut of the current image into memory.
' Since LUTs 0 through 4 are the fixed LUTs, the first user
' LUT is LUT number 5
Dim im As Long
Dim r(256) As Byte
Dim g(256) As Byte
Dim b(256) As Byte
MM.GetCurrentImage im
MM.GetLut im, 5, 0, 256, r, g, b
```

**See also:**
- GetCurrentImage (Section 4.3)
- GetImage (Section 4.3)
- GetLutModel
- SetLut

### GetLutModel

**Description**

Obtains a value that indicates the look-up table (LUT) currently being used to display the given image.

**Syntax**

```
GetLutModel(hImage As Long, nRetLutModel As Integer) As Long
```

**Remarks**

The LUT that is in use is indicated by a numeric value. The values returned in `nRetLutModel` are:

- 0 = Monochrome
- 1 = Pseudocolor
- 2 = Red
- 3 = Green
- 4 = Blue

or a user-defined LUT, with a number between 5 and 15.

**Parameters**

- `hImage` Specifies the handle of the image for which you want to determine the LUT.

**Return values**

- `nRetLutModel` Returns the LUT value. (See Remarks.)

*Continued on next page*
5.5 Working with Look-up Tables and Palettes, continued

GetLutModel
(continued)

Example

' Determine which LUT is currently being used on the current ' image

Dim im As Long
Dim lutnum As Integer

MM.GetCurrentImage im
MM.GetLutModel im, lutnum

If lutnum = 0 Then
    MM.PrintMsg "LUT is Monochrome"
ElseIf lutnum = 1 Then
    MM.PrintMsg "LUT is Pseudocolor"
ElseIf lutnum = 2 Then
    MM.PrintMsg "LUT is Red"
ElseIf lutnum = 3 Then
    MM.PrintMsg "LUT is Green"
ElseIf lutnum = 4 Then
    MM.PrintMsg "LUT is Blue"
Else
    MM.PrintMsg "LUT is user LUT number " + Str(lutnum = 5)
End If

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetLut, SetLutModel

Continued on next page
5.5 Working with Look-up Tables and Palettes, continued

SetLut

Description
Writes the contents of the red, green, and blue arrays passed into the elements of the
given look-up table (LUT).

Syntax

\[
\text{SetLut}(\text{hImage} \text{ As Long, lutNumber As Integer, startElement As Integer, nElements As Integer, red()} \text{ As Byte, green()} \text{ As Byte, blue()} \text{ As Byte}) \text{ As Long}
\]

Parameters

- \(hImage\) Specifies the handle of the image for which you want to set the LUT.
- \(lutNumber\) Denotes the user-defined LUT to which you want to write (5 to 15). You
cannot write to the predefined LUTs (Monochrome, etc).
- \(startElement\) Indicates the number of the element in the LUT where writing is to
start.
- \(nElements\) Specifies the number of LUT elements to write.
- \(red()\) Indicates the array for the red channel of the LUT. LUTs have a maximum
possible total of 256 elements. The red array must contain at least \(nElements\)
elements.
- \(green()\) Indicates the array for the green channel of the LUT. LUTs have a maximum
possible total of 256 elements. The green array must contain at least \(nElements\)
elements.
- \(blue()\) Indicates the array for the blue channel of the LUT. LUTs have a maximum
possible total of 256 elements. The blue array must contain at least \(nElements\)
elements.

Example

' Configure user LUT 0 on the current image as an inverted
' contrast monochrome LUT and then use it

Dim im As Long
Dim i As Integer
Dim lut(256) As Byte

For i = 0 To 255
    \(lcut(i) = 255 - i\)
Next i

\text{MM.GetCurrentImage} \text{ im}
\text{MM.SetLut} \text{ im, 5, 0, 256, lut, lut, lut}
\text{MM.SetLutModel} \text{ im, 5}

See also: \text{GetCurrentImage} (Section 4.3), \text{GetImage} (Section 4.3), \text{SetLutModel},
\text{SetNumPaletteEntries}

Continued on next page
5.5 Working with Look-up Tables and Palettes, continued

SetLutModel

**Description**
Specifies a look-up table (LUT) to use to display the given image.

**Syntax**
```
SetLutModel(hImage As Long, nLutModel As Integer) As Long
```

**Remarks**
The model of LUT to use is specified with `nLutModel`. Values you can use are

- 0 = Monochrome
- 1 = Pseudocolor
- 2 = Red
- 3 = Green
- 4 = Blue

or a user-defined LUT, with a number between 5 and 15.

**Parameters**
- `hImage` Specifies the handle of the image for which you want to set the LUT model.
- `nLutModel` Specifies the value of the LUT you want to use. (See Remarks.)

**Example**
```
' Configure user LUT 0 on the current image as an inverted
' contrast monochrome LUT and then use it

Dim im As Long
Dim i As Integer
Dim lut(256) As Byte

For i = 0 To 255
    lut(i) = 255 - i
Next i

MM.GetCurrentImage im
MM.SetLut im, 5, 0, 256, lut, lut, lut
MM.SetLutModel im, 5
```

See also:
- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetLut`,
- `SetNumPaletteEntries`

*Continued on next page*
5.5 Working with Look-up Tables and Palettes, continued

SetNumPaletteEntries

Description
Specifies the number of palette entries to use for displaying an image.

Syntax
SetNumPaletteEntries(hImage As Long, newNumEntries As Integer) As Long

Remarks
The maximum possible number of palette entries is 236. This is because the Windows operating system reserves 20 palette colors for use by such interface elements as dialog box buttons, title bars, and the like.

Parameters
hImage Specifies the handle of the image for which you want to set the number of palette entries.
newNumEntries Specifies the number of palette entries: 2, 4, 8, 16, 32, 64, 128, or 236.

Example
' Set the number of palette entries on the current image to 8.
' This will make the image appear as if it is quantized.
Dim im As Long
MM.GetCurrentImage im
MM.SetNumPaletteEntries im, 8

See also:
GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetLut, SetLutModel
Chapter 6 – Reading and Using Image Pixel Data

6.1 Overview

Introduction

Digital images are constructed of individual points, or pixels, each of which has a specific, quantifiable intensity value or color. Because each pixel can be represented in memory by a set of numeric values that encode its location (X and Y coordinates) and brightness or color (grayscale or color value), a variety of manipulations can be applied that analyze and process the information contained in an image. This chapter deals with the functions that read and manipulate image pixel data.

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applying Thresholding</td>
<td>78</td>
</tr>
<tr>
<td>Reading and Manipulating Image Data</td>
<td>82</td>
</tr>
</tbody>
</table>
6.2 Applying Thresholding

Introduction

Most procedures in image analysis require that a distinction be made between the objects to be measured and the rest of the image. MetaMorph makes this distinction on the basis of the intensity values of pixels in the object vs. those in the background regions. This procedure, which is called **segmentation or thresholding**, involves defining a range of intensity values that will be considered as belonging to the objects being measured. A threshold range can be inclusive or exclusive. An **inclusive** threshold is one which considers all grayscale values between the upper and lower limits of the threshold range as belonging to the objects being measured. Conversely, an **exclusive** threshold considers all grayscale values that are equal to, or outside of, the upper and lower limits as belonging to the objects being measured. The four functions described in this section are used for determining the current threshold settings and for defining a new set of threshold settings.

GetThresholdRange

**Description**

Obtains the current threshold range (maximum and minimum grayscale settings) of the specified image.

**Syntax**

```
GetThresholdRange(hImage As Long, nRetLow As Integer, nRetHigh As Integer) As Long
```

**Parameters**

- **hImage**  Specifies the handle of the image whose threshold range you want to obtain.
- **nRetLow** Returns the grayscale value of the lower threshold limit.
- **nRetHigh** Returns the grayscale value of the upper threshold limit.

**Example**

```vbnet
' Get the threshold range and state of the current image
Dim im As Long
Dim low As Integer, high As Integer, state As Integer
MM.GetCurrentImage im
MM.GetThresholdRange im, low, high
MM.GetThresholdState im, state
MM.PrintMsg "The low and high threshold values of the current image are " + Str(low) + " and " + Str(high)
If state = 0 Then
    MM.PrintMsg "Thresholding is currently off"
ElseIf state = 1 Then
    MM.PrintMsg "Thresholding is set to 'On-Inclusive' mode"
ElseIf state = 2 Then
    MM.PrintMsg "Thresholding is set to 'On-Exclusive' mode"
End If
```

**See also:**

- **GetCurrentImage** (Section 4.3), **GetImage** (Section 4.3), **GetThresholdState**, **SetThresholdRange**

*Continued on next page*
## 6.2 Applying Thresholding, continued

### GetThresholdState

**Description**
Obtains the threshold state of the specified image

**Syntax**
```vbnet
GetThresholdState(hImage As Long, nRetState As Integer) As Long
```

**Parameters**
- `hImage` Specifies the handle of the image whose thresholding state you want to obtain.

**Return values**
- `nRetState` Returns the current thresholding state of the given image. If thresholding is disabled, `nRetState` will return a value of 0. If the threshold state is set to On-Inclusive (pixels with grayscale values between the upper and lower threshold limits are included), `nRetState` will be set to 1. If the threshold state is set to On-Exclusive (pixels with grayscale values equal or lower than the lower threshold limit, and those with values equal or higher than the upper limit, are included), `nRetState` will return a value of 2.

**Example**
```vbnet
' Get the threshold range and state of the current image
Dim im As Long
Dim low As Integer, high As Integer, state As Integer
MM.GetCurrentImage im
MM.GetThresholdRange im, low, high
MM.GetThresholdState im, state

MM.PrintMsg "The low and high threshold values of the current image are " + Str(low) + " and " + Str(high)
If state = 0 Then
   MM.PrintMsg "Thresholding is currently off"
ElseIf state = 1 Then
   MM.PrintMsg "Thresholding is set to 'On-Inclusive' mode"
ElseIf state = 2 Then
   MM.PrintMsg "Thresholding is set to 'On-Exclusive' mode"
End If
```

**See also:**
- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `GetThresholdRange`, `SetThresholdState`
6.2 Applying Thresholding, continued

SetThresholdRange
Description
Sets the upper and lower limit of a threshold range for a specified image.

Syntax
SetThresholdRange(hImage As Long, low As Integer, high As Integer) As Long

Parameters
- hImage: Specifies the handle of the image whose thresholding range you want to set.
- low: Specifies the grayscale value of the lower threshold limit.
- high: Specifies the grayscale value of the upper threshold limit.

Example
' Enable thresholding in the current image, and highlight pixels
' between grayscale values 50 and 187
Dim im As Long
MM.GetCurrentImage im
MM.SetThresholdRange im, 50, 187
MM.SetThresholdState im, 1

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetThresholdState

SetThresholdState
Description
Sets the threshold state of the specified image.

Syntax
SetThresholdState(hImage As Long, state As Integer) As Long

Remarks
The threshold state is set with the state variable. Possible values are

0: Disables thresholding (Off).
1: Enables inclusive thresholding, in which all pixels with grayscale values between the high and low values (see SetThresholdRange) are highlighted.
2: Enables exclusive thresholding, in which all pixels with grayscale values equal to or outside the high and low values are highlighted.

Parameters
- hImage: Specifies the handle of the image whose thresholding state you want to set.
- state: Indicates the thresholding state for the image. (See Remarks.)

Continued on next page
6.2  Applying Thresholding, continued

SetThresholdState
(continued)

Example

' Enable thresholding in the current image, and highlight pixels ' between grayscale values 50 and 187
Dim im As Long
MM.GetCurrentImage im
MM.SetThresholdRange im, 50, 187
MM.SetThresholdState im, 1

See also:  GetCurrentImage (Section 4.3), GetImage (Section 4.3), SetThresholdRange
6.3 Reading and Manipulating Image Data

Introduction

Pixel intensities are represented by numeric values in computer memory, and can be easily read and manipulated. This section describes the functions that perform such “read” and “write” procedures. The BinarizeImage and WriteText functions are included here because the process of converting all pixels to black or white and the process of adding a text label to an image are little more than a reassignment of the intensity values of the underlying pixels.

BinarizeImage

Description

Creates a binarized (1-bit) version of a selected image.

Syntax

BinarizeImage(hImage As Long, hDest As Long) As Long

Remarks

This function places a binarized version of hImage in the image given by hDest. hImage should be thresholded before this function is called, or you will get a blank result image.

Parameters

hImage  Specifies the handle of the image you want to binarize.

hDest  Specifies the handle of a destination image that will be overwritten with the binarized version of hImage. The destination image must exist before you can apply this function. Typically, you will create an image with the CreateImage command.

Example

' Binarize the current image and place it in "Binarized"

Dim curIm As Long
Dim binIm As Long

MM.GetCurrentImage curIm
MM.CreateImage 512, 512, 1, "Binarized", binIm

' curIm must be thresholded before it can be binarized
MM.SetThresholdState curIm, 1
MM.SetThresholdRange im, 50, 150

MM.BinarizeImage curIm, binIm

See also:

GetCurrentImage (Section 4.3), CreateImage (Section 4.2), GetImage (Section 4.3)

Continued on next page
ReadColumn

Description  Reads the intensity values of a vertical line of pixels from a selected image and writes them into an array.

Syntax  

```
ReadColumn(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, pixelBuffer() As Variant) As Long
```

Parameters  

- **hImage**  Specifies the handle of the image to be read.
- **xPos**  Specifies the X-coordinate of the pixel where reading should start.
- **yPos**  Specifies the Y-coordinate of the pixel where reading should start.
- **nPixels**  Specifies the number of pixels to be read, starting at the position given by **xPos** and **yPos**. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, Data Types and Arrays.)
- **depth**  Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, Data Types and Arrays.)

- **pixelBuffer()**  Defines a buffer into which the pixel values will be read. If **depth** is 8, **pixelBuffer** should be defined as Byte. If **depth** is 16, **pixelBuffer** should be Integer. If **depth** is 24, **pixelBuffer** should be Byte.

Return values  

**pixelBuffer()**  The pixel values will be read into this predefined buffer. If the source image is 24-bit, the array will be read out in the format B, G, R, B, G, R, etc. Thus, array(0) will be the first blue pixel, array(1) will be the first green pixel, and so forth. (See Parameters.)

Example  

```
' Read a column of 50 pixels starting at location (10, 0) from the current image
Dim im As Long
Dim buf(50) As Byte
MM.GetCurrentImage im
MM.ReadColumn im, 10, 0, 50, 8, buf
```

See also:  

GetCurrentImage (Section 4.3), GetImage (Section 4.3), ReadColumnEx, ReadPixel, ReadRow  

Continued on next page
6.3 Reading and Manipulating Image Data, continued

ReadColumnEx – for Visual Basic 6 & earlier  

**Description**  
Reads the intensity values of a vertical line of pixels from a selected image and writes the values into an array, starting at a specified starting location in the array.

**Syntax**

```
ReadColumnEx(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long

ReadColumnEx2(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image to be read.
- `xPos` Specifies the X-coordinate of the pixel where reading should start.
- `yPos` Specifies the Y-coordinate of the pixel where reading should start.
- `nPixels` Specifies the number of pixels to be read, starting at the position given by `xPos` and `yPos`. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, *Data Types and Arrays.*)
- `depth` Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, *Data Types and Arrays.*)
- `xStart` Specifies the X-coordinate of the position in the array where writing should start.
- `yStart` Specifies the Y-coordinate of the start position in the array for writing.
- `pixelBuffer()` Defines a buffer into which the pixel values will be read. This is a two-dimensional array. Data are always placed in the array in “column-major” fashion. If `depth` is 8, `pixelBuffer` should be defined as `Byte`. If `depth` is 16, `pixelBuffer` should be `Integer`. If `depth` is 24, `pixelBuffer` should be `Byte`.

**Return values**

- `pixelBuffer()` The pixel values will be read into this predefined buffer. If the source image is 24-bit, the array will be read out in the format B, G, R, B, G, R, etc. Thus, `array(0)` will be the first blue pixel, `array(1)` will be the first green pixel, and so forth. (See Parameters.)

**Example**

```vbnet
' Read a 50 x 50 block of pixels from the current image
' and write it back rotated 90 degrees counterclockwise
Dim im As Long
Dim x As Integer
Dim buf(50, 50) As Byte
MM.GetCurrentImage im

For x = 0 To 49
    MM.ReadColumnEx im, x, 0, 50, 8, 0, x, buf
    MM.WriteRowEx im, 0, x + 50, 50, 8, 0, x, buf
Next x
```
See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), ReadColumn, ReadPixel, ReadRowEx

Continued on next page
6.3 Reading and Manipulating Image Data, continued

**ReadPixel**

**Description**
Reads the intensity value of a specified pixel in a selected image.

**Syntax**
```
ReadPixel(hImage As Long, xPos As Integer, yPos As Integer, nRetPixelValue As Integer) As Long
```

**Remarks**
This function does not operate on 24-bit color images.

**Parameters**
- **hImage** Specifies the handle of the image to be read.
- **xPos** Specifies the X-coordinate of the pixel to be read.
- **yPos** Specifies the Y-coordinate of the pixel to be read.

**Return values**
- **nRetPixelValue** Returns the grayscale value of the pixel.

**Example**
```
' Read a pixel at location 43, 98 on the current image
Dim im As Long
Dim pix As Integer
MM.GetCurrentImage im
MM.ReadPixel im, 43, 98, pix
```

**See also:**
- **GetCurrentImage** (Section 4.3), **GetImage** (Section 4.3), **ReadColumn**, **ReadRow**

---

**ReadRow – for Visual Basic 6 & earlier**

**Description**
Reads the intensity values of a horizontal line of pixels from a selected image and writes them into an array.

**Syntax**
```
ReadRow(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, pixelBuffer() As Variant) As Long
```

**Parameters**
- **hImage** Specifies the handle of the image to be read.
- **xPos** Specifies the X-coordinate of the pixel where reading should start.
- **yPos** Specifies the Y-coordinate of the pixel where reading should start.
- **nPixels** Specifies the number of pixels to be read, starting at the position given by **xPos** and **yPos**. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, *Data Types and Arrays.*)

**Continued on next page**
6.3 Reading and Manipulating Image Data, continued

---

**ReadRow**  
(continued)

*depth*  
Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, *Data Types and Arrays.*)

*pixelBuffer()*  
This is a defined buffer into which the pixel values will be read. If *depth* is 8, *pixelBuffer* should be defined as *Byte*. If *depth* is 16, *pixelBuffer* should be *Integer*. If *depth* is 24, *pixelBuffer* should be *Byte*.

**Return values**

*pixelBuffer()*  
The pixel values will be read into this predefined buffer. If the source image is 24-bit, the array will be read out in the format B, G, R, B, G, R, etc. Thus, array(0) will be the first blue pixel, array(1) will be the first green pixel, and so forth. (See Parameters.)

**Example**

' Read a row of 50 pixels from the current image  
Dim im As Long  
Dim buf(50) As Byte  
MM.GetCurrentImage im  
MM.ReadRow im, 0, 0, 50, 8, buf

**See also:**  
GetCurrentImage (Section 4.3), GetImage (Section 4.3), ReadColumn, ReadPixel, ReadRowEx
ReadRowEx – for Visual Basic 6 & earlier

Description
Reads the intensity values of a horizontal line of pixels from a selected image and writes the values into an array, starting at a specified starting location in the array.

Syntax
`ReadRowEx(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long`

Parameters
- `hImage` Specifies the handle of the image to be read.
- `xPos` Specifies the X-coordinate of the pixel where reading should start.
- `yPos` Specifies the Y-coordinate of the pixel where reading should start.
- `nPixels` Specifies the number of pixels to be read, starting at the position given by `xPos` and `yPos`. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, Data Types and Arrays.)
- `depth` Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, Data Types and Arrays.)
- `xStart` Specifies the X-coordinate (row) of the position in the array where writing should start.

Continued on next page
6.3 Reading and Manipulating Image Data, continued

ReadRowEx 2 (continued)

\textit{yStart} \quad \text{Specifies the Y-coordinate (column) of the position in the array where writing should start.}

\textit{pixelBuffer()} \quad \text{This is a defined buffer into which the pixel values will be read. This is a two-dimensional array. Data are always placed in the array in “column-major” fashion. If depth is 8, \textit{pixelBuffer} should be defined as \textit{Byte}. If depth is 16, \textit{pixelBuffer} should be \textit{Integer}. If depth is 24, \textit{pixelBuffer} should be \textit{Byte}.}

Return values

\textit{pixelBuffer()} \quad \text{The pixel values will be read into this predefined buffer. If the source image is 24-bit, the array will be read out in the format B, G, R, B, G, R, etc. Thus, array(0) will be the first blue pixel, array(1) will be the first green pixel, and so forth. (See Parameters.)}

Example

' Read a 50 x 50 block of pixels from the current image ' and write it back rotated 90 degrees counterclockwise
Dim \textit{im} As Long
Dim \textit{x} As Integer
Dim \textit{buf}(50, 50) As Byte
\textit{MM.GetCurrentImage} \ \textit{im}

For \textit{x} = 0 To 49
    \textit{MM.ReadColumnEx} \ \textit{im}, \textit{x}, 0, 50, 8, 0, \textit{x}, \textit{buf}
    \textit{MM.WriteRowEx} \ \textit{im}, 0, \textit{x} + 50, 50, 8, 0, \textit{x}, \textit{buf}
Next \textit{x}

See also: \textit{GetCurrentImage} (Section 4.3), \textit{GetImage} (Section 4.3), \textit{ReadColumnEx}, \textit{ReadPixel}, \textit{ReadRow}

WriteColumn – for Visual Basic 6 & earlier

Description

Reads a set of intensity values from an array and writes them to a vertical line of pixels in a selected image.

Syntax

\textbf{WriteColumn}(\textit{hImage} As Long, \textit{xPos} As Integer, \textit{yPos} As Integer, \textit{nPixels} As Integer, \textit{depth} As Integer, \textit{pixelBuffer()} As Variant) As Long

Parameters

\textit{hImage} \quad \text{Specifies the handle of the image to be written to.}

\textit{xPos} \quad \text{Specifies the X-coordinate of the pixel where writing should start.}

\textit{yPos} \quad \text{Specifies the Y-coordinate of the pixel where writing should start.}

\textit{nPixels} \quad \text{Specifies the number of pixels to be written, starting at the position given by \textit{xPos} and \textit{yPos}. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, \textit{Data Types and Arrays}.)}
WriteColumn
(continued)

depth  Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, Data Types and Arrays.)

pixelBuffer()  This is a buffer from which the pixel values will be read. If depth is 8, pixelBuffer should be defined as Byte. If depth is 16, pixelBuffer should be Integer. If depth is 24, pixelBuffer should be Byte.

Example

' Write a white vertical line on the current image

Dim im As Long
Dim buf(50) As Byte
Dim x As Integer

For x = 0 To 49
  buf(x) = 255
Next x

MM.GetCurrentImage im
MM.WriteColumn im, 50, 0, 50, 8, buf

See also:  GetCurrentImage (Section 4.3), GetImage (Section 4.3), WriteColumnEx, WritePixel, WriteRow

WriteColumnEx – for Visual Basic 6 & earlier

Description
Reads a set of intensity values from a specified starting location in an array and writes them to a vertical line of pixels in a selected image.

Syntax

WriteColumnEx(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long

WriteColumnEx2(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long

Parameters

hImage  Specifies the handle of the image to be written to.

xPos  Specifies the X-coordinate of the pixel where writing should start.

yPos  Specifies the Y-coordinate of the pixel where writing should start.

nPixels  Specifies the number of pixels to be written, starting at the position given by xPos and yPos. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, Data Types and Arrays.)

depth  Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, Data Types and Arrays.)
6.3 Reading and Manipulating Image Data, continued

WriteColumnEx
(continued)

\textit{xStart}  Specifies the X-coordinate (row) of the position in the array where reading
should start.

\textit{yStart}  Specifies the Y-coordinate (column) of the position in the array where reading
should start.

\textit{pixelBuffer}()  This is a buffer from which the pixel values will be read. If \textit{depth} is 8,
\textit{pixelBuffer} should be defined as \textit{Byte}. If \textit{depth} is 16, \textit{pixelBuffer} should be \textit{Integer}. If
\textit{depth} is 24, \textit{pixelBuffer} should be \textit{Byte}.

Example

\begin{verbatim}
' Read a 50 x 50 block of pixels from the current image
' and write it back rotated 90 degrees clockwise
Dim im As Long
Dim x As Integer
Dim buf(50, 50) As Byte

MM.GetCurrentImage im

For x = 0 To 49
    MM.ReadRowEx im, 0, x, 50, 8, 0, x, buf
    MM.WriteColumnEx im, x + 50, 0, 50, 8, 0, x, buf
Next x
\end{verbatim}

See also:  \textit{GetCurrentImage} (Section 4.3), \textit{GetImage} (Section 4.3), \textit{WriteColumn},
\textit{WritePixel}, \textit{WriteRowEx}

WritePixel

\textbf{Description}  Writes a specified intensity value to a selected image pixel.

\textbf{Syntax}  \textbf{WritePixel}(\textit{hImage As Long, xPos As Integer, yPos As Integer, pixelValue As Integer}) As Long

\textbf{Remarks}  No pixel conversion occurs in this function. This function does not operate on 24-bit
color images.

\textbf{Parameters}  
\textit{hImage}  Specifies the handle of the image to be written to.

\textit{xPos}  Specifies the X-coordinate of the pixel where writing should start.

\textit{yPos}  Specifies the Y-coordinate of the pixel where writing should start.

\textit{pixelValue}  Provides the intensity value to be written to the selected pixel.

\textit{Continued on next page}
6.3 Reading and Manipulating Image Data, continued

WritePixel (continued)

Example

' Write a white pixel at location 50, 50 on the current image
Dim im As Long
MM.GetCurrentImage im
MM.WritePixel im, 50, 50, 255

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), ReadPixel, WriteColumn, WriteRow

WriteRow – for Visual Basic 6 & earlier

Description

Reads a set of intensity values from an array and writes them to a horizontal line of pixels in a selected image.

Syntax

WriteRow(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, pixelBuffer() As Variant) As Long

Parameters

hImage Specifies the handle of the image to be written to.

xPos Specifies the X-coordinate of the pixel where writing should start.

yPos Specifies the Y-coordinate of the pixel where writing should start.

nPixels Specifies the number of pixels to be written, starting at the position given by xPos and yPos. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, Data Types and Arrays.)

depth Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, Data Types and Arrays.)

pixelBuffer() This is a buffer from which the pixel values will be read. If depth is 8, pixelBuffer should be defined as Byte. If depth is 16, pixelBuffer should be Integer. If depth is 24, pixelBuffer should be Byte.

Continued on next page
6.3 Reading and Manipulating Image Data, continued

WriteRow (continued)

Example

' Write a white horizontal line on the current image
Dim im As Long
Dim buf(50) As Byte
Dim x As Integer

For x = 0 To 49
    buf(x) = 255
Next x

MM.GetCurrentImage im
MM.WriteRow im, 50, 0, 50, 8, buf

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), WriteColumn, WritePixel, WriteRowEx

WriteRowEx – for Visual Basic 6 & earlier

Description

Reads a set of intensity values from a specified starting location in an array and writes them to a horizontal line of pixels in a selected image

Syntax

WriteRowEx(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long
WriteRowEx2(hImage As Long, xPos As Integer, yPos As Integer, nPixels As Integer, depth As Integer, xStart As Integer, yStart As Integer, pixelBuffer() As Variant) As Long
Parameters

- **hImage**  Specifies the handle of the image to be written to.
- **xPos**  Specifies the X-coordinate of the pixel where writing should start.
- **yPos**  Specifies the Y-coordinate of the pixel where writing should start.
- **nPixels**  Specifies the number of pixels to be written, starting at the position given by **xPos** and **yPos**. The size of the array will depend on the image bit-depth and the data type used for the array buffer. (See Section 1.6, *Data Types and Arrays*.)
- **depth**  Specifies what the bit-depth of the pixel data values should be. (See Section 1.6, *Data Types and Arrays*.)
- **xStart**  Specifies the X-coordinate (row) of the position in the array where reading should start.
- **yStart**  Specifies the Y-coordinate (column) of the position in the array where reading should start.
- **pixelBuffer()**  This is a buffer from which the pixel values will be read. If **depth** is 8, **pixelBuffer** should be defined as *Byte*. If **depth** is 16, **pixelBuffer** should be *Integer*. If **depth** is 24, **pixelBuffer** should be *Byte*.

*Continued on next page*
6.3 Reading and Manipulating Image Data, continued

WriteRowEx2
(continued)

Example
' Read a 50 x 50 block of pixels from the current image
' and write it back rotated 90 degrees clockwise
Dim im As Long
Dim x As Integer
Dim buf(50, 50) As Byte

MM.GetCurrentImage im

For x = 0 To 49
    MM.ReadRowEx im, 0, x, 50, 8, 0, x, buf
    MM.WriteColumnEx im, x + 50, 0, 50, 8, 0, x, buf
Next x

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3), WriteColumnEx, WritePixel, WriteRow

WriteText

Description
Writes a specified string of text onto an image.

Syntax
WriteText(hImage As Long, xPos As Integer, yPos As Integer, text As String, fillBackground As Boolean, foreColor As Integer, backColor As Integer ) As Long

Parameters
hImage  Specifies the handle of the destination image.

xPos    Specifies the X-coordinate where writing of the text is to start.

yPos    Specifies the Y-coordinate where writing of the text is to start.

text    Provides the string of text that will be written onto the image.

fillBackground  Selects whether the area behind the text is to be filled in with an opaque background of a grayscale value selected by backColor. If fillBackground is set to TRUE, a background “fill” will be used. If fillBackground is set to FALSE, no “fill” will be used, and backColor will be ignored.

foreColor  Specifies the grayscale value of the text.

backColor  Specifies the grayscale value of the “fill” behind the text on the image.

Example
' Write "Nerve Cell" at the top of the current image
Dim im As Long

MM.GetCurrentImage im
MM.WriteText im, 50, 20, "Nerve Cell", TRUE, 255, 0

See also: GetCurrentImage (Section 4.3), GetImage (Section 4.3)
Chapter 7 – Working with Regions of Interest

7.1 Overview

Introduction

A region of interest, or ROI, is an area in an image window that you define and specify for subsequent processing and analysis. For the purposes of user programs, MetaMorph only uses rectangular regions. If you need to use elliptical, linear, or irregularly shaped regions, you will need to use a journal to manipulate them. This chapter presents the Visual Basic functions that you will need for creating and manipulating regions and for measuring and obtaining data from regions.

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating and Removing Regions</td>
<td>97</td>
</tr>
<tr>
<td>Finding Regions</td>
<td>99</td>
</tr>
<tr>
<td>Reading and Manipulating Region Properties</td>
<td>103</td>
</tr>
<tr>
<td>Reading Image Data from Regions</td>
<td>109</td>
</tr>
</tbody>
</table>
### 7.2 Creating and Removing Regions

**Introduction**

The following functions are used for creating and removing regions of interest. At the time you create an ROI, MetaMorph assigns it a handle, just as it assigns handles to functions and entire image windows. All of the remaining functions that involve regions in this chapter will need to be supplied with the region handle. If you need to obtain the handle after you have already created the region, you will need to use the `GetRegion` or `GetActiveRegion` functions which are covered in Section 7.3.

---

#### CreateRectRegion

**Description**

Creates a rectangular region of interest in a specified image. You can specify the size and placement of a region you create simply by dictating the locations of its four corners.

**Syntax**

```
CreateRectRegion(hImage As Long, x1 As Integer, y1 As Integer, x2 As Integer, y2 As Integer, hRetRegion As Long) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image in which the region is to be created.
- `x1` Specifies the X-coordinate of the upper left corner of the region.
- `y1` Specifies the Y-coordinate of the upper left corner of the region.
- `x2` Specifies the X-coordinate of the lower right corner of the region.
- `y2` Specifies the Y-coordinate of the lower right corner of the region.

**Return values**

- `hRetRegion` Returns the handle for the created region.

**Example**

```
' Create a rectangular region on the current image
Dim im As Long
Dim r As Long
MM.GetCurrentImage im
MM.CreateRectRegion im, 10, 10, 50, 50, r
```

**See also:**

`DestroyRegion`, `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3), `SetActiveRegion` (Section 6.4)

---

Continued on next page
7.2 Creating and Removing Regions, continued

DestroyRegion

Description
Removes the specified region of interest.

Syntax
\texttt{DestroyRegion}(hRegion \texttt{As Long}) \texttt{As Long}

Parameters
\textit{hRegion}  Specifies the handle of the region to be removed. The handle can be obtained with the \texttt{GetRegion} function.

Example
\begin{verbatim}
' Delete the active region on the current image, if there ' is one
Dim im As Long
Dim r As Long

MM.GetCurrentImage im
MM.GetActiveRegion im, r

If r <> 0 Then
  MM.DestroyRegion r
Else
  MM.PrintMsg "No active region on the current image"
End If
\end{verbatim}

See also:
\texttt{GetRegion} (Section 7.3), \texttt{GetActive Region} (Section 7.3)
7.3 Finding Regions

Introduction

The functions in this section are essential for the control of procedures involving regions. These functions are used for finding a region’s handle, the “indexing tag” that both Visual Basic and MetaMorph use to keep track of the region, as well as for determining how many regions have been defined and whether a handle is currently in use. The GetActiveRegion and GetRegion functions, which obtain a region’s handle, are particularly important, and you will see these two functions referenced in the “See Also” lists throughout the rest of this chapter.

GetActiveRegion

Description

Obtains the handle number of the active region, if any, in a specified image.

Syntax

GetActiveRegion(hImage As Long, lRetActiveRegion As Long) As Long

Parameters

hImage Specifies the handle of the image.

Return values

lRetActiveRegion Returns the handle of the active region. If there is no active region, the value will be returned as 0. Note: An image can contain regions without any of them being active.

Example

' Delete the active region on the current image, if there ' is one
Dim im As Long
Dim r As Long

MM.GetCurrentImage im
MM.GetActiveRegion im, r

If r <> 0 Then
    MM.DestroyRegion r
Else
    MM.PrintMsg "No active region on the current image"
End If

See also:

GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetRegion, SetActiveRegion
### 7.3 Finding Regions, continued

#### GetNumberOfRegions

**Description**

Returns the number of regions that exist on a specified image.

**Syntax**

```
GetNumberOfRegions(hImage As Long, nRetNumberOfRegions As Integer) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image.

**Return values**

- `nRetNumberOfRegions` Returns the number of regions that currently exist on the image.

**Example**

```vbnet
' Get the number of regions on the current image

Dim im As Long
Dim rCount As Integer

MM.GetCurrentImage im

MM.GetNumberOfRegions im, rCount
MM.PrintMsg "There are " + Str(rCount) + " regions on the current image"
```

**See also:**

- `GetCurrentImage` (Section 4.3), `GetImage` (Section 4.3)

#### GetRegion

**Description**

Obtains the handle number of a specified region.

**Syntax**

```
GetRegion(hImage As Long, index As Integer, lRetRegion As Long) As Long
```

**Remarks**

All of the regions loaded in MetaMorph have an index number, from 0 to \( n - 1 \), where \( n \) is the number of loaded regions. Regions are numbered according to their order of creation; thus, the first region is 0, the next is 1, and so on.

**Parameters**

- `hImage` Specifies the handle of the image.
- `index` Specifies the number of the region. (See Remarks.)

**Return values**

- `lRetRegion` Returns the handle of the selected region. (*Note:* The handle is a number used by MetaMorph to carry out programmatic functions. This differs from the index number associated with the region.) If an error occurs, for example, if `index` is not a valid number, `lRetRegion` will contain 0.

---

*Continued on next page*
7.3 Finding Regions, continued

GetRegion (continued)

Example

' Get the first region on the current image
Dim im As Long
Dim r As Long
MM.GetRegion im, 0, r

See also: GetActiveRegion, GetCurrentImage (Section 4.3), GetImage (Section 4.3)

IsValidRegion

Description
Tests whether or not the region handle you pass is valid.

Syntax
IsValidRegion(hRegion As Long) As Long

Parameters
hRegion Specifies the handle of the region in question. If the handle value that you pass is not valid, IsValidRegion returns a value of “0”. The function will return a nonzero value if the region handle is valid.

Example

' Check if some region is valid, and if it is, make it the active region. 'r' and 'im' are region and image handles that were declared and used previous to the code here.

If MM.IsValidRegion(r) <> 0 Then
    MM.PrintMsg "Region is valid"
    MM.SetActiveRegion im, r
Else
    MM.PrintMsg "Region is no longer valid"
End If

See also: GetRegion

Continued on next page
SetActiveRegion

**Description**
Makes a selected region the active region.

**Syntax**

```visualbasic
SetActiveRegion(hImage As Long, hRegion As Long) As Long
```

**Parameters**

- `hImage` Specifies the handle of the image.
- `hRegion` Specifies the handle of the region to be made active.

**Example**

' Check if some region is valid, and if it is, make it the active region. 'r' and 'im' are region and image handles that were declared and used previous to the code here.

```visualbasic
If MM.IsValidRegion(r) <> 0 Then
    MM.PrintMsg "Region is valid"
    MM.SetActiveRegion im, r
Else
    MM.PrintMsg "Region is no longer valid"
End If
```

**See also:**

GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetRegion
### 7.4 Reading and Manipulating Region Properties

**Introduction**

It is often necessary or useful to obtain information about a region, such as its width, height, area, perimeter, or location. The functions that follow can be used to read or configure such region properties, as well as to obtain the X and Y coordinates of the region’s outline. One programming example is given for all of the region property “Get” functions (see Figure 7.1 on page 106).

---

**GetRegionArea**

**Description**

Obtains the area, expressed as the number of pixels, of a specified region of interest.

**Syntax**

\[
\text{GetRegionArea}(h\text{Region} \text{ As Long, } l\text{RetArea} \text{ As Long) As Long}
\]

**Parameters**

- \( h\text{Region} \): Specifies the handle of the region in question.

**Return values**

- \( l\text{RetArea} \): Returns the number of pixels in the region.

**See also:**

GetActiveRegion (Section 7.3), GetRegion (Section 7.3), GetRegionSize

---

**GetRegionDistance**

**Description**

Obtains the perimeter of a specified region of interest. The distance measurement will be reported in calibrated units.

**Syntax**

\[
\text{GetRegionDistance}(h\text{Region} \text{ As Long, } d\text{RetDistance} \text{ As Double) As Long}
\]

**Remarks**

This function differs from RegionGetNumEdgePixels in its expression of the perimeter in calibrated units. RegionGetNumEdgePixels expresses the perimeter in terms of the number of pixels.

**Parameters**

- \( h\text{Region} \): Specifies the handle of the region in question.

**Return values**

- \( d\text{RetDistance} \): Returns the perimeter of the region, in calibrated units.

**See also:**

GetActiveRegion (Section 7.3), GetRegion (Section 7.3), RegionGetNumEdgePixels

---

*Continued on next page*
7.4 Reading and Manipulating Region Properties, continued

GetRegionPosition

Description
Obtains the on-screen position of a specified region of interest.

Syntax
GetRegionPosition(hRegion As Long, nRetXPos As Integer, nRetYPos As Integer) As Long

Parameters
hRegion Specifies the handle of the region in question.

Return values
nRetXPos Returns the starting (upper left) X-coordinate of the specified region.
nRetYPos Returns the starting Y-coordinate of the specified region.

See also:
GetActiveRegion (Section 7.3), GetRegion (Section 7.3), GetRegionSize,
SetRegionPosition

GetRegionSize

Description
Obtains the width and height of a specified region of interest.

Syntax
GetRegionSize(hRegion As Long, nRetXSize As Integer, nRetYSize As Integer) As Long

Parameters
hRegion Specifies the handle of the region in question.

Return values
nRetXSize Returns the width, in pixels, of the specified region.
nRetYSize Returns the height, in pixels, of the specified region.

See also:
GetActiveRegion (Section 7.3), GetRegion (Section 7.3), GetRegionArea,
GetRegionPosition, SetRegionSize

Continued on next page
7.4 Reading and Manipulating Region Properties, continued

RegionGetNumEdgePixels

Description
Obtains the perimeter of a specified region of interest. The distance measurement will be reported in terms of the number of pixels.

Syntax
RegionGetNumEdgePixels(hRegion As Long, lNumPixels As Long) As Long

Remarks
RegionGetNumEdgePixels returns in lNumPixels the number of pixels comprising the edge of the given region. This function will usually be used in conjunction with RegionGetEdgePixelCoordinates. You would call RegionGetNumEdgePixels first to determine how big to make the arrays you pass to RegionGetEdgePixelCoordinates.

Parameters
hRegion Specifies the handle of the region in question.

Return values
lNumPixels Returns the perimeter of the region, in pixels.

See also:
GetActiveRegion (Section 7.3), GetRegion (Section 7.3), GetRegionDistance

RegionGetEdgePixelCoordinates – for Visual Basic 6 & earlier

Description
Obtains the X and Y coordinates of all the pixels lying under the edge of a region.

Syntax
RegionGetEdgePixelCoordinates(hRegion As Long, lBufferSize As Long, aRetXBuffer() As Integer, aRetYBuffer() As Integer) As Long
RegionGetEdgePixelCoordinatesEx2(hRegion As Long, lBufferSize As Long, aRetXBuffer() As Integer, aRetYBuffer() As Integer) As Long

Parameters
hRegion Specifies the handle of the region in question.

lBufferSize Sets the size of the array into which the X and Y coordinates will be read. You can use RegionGetNumEdgePixels to determine how many pixels that should be.

Return values
aRetXBuffer() Returns the X-coordinates of the pixels lying under the edge of the region. The size of this array will be determined by the number of pixels in the edgelist.

aRetYBuffer() Returns the Y-coordinates of the pixels lying under the edge of the region. The size of this array will be determined by the number of pixels in the edgelist.

See also:
GetActiveRegion (Section 7.3), GetRegion (Section 7.3), RegionGetNumEdgePixels

Continued on next page
7.4 Reading and Manipulating Region Properties, continued

Figure 7.1 Region Property “Get” Function Programming Example

```vbnet
' Get property information about the active region
Dim im As Long
Dim r As region

MM.GetCurrentImage im
MM.GetActiveRegion im, r

Dim area As Long
MM.GetRegionArea r, area
MM.PrintMsg "Pixel area of the active region is " + Str(area)

Dim distance As Double
MM.GetRegionDistance r, distance
MM.PrintMsg "Calibrated perimeter of the active region is " + Str(distance)

Dim x As Integer, y As Integer
MM.GetRegionPosition r, x, y
Dim dx As Integer, dy As Integer
MM.GetRegionSize r, dx, dy
MM.PrintMsg "Position of the region is " + Str(x) + ", " + Str(y)
MM.PrintMsg "Size of the region is " + Str(dx) + ", " + Str(dy)

Dim numpix As Long
' First find out how many pixels are in the edge of
' the region
MM.RegionGetNumEdgePixels r, numpix

' Then get the coordinates of all the edge pixels
Dim x(numpix) As Integer, y(numpix) As Integer
MM.RegionGetEdgePixelCoordinates r, numpix, x, y
```

Continued on next page
7.4 Reading and Manipulating Region Properties, continued

SetRegionPosition

Description: Moves a selected region of interest to a specified position.

Syntax: 

```
SetRegionPosition(hRegion As Long, xPos As Integer, yPos As Integer) As Long
```

Parameters:

- `hRegion` Specifies the handle of the region in question.
- `xPos` Specifies the X-coordinate of the new starting point (upper left corner) of the selected region.
- `yPos` Specifies the Y-coordinate of the new starting point of the selected region.

Example:

```
' Move the active region to the upper left corner of the image ' and make it 100 x 100 pixels in size

Dim im As Long
Dim r As Long

MM.GetCurrentImage im
MM.SetActiveRegion im, r

MM.SetRegionPosition r, 0, 0
MM.SetRegionSize r, 100, 100
```

See also: 

GetRegion (Section 7.3), GetRegionPosition

SetRegionSize

Description: Resizes a selected region of interest to a specified width and height.

Syntax: 

```
SetRegionSize(hRegion As Long, xSize As Integer, ySize As Integer) As Long
```

Parameters:

- `hRegion` Specifies the handle of the region in question.
- `xSize` Specifies the new width, in pixels, of the selected region.
- `ySize` Specifies the new height, in pixels, of the selected region.

Continued on next page
7.4 Reading and Manipulating Region Properties, continued

SetRegionSize
(continued)

Example

' Move the active region to the upper left corner of the image
' and make it 100 x 100 pixels in size

Dim im As Long
Dim r As Long

_MM.GetCurrentImage im
_MM.SetActiveRegion im, r

_MM.SetRegionPosition r, 0, 0
_MM.SetRegionSize r, 100, 100

See also: GetActiveRegion (Section 7.3), GetRegion (Section 7.3), GetRegionSize

SwapRegionNumbers

Description
Exchanges the numbers associated with two selected regions.

Syntax

SwapRegionNumbers(rhRegion1 As Long, rhRegion2 As Long)

Parameters

rhRegion1  Specifies the handle of one of the two selected regions.

rhRegion2  Specifies the handle of the other selected region.

See also: GetActiveRegion (Section 7.3), GetRegion (Section 7.3)
7.5 Reading Image Data from Regions

Introduction
This section describes a number of functions which are invaluable for obtaining grayscale information from a region of interest. You can derive the minimum, maximum, and average intensity value in a region, as well as the standard deviation around the mean. One programming example is given for all of the region data “Get” functions (see Figure 7.2 on page 112). You will need to apply the **MeasureRegion** function as a preliminary step to obtaining these data. If your images have been calibrated in MetaMorph with the Calibrate Distances or Calibrate Gray Levels commands, the “read” functions in this section will return data expressed in calibrated units, rather than as numbers of pixels or as grayscale levels.

---

**GetRegionAverageValue**

**Description**
Obtains the average of all of the grayscale values in a specified region of interest.

**Syntax**

```
GetRegionAverageValue(hRegion As Long, dRetAverageValue As Double) As Long
```

**Remarks**
Before you apply this function, the region must first be measured with the **MeasureRegion** function.

**Parameters**

- **hRegion**  Specifies the handle of the region.

**Return values**

- **dRetAverageValue**  Returns the average intensity value from the region.

**See also:**
- **GetActive Region** (Section 7.3), **GetRegion** (Section 7.3), **GetRegionMaximumValue**, **GetRegionMinimumValue**, **GetRegionStdDeviation**, **MeasureRegion**

---

**GetRegionMinimumValue**

**Description**
Obtains the lowest grayscale value in a specified region of interest.

**Syntax**

```
GetRegionMinimumValue(hRegion As Long, dRetMinValue As Double) As Long
```

**Remarks**
Before you apply this function, the region must first be measured with the **MeasureRegion** function.

**Parameters**

- **hRegion**  Specifies the handle of the region.

**Return values**

- **dRetMinValue**  Returns the minimum intensity value in the region.

**See also:**
- **GetActive Region** (Section 7.3), **GetRegion** (Section 7.3), **GetRegionMaximumValue**, **GetRegionMinimumValue**, **GetRegionStdDeviation**, **MeasureRegion**

---

Continued on next page
### GetRegionMaximumValue

**Description**
Obtains the highest grayscale value in a specified region of interest.

**Syntax**
`GetRegionMaximumValue(hRegion As Long, dRetMaxValue As Double) As Long`

**Remarks**
Before you apply this function, the region must first be measured with the `MeasureRegion` function.

**Parameters**
- `hRegion`  Specifies the handle of the region.

**Return values**
- `dRetMaxValue`  Returns the maximum intensity value in the region.

**See also:**
- `GetActive Region` (Section 7.3), `GetRegion` (Section 7.3),
- `GetRegionAverageValue`, `GetRegionMinimumValue`, `GetRegionStdDeviation`,
- `MeasureRegion`

---

### GetRegionStdDeviation

**Description**
Obtains the standard deviation of the grayscale values in a specified region of interest.

**Syntax**
`GetRegionStdDeviation(hRegion As Long, dRetStdDeviation As Double) As Long`

**Remarks**
Before you apply this function, the region must first be measured with the `MeasureRegion` function.

**Parameters**
- `hRegion`  Specifies the handle of the region.

**Return values**
- `dRetStdDeviation`  Returns the standard deviation of the intensity values in the region.

**See also:**
- `GetActive Region` (Section 7.3), `GetRegion` (Section 7.3),
- `GetRegionAverageValue`, `GetRegionMaximumValue`,
- `GetRegionMinimumValue`, `MeasureRegion`

---

Continued on next page
7.5  Reading Image Data from Regions, continued

GetRegionThresholdArea

Description
Obtains the number of pixels in a specified region that are inside the current threshold.

Syntax
GetRegionThresholdArea(hRegion As Long, lRetArea As Long) As Long

Remarks
Before you apply this function, the region must first be measured with the MeasureRegion function.

Parameters
hRegion  Specifies the handle of the region.

Return values
lRetArea  Returns the number of thresholded pixels in the region.

See also:
GetActive Region (Section 7.3), GetRegion (Section 7.3), MeasureRegion

MeasureRegion

Description
Measures the grayscale data in a specified region of interest of an image.

Syntax
MeasureRegion(hRegion As Long, hImage As Long, bUseThreshold As Boolean) As Long

Parameters
hRegion  Specifies the handle of the region.

hImage  Specifies the handle of the image.

bUseThreshold  Determines whether the measurement will be made of just those pixels that are thresholded, or if all pixels are to be measured. If only thresholded pixels are to be measured, bUseThreshold should be set to TRUE. If all pixels are to be measured, set bUseThreshold to FALSE.

See also:
GetActive Region (Section 7.3), GetCurrentImage (Section 4.3), GetImage (Section 4.3), GetRegion (Section 7.3)
Figure 7.2  Region Data “Get” Function Programming Example

' Measure the active region on the current image and print ' out the information that was measured

Dim im As Long
Dim r As region

MM.GetCurrentImage  im
MM.GetActiveRegion  r

MM.MeasureRegion  r, im, FALSE        ' Do not use threshold
Dim average As Double
Dim min As Double
Dim max As Double
Dim stddev As Double
Dim area As Long
MM.GetRegionAverageValue  r, average
MM.GetRegionMinimumValue  r, min
MM.GetRegionMaximumValue  r, max
MM.GetRegionStdDeviation  r, stddev
MM.GetRegionThresholdArea  r, area

MM.PrintMsg "Average is " + Str(average)
MM.PrintMsg "Minimum is " + Str(min)
MM.PrintMsg "Maximum is " + Str(max)
MM.PrintMsg "Standard deviation is " + Str(stddev)
MM.PrintMsg "Threshold area should be 0, because the region was"

MM.PrintMsg "Measured without thresholding. The area is: " + Str(area)
Chapter 8 – Performing Morphometry

8.1 Overview

Introduction

As the premier morphometric analysis system, MetaMorph offers an extensive array of functions for measurement and analysis of image objects. This chapter discusses the Visual Basic functions that you will need for configuring your morphometric measurements, creating classifier filters, and measuring objects. The operative function that measures your image, **MorphMeasureObjects**, is covered in Section 8.5. Before you can apply this function, however, please remember to define a threshold range, as detailed in Section 6.2, *Applying Thresholding*.

In this chapter

This chapter contains the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring Measurement Preferences</td>
<td>114</td>
</tr>
<tr>
<td>Configuring Object Measurements</td>
<td>115</td>
</tr>
<tr>
<td>Configuring Classifier Filters</td>
<td>119</td>
</tr>
<tr>
<td>Measuring All Objects in an Image</td>
<td>121</td>
</tr>
<tr>
<td>Measuring Single Objects</td>
<td>126</td>
</tr>
</tbody>
</table>
8.2 Configuring Measurement Preferences

Introduction

Before you measure your images, you may want to configure MetaMorph’s behavior during the measurement procedure. The settings you use for the two functions that follow will determine two types of behavior: whether objects that “fail” a classifier filter (see Section 8.4) are redrawn in the result image, and whether “holes” (unthresholded areas in the middle of an object) are “filled” prior to measurement.

MorphSetDrawFailedObjectsFlag

Description

Sets whether objects that fail to pass any classifiers should be drawn in the resultant measurement image.

Syntax

MorphSetDrawFailedObjectsFlag(bRedraw As Boolean) As Long

Parameters

bRedraw  Determines whether failed objects will be drawn. If bRedraw is set to TRUE, failed objects will be drawn. If bRedraw is set to FALSE, failed objects will not be drawn.

Example

' The next time measurements are performed, don't draw objects
' that don't pass any classifiers and fill holes in objects
MM.MorphSetDrawFailedObjectsFlag  FALSE
MM.MorphSetFillHoleFlag  TRUE

MorphSetFillHoleFlag

Description

Sets whether “holes” (unthresholded areas enclosed within thresholded areas) will be filled in (treated as thresholded) when measurement occurs.

Syntax

MorphSetFillHoleFlag(bRedraw As Boolean) As Long

Remarks

If bRedraw is TRUE, they will be filled in. If FALSE, they will not be.

Parameters

bRedraw  Determines whether or not holes are to be filled. If bRedraw is set to TRUE, holes will be filled and subsequent measurements will include their areas. If bRedraw is set to FALSE, holes will not be filled.

Example

' The next time measurements are performed, don't draw objects
' that don't pass any classifiers and fill holes in objects
MM.MorphSetDrawFailedObjectsFlag  FALSE
MM.MorphSetFillHoleFlag  TRUE
8.3 Configuring Object Measurements

Introduction

Just as image windows, functions, and regions are manipulated by their handles, so too are object parameters manipulated by using an index number, the parameter number. Several configuration and measurement functions depend on the correct parameter number being passed to them. In this section, we discuss several functions that allow you to obtain a parameter’s number, or to obtain other information about a parameter based on the number you pass. In addition, another measurement configuration function, MorphSetupMeasurements, is used for specifying the image to be measured and a second image, the mask image, which will be used to define the distribution of the thresholding overlay.

MorphFindParmIndex

Description

Obtains the number of a specified parameter.

Syntax

MorphFindParmIndex(sParmName As String, nRetParmNumber As Integer) As Long

Parameters

sParmName  Specifies the name of the parameter for which you want to obtain a parameter number. The parameter names that you can use consist of those used for the MetaMorph morphometry functions. You can see these names in the Configure Object Classifiers, Configure Object Measurements, or Integrated Morphometry Analysis dialog boxes.

nRetParmNumber  Returns the number of the parameter specified by sParmName. If no parameter by that name exists, nRetParmNumber will be returned as -1.

Example

' Find the width of object 5.  This code assumes that ' measurements have already been made.
Dim I As Integer
Dim nWidth As Single
MM.MorphFindParmIndex "Width", n
MM.MorphGetParmMeasurement 5, n, nWidth
MM.PrintMsg "The width of object 5 is " + Str(nWidth)

See also:

MorphGetParmDescription, MorphGetParmMeasurement (Section 8.6), MorphGetParmName

Continued on next page
8.3 Configuring Object Measurements, continued

MorphGetNumberOfParms

Description Obtains the total number of measurable parameters.

Syntax `MorphGetNumberOfParms(nRetParms As Integer) As Long`

Return values `nRetParms` Returns the total number of measurable parameters.

Example

' Print out the names and descriptions of all the measurement parameters
Dim nParms As Integer
Dim i As Integer
Dim parmName As String
Dim desc As String

`MM.MorphGetNumberOfParms nParms`
For i = 0 To nParms - 1
  `MM.MorphGetParmName i, parmName`
  `MM.MorphGetParmDescription i, desc`
  `MM.PrintMsg parmName + ": " + desc`
Next i

MorphGetParmDescription

Description Obtains a description of a parameter for which you have a parameter number.

Syntax `MorphGetParmDescription(nParmNumber As Integer, sRetParmDescription As String) As Long`

Parameters `nParmNumber` Specifies the number of the parameter for which you want to obtain a description. This number can be obtained with the `MorphFindParmIndex` function.

Return values `sRetParmDescription` Returns a textual description of the parameter in question.

Continued on next page
MorphGetParmDescription
(continued)

Example

' Print out the names and descriptions of all the measurement
' parameters

Dim nParms As Integer
Dim i As Integer
Dim parmName As String
Dim desc As String

MM.MorphGetNumberOfParms nParms
For i = 0 To nParms - 1
    MM.MorphGetParmName i, parmName
    MM.MorphGetParmDescription i, desc
    MM.PrintMsg parmName + ": " + desc
Next i

See also: MorphFindParmIndex, MorphGetParmName

MorphGetParmName

Description
Obtains the name of a parameter for which you have a parameter number.

Syntax

MorphGetParmName(nParmNumber As Integer, sRetParmName As String) As Long

Parameters

nParmNumber  Specifies the number of the parameter for which you want to obtain a
description. This number can be obtained with the MorphFindParmIndex function.

Return values

sRetParmName  Returns the name of the parameter in question.

Example

' Print out the names and descriptions of all the measurement
' parameters

Dim nParms As Integer
Dim i As Integer
Dim parmName As String
Dim desc As String

MM.MorphGetNumberOfParms nParms
For i = 0 To nParms - 1
    MM.MorphGetParmName i, parmName
    MM.MorphGetParmDescription i, desc
    MM.PrintMsg parmName + ": " + desc
Next i

See also: MorphFindParmIndex, MorphGetParmDescription
8.3 Configuring Object Measurements, continued

**MorphSetupMeasurements**

**Description**
Selects images to be used as measurement and mask images for subsequent measurement with **MorphMeasureObjects**.

**Syntax**

```
MorphSetupMeasurements(hGrayImage As Long, hMaskImage As Long, fStdArea As Single) As Long
```

**Remarks**
This function selects the measurement image and mask image for use in subsequent measurements with **MorphMeasureObjects**. Also, if you want to count objects, this function allows you to define and make measurements with a *standard area*. Counting objects can sometimes be difficult because they may overlap or are poorly defined in the image. A standard area is a value that you believe represents the area of a standard object, based on the assumption that the objects being measured are of a fairly uniform size. The clumps of objects will be counted by dividing the area of the clump by the standard area.

If you are not interested in using a standard area, you can set `fStdArea` to 1.

**Parameters**
- `hGrayImage` Specifies the handle of the image to be used for measurement. Image handles can be obtained with **GetImage**.
- `hMaskImage` Specifies the handle of the image to be used as a mask image. Typically, `hMaskImage` is an image created by setting a threshold on `hGrayImage` and then using **BinarizeImage** to create a binary image from that.
- `fStdArea` Defines the size of the standard area. The units used depends on whether you have calibrated the image for distance.

**Example**

```
' Create a binary mask of the current image and then measure ' the image
Dim im As Long

' Put a threshold on the current image
MM.GetCurrentImage im
MM.SetThresholdState im, 1
MM.SetThresholdRange im, 50, 150

' Create the binarized image
Dim bin As Long
MM.CreateImage 512, 512, 1, "binary image", bin
MM.BinarizeImage im, bin

' Measure
MM.MorphSetupMeasurements im, bin, 1#
MM.MorphMeasureObjects TRUE
```

**See also:**
- **BinarizeImage** (Section 6.3), **GetCurrentImage** (Section 4.3), **GetImage** (Section 4.3), **MorphMeasureObjects** (Section 8.5)
8.4 Configuring Classifier Filters

Introduction
Classifier filters are morphometric measurement ranges through which an object’s measurements must “pass” to be included in the final set. Using a classifier filter, you can restrict your measurements to just those objects that meet your set criteria, while excluding other classes of objects. The two functions which follow are used for reading the current settings of a classifier filter for a specified parameter and for configuring those settings.

MorphGetFilter

Description
Obtains the settings of the classifier filter for a specified parameter.

Syntax
`MorphGetFilter(nClassifierNumber As Integer, nParmNumber As Integer, fRetMinVal As Single, fRetMaxVal As Single, bRetInclusive As Boolean, bRetEnableFilter As Boolean) As Long`

Parameters
- `nClassifierNumber` Specifies the number of the classifier filter in question (0 – 7). This classifier is one you will have defined, and the number is one you will have assigned with the `MorphSetFilter` function.
- `nParmNumber` Specifies the parameter number of the classifier for which the filter is being configured. This number can be obtained with the `MorphFindParmIndex` function. These predefined classifiers are those that correspond to the parameters used for the MetaMorph morphometry functions. You can see these parameters in the Configure Object Classifiers, Configure Object Measurements, or Integrated Morphometry Analysis dialog boxes.

Return values
- `fRetMinVal` Returns the minimum value of the filter.
- `fRetMaxVal` Returns the maximum value of the filter.
- `bRetInclusive` Indicates whether the range minimum and maximum are inclusive (that is, the filter “passes” objects if they have values between the minimum and maximum) or exclusive (that is, the filter “passes” objects if they have values equal to or outside the minimum and maximum). If the filter has been set to the inclusive state, a `bRetInclusive` value of TRUE will be returned. If the filter is exclusive, a value of FALSE will be returned.
- `bRetEnableFilter` Indicates whether the filter is active or not. If the filter is active, `bRetEnableFilter` will return with a value of TRUE. If the filter is inactive, a value of FALSE will be returned.

Continued on next page
8.4 Configuring Classifier Filters, continued

MorphGetFilter (continued)

Example

' Tell the first filter to stop filtering on total area, while
' leaving the filtering parameters unchanged.
Dim min As Single, max As Single
Dim inclusive As Boolean, enable As Boolean
MM.MorphGetFilter 0, 0, min, max, inclusive, enable
MM.MorphSetFilter 0, 0, min, max, inclusive, FALSE

See also: MorphFindParmIndex (Section 8.3), MorphSetFilter

MorphSetFilter

Description
Configures the settings of a classifier filter for a specified parameter.

Syntax
MorphSetFilter(nClassifierNumber As Integer, nParmNumber As Integer, fMinVal As Single, fMaxVal As Single, bInclusive As Boolean, bEnableFilter As Boolean) As Long

Parameters
nClassifierNumber  Specifies a number for the classifier (0 – 7). Typically, this number will be based on the order of configuration.

nParmNumber  Specifies the parameter number of the classifier for which the filter is being configured. This number can be obtained with the MorphFindParmIndex function.

fMinVal  Specifies the minimum value of the filter.

fMaxVal  Specifies the maximum value of the filter.

bInclusive  Specifies whether the range minimum and maximum are to be inclusive (that is, the filter will “pass” objects that have values between the minimum and maximum) or exclusive (that is, the filter will “pass” objects that have values equal to or outside the minimum and maximum). To set the filter to the inclusive state, assign a value of TRUE to bInclusive. To make the filter exclusive, assign a value of FALSE.

bEnableFilter  Specifies whether the filter will be active or inactive. To activate the filter, set bEnableFilter to TRUE. To deactivate the filter, assign a value of FALSE.

Example

' Set the first filter to only pass objects whose total area
' is between 10 and 50.
MM.MorphSetFilter 0, 0, 10, 50, TRUE, TRUE

See also: MorphFindParmIndex (Section 8.3), MorphGetFilter
8.5 Measuring All Objects in an Image

Introduction
MetaMorph automatically measures all objects in an image, but the resulting data are managed either as a group or on an object-by-object basis. Data for the entire group are presented as a statistical summary, and can be saved in a summary log. The functions in this section deal with obtaining and managing group measurement data. This includes the important `MorphMeasureObjects` function. (For a discussion of single object measurement functions, see Section 8.6.)

MorphDeleteObject

Description
Removes a specified object from the list of measured objects.

Syntax
`MorphDeleteObject(nObjectID As Integer) As Long`

Remarks
You can prevent a specific object from having a contribution to the summary data by using this function in conjunction with `MorphRecalc`.

Parameters
`nObjectID` Specifies the number of the object to be deleted. Objects are numbered from 0 to \((n-1)\), where \(n\) is the total number of objects.

Example
```
' Delete object 4 from the morph summary data. This code assumes measurements have already been performed.
MM.MorphDeleteObject 4
MM.MorphRecalc
```

See also:
`MorphRecalc`

MorphGetNumberOfObjects

Description
Obtains the number of objects that were last measured by `MorphMeasureObjects`.

Syntax
`MorphGetNumberOfObjects(nRetObjects As Integer) As Integer`

Return values
`nRetObjects` Returns the number of objects that were measured.

See also:
`MorphMeasureObjects`

Continued on next page
8.5 Measuring All Objects in an Image, continued

**MorphGetParmAccumSummary**

**Description**
Obtains the cumulative statistics for all objects that passed a specified classifier filter for a selected parameter. This function returns the same information as **MorphGetParmSummary**, but combines the information with all previous measurements that have been made using the selected classifier filter.

**Syntax**

```
MorphGetParmAccumSummary(nClassifierNumber As Integer, nParmNumber As Integer, lRetCount As Long, fRetAverage As Single, fRetMinVal As Single, fRetMaxVal As Single, fRetStdDeviation As Single, fRetTotal As Single) As Long
```

**Parameters**

- `nClassifierNumber` Specifies the classifier number (0 – 7). This number is one that you will have assigned with the **MorphSetFilter** function.
- `nParmNumber` Specifies the parameter number of the classifier filter. This number can be obtained with the **MorphFindParmIndex** function.

**Return values**

- `lRetCount` Returns the total number of objects which passed the classifier.
- `fRetAverage` Returns the average value of the given parameter for all passed objects.
- `fRetMinVal` Returns the lowest value of the parameter for all objects.
- `fRetMaxVal` Returns the highest value of the parameter for all objects.
- `fRetStdDeviation` Returns the standard deviation of the parameter for all objects.
- `fRetTotal` Returns the sum of the values for all objects.

**Example**

```
' Get the parameter summary and the accumulated summary data of
' the first filter for total area
MM.MorphGetParmSummary 0, 0, count, average, min, max, stddev, total

Dim count As Long
Dim average As Single, min As Single, max As Single
Dim stddev As Single, total As Single

MM.PrintMsg "The count is " + Str(count)
MM.PrintMsg "The average is " + Str(average)
MM.PrintMsg "The min is " + Str(min)
MM.PrintMsg "The max is " + Str(max)
MM.PrintMsg "The standard deviation is " + Str(stddev)
MM.PrintMsg "The total is " + Str(total)

MM.MorphGetParmAccumSummary 0, 0, count, average, min, max, stddev, total

Continued on next page
8.5 Measuring All Objects in an Image, continued

MorphGetParmAccumSummary (continued)

MM.PrintMsg "The accumulated count is " + Str(count)
MM.PrintMsg "The accumulated average is " + Str(average)
MM.PrintMsg "The accumulated min is " + Str(min)
MM.PrintMsg "The accumulated max is " + Str(max)
MM.PrintMsg "The accumulated standard deviation is "
    + Str(stddev)
MM.PrintMsg "The accumulated total is " + Str(total)

See also: MorphFindParmIndex (Section 8.3), MorphGetParmSummary, MorphSetFilter (Section 8.4)

MorphGetParmSummary

Description
Obtains the statistics for all objects that passed a specified classifier filter for a selected parameter.

Syntax
MorphGetParmSummary(nClassifierNumber As Integer, nParmNumber As Integer, lRetCount As Long, fRetAverage As Single, fRetMinVal As Single, fRetMaxVal As Single, fRetStdDeviation As Single, fRetTotal As Single) As Long

Parameters
nClassifierNumber  Specifies the classifier number (0 – 7). This number is one that you will have assigned with the MorphSetFilter function.

nParmNumber  Specifies the parameter number of the classifier filter. This number can be obtained with the MorphFindParmIndex function.

Return values
lRetCount  Returns the total number of objects which passed the classifier.

fRetAverage  Returns the average value of the given parameter for all passed objects.

fRetMinVal  Returns the lowest value of the parameter for all objects.

fRetMaxVal  Returns the highest value of the parameter for all objects.

fRetStdDeviation  Returns the standard deviation of the parameter for all objects.

fRetTotal  Returns the sum of the values for all objects.

Continued on next page
8.5 Measuring All Objects in an Image, continued

MorphGetParmSummary
(continued)

Example

' Get the parameter summary and the accumulated summary data of
' the first filter for total area
MM.MorphGetParmSummary 0, 0, count, average, min, max, stddev,
  total

Dim count As Long
Dim average As Single, min As Single, max As Single
Dim stddev As Single, total As Single

MM.PrintMsg "The count is " + Str(count)
MM.PrintMsg "The average is " + Str(average)
MM.PrintMsg "The min is " + Str(min)
MM.PrintMsg "The max is " + Str(max)
MM.PrintMsg "The standard deviation is " + Str(stddev)
MM.PrintMsg "The total is " + Str(total)

MM.MorphGetParmAccumSummary 0, 0, count, average, min, max,
  stddev, total

MM.PrintMsg "The accumulated count is " + Str(count)
MM.PrintMsg "The accumulated average is " + Str(average)
MM.PrintMsg "The accumulated min is " + Str(min)
MM.PrintMsg "The accumulated max is " + Str(max)
MM.PrintMsg "The accumulated standard deviation is "+ Str(stddev)
MM.PrintMsg "The accumulated total is " + Str(total)

See also:
MorphFindParmIndex (Section 8.3), MorphGetParmAccumSummary,
MorphSetFilter (Section 8.4)

MorphMeasureObjects

Description
Measures the hGrayImage image that was last specified by
MorphSetupMeasurements.

Syntax
MorphMeasureObjects(bMeasureAll As Boolean) As Long

Parameters
bMeasureAll Determines whether object measurements will be handled on a single
basis only or as a whole. If you set bMeasureAll to TRUE, all of the objects in the
image will be measured and their data made available to the various summary data
functions, as well as to the object data functions. If you set this variable to FALSE,
data will be passed only to the object data functions.
8.5 Measuring All Objects in an Image, continued

MorphMeasureObjects
(continued)

Example

' Create a binary mask of the current image and then measure
' the image

Dim im As Long

' Put a threshold on the current image
MM.GetCurrentImage im
MM.SetThresholdState im, 1
MM.SetThresholdRange im, 50, 150

' Create the binarized image
Dim bin As Long
MM.CreateImage 512, 512, 1, "binary image", bin
MM.BinarizeImage im, bin

' Measure
MM.MorphSetupMeasurements im, bin, 1#
MM.MorphMeasureObjects TRUE

See also: MorphSetupMeasurements (Section 8.3)

MorphRecalc

Description
Recomputes the statistical summary data for the current list of objects to be measured.

Syntax
MorphRecalc() As Long

Remarks
If any objects have been deleted (using MorphDeleteObject), those objects will be
eliminated from the measurement of summary data.

Example

' Delete object 4 from the morph summary data. This code
' assumes measurements have already been performed.
MM.MorphDeleteObject 4
MM.MorphRecalc

See also: MorphDeleteObject
8.6 Measuring Single Objects

Introduction

Data for individual objects are measured for all objects in an image simultaneously, but can be obtained and used on an object-by-object basis. This section describes the functions that you can use to obtain measurement data for an individual object. (For a discussion of group object measurement functions, see Section 8.5.) One programming example is given for all of the single object data “Get” functions (see Figure 8.1 on page 132).

MorphGetCentroid

Description

Obtains the coordinates of the centroid (point that represents the center of mass) of a specified object. The centroid will be given in fractional pixel coordinates.

Syntax

MorphGetCentroid(nObjectID As Integer, fRetX As Single, fRetY As Single) As Long

Remarks

This function differs from MorphGetCentroidPixel in its use of fractional coordinates. MorphGetCentroidPixel expresses coordinates as integer values by locating the pixel nearest to the “fractional” centroid.

Parameters

nObjectID Specifies the number of the object. Objects are numbered from 0 to (n – 1), where n is the total number of objects.

Return values

fRetX Returns the X-coordinate of the object centroid.

fRetY Returns the Y-coordinate of the object centroid.

See also:

MorphGetCentroidPixel

MorphGetCentroidPixel

Description

Obtains the coordinates of the pixel nearest to the centroid (the point that represents the center of mass) of a specified object.

Syntax

MorphGetCentroidPixel(nObjectID As Integer, nRetX As Integer, nRetY As Integer) As Long

Remarks

This function differs from MorphGetCentroid in that it expresses coordinates as integer values by locating the pixel nearest to the “fractional” centroid. MorphGetCentroid expresses the exact location of the centroid using fractional coordinates.

Parameters

nObjectID Specifies the number of the object. Objects are numbered from 0 to (n – 1), where n is the total number of objects.
8.6 Measuring Single Objects, continued

MorphGetCentroidPixel
(continued)

Return values

\[ fRetX \] Returns the X-coordinate of the pixel nearest to the object centroid.
\[ fRetY \] Returns the Y-coordinate of the pixel nearest to the object centroid.

See also: MorphGetCentroid

MorphGetClassifiersPassed

Description
Obtains the number of classifier filters that a specified object has passed.

Syntax
\[
\text{MorphGetClassifiersPassed}(nObjectID \text{ As Integer, } nRetClassifiersPassed \text{ As Integer}) \text{ As Long}
\]

Parameters

\[ nObjectID \] Specifies the number of the object. Objects are numbered from 0 to \( (n - 1) \), where \( n \) is the total number of objects.

Return values

\[ nRetClassifiersPassed \] Returns the number of classifier filters that the object has passed.

Example

' Find out how many classifiers object 9 passed
Dim n As Integer
MM.MorphGetClassifiersPassed 9, n

MorphGetClosestObject

Description
Obtains the number of the object closest to a specified set of X and Y coordinates.

Syntax
\[
\text{MorphGetClosestObject}(nX \text{ As Integer, } nY \text{ As Integer, } nRetObjectID \text{ As Integer}) \text{ As Long}
\]

Parameters

\[ nX \] Specifies the X-coordinate.
\[ nY \] Specifies the Y-coordinate.

Return values

\[ nRetObjectID \] Returns the object number of the object closest to the specified coordinates.

Example

' Get the object number of the object nearest to coordinate ' 100, 100
Dim obj As Integer
MM.MorphGetClosestObject 100, 100, obj

Continued on next page
8.6 Measuring Single Objects, continued

MorphGetInternalPoint

Description
Obtains the set of coordinates of some pixel within a specified object.

Syntax
MorphGetInternalPoint(nObjectID As Integer, nRetX As Integer, nRetY As Integer)
As Long

Parameters
nObjectID Specifies the number of the object. Objects are numbered from 0 to 
(n – 1), where n is the total number of objects.

Return values
nRetX Returns the X-coordinate of a randomly selected pixel in the object.

nRetY Returns the Y-coordinate of the same randomly selected pixel.

MorphGetMeasurements – for Visual Basic 6 & earlier

Description
Fills a defined array with all measurement data for a specified object.

Syntax
MorphGetMeasurements(nObjectID As Integer, aMeasurements() As Single) As Long
MorphGetMeasurementsEx2(nObjectID As Integer, aMeasurements() As Single)
As Long

Parameters
nObjectID Specifies the number of the object. Objects are numbered from 0 to 
(n – 1), where n is the total number of objects.

Return values
aMeasurements() Defines an array into which the object’s measurement data will be 
read. The array should have at least the number of elements returned by 
MorphGetNumberOfParms.

Example
' Get all the measurements for object number 12. This code 
' assumes measurements have already been made.
Dim n As Integer
MM. MorphGetNumberOfParms nParms
Dim measurements(nParms) As Single
MM. MorphGetMeasurements 12, measurements

See also:
MorphGetNumberOfParms (Section 8.3), MorphMeasureObjects (Section 8.5)

Continued on next page
8.6 Measuring Single Objects, continued

MorphGetNumberOfRuns

Description
Obtains the number of horizontal rows that compose a specified object.

Syntax
MorphGetNumberOfRuns(nObjectID As Integer, nRetNumberOfRuns As Integer) As Long

Parameters
nObjectID  Specifies the number of the object. Objects are numbered from 0 to 
            (n – 1), where n is the total number of objects.

Return values
nRetNumberOfRuns  Returns the number of horizontal rows in the object.

MorphGetNumberOfVertices

Description
Obtains the number of vertices that make up the edgelist of a specified object.

Syntax
MorphGetNumberOfVertices(nObjectID As Integer, nRetNumVertices As Integer) As Long

Parameters
nObjectID  Specifies the number of the object. Objects are numbered from 0 to 
            (n – 1), where n is the total number of objects.

Return values
nRetNumVertices  Returns the number of vertices in the object’s edgelist.

See also:
MorphGetVertexList

MorphGetObjectBoundingRect

Description
Obtains the coordinates of the upper left and lower right corners of a specified 
object’s bounding rectangle.

Syntax
MorphGetObjectBoundingRect(nObjectID As Integer, nRetX1 As Integer, nRetY1 
As Integer, nRetX2 As Integer, nRetY2 As Integer) As Long

Remarks
A bounding rectangle is a device used by MetaMorph to work with irregularly shaped 
objects. This contrivance is created by placing an imaginary rectangle over the 
object’s outline. The sides of the rectangle will be perfectly horizontal and vertical, 
and the smallest rectangle possible will be used. The MorphGetObjectBounding-
Rect function provides the coordinates of the starting point (upper left corner) and 
ending point (lower right corner).

Continued on next page
8.6  Measuring Single Objects, continued

**MorphGetObjectBoundingRect**

(continued)

**Parameters**

- *nObjectId*  Specifies the number of the object. Objects are numbered from 0 to 
  \((n - 1)\), where \(n\) is the total number of objects.

**Return values**

- *nRetX1*  Returns the X-coordinate of the upper left corner of the bounding rectangle.
- *nRetY1*  Returns the Y-coordinate of the upper left corner of the bounding rectangle.
- *nRetX2*  Returns the X-coordinate of the lower right corner of the bounding rectangle.
- *nRetY2*  Returns the Y-coordinate of the lower right corner of the bounding rectangle.

**See also:**  MorphGetVertexList

---

**MorphGetParmMeasurement**

**Description**

Obtains the measured value of a selected parameter for a specified object.

**Syntax**

```
MorphGetParmMeasurement(nObjectID As Integer, nParmNumber As Integer,
fRetMeasurement As Single) As Long
```

**Parameters**

- *nObjectId*  Specifies the number of the object. Objects are numbered from 0 to 
  \((n - 1)\), where \(n\) is the total number of objects.
- *nParmNumber*  Specifies the parameter number. This number can be obtained with 
  the MorphFindParmIndex function.

**Example**

```
' Find the width of object 5.  This code assumes that
' measurements have already been made.
Dim n As Integer
Dim nWidth As Single
MM.MorphFindParmIndex "Width", n
MM.MorphGetParmMeasurement 5, n, nWidth
MM.PrintMsg "The width of object 5 is " + Str(nWidth)
```

**Return values**

- *fRetMeasurement*  Returns the measured value of the selected parameter.

**See also:**  MorphFindParmIndex (Section 8.3)

*Continued on next page*
8.6  Measuring Single Objects, continued

MorphGetPixelArea

Description
Obtains the number of pixels in a specified object.

Syntax
MorphGetPixelArea(nObjectID As Integer, lRetNumberOfPixels As Long) As Long

Parameters
nObjectID  Specifies the number of the object. Objects are numbered from 0 to
            (n – 1), where n is the total number of objects.

Return values
lRetNumberOfPixels  Returns the number of pixels in the object.

MorphGetVertexList – for Visual Basic 6 & earlier

Description
Obtains the X and Y coordinates of all vertices in the edgelist of a specified object.

Syntax
MorphGetVertexList(nObjectID As Integer, aX() As Integer, aY() As Integer) As Long
MorphGetVertexListEx2(nObjectID As Integer, aX() As Integer, aY() As Integer) As Long

Remarks
returns the coordinates of the vertices making up the edge list of the given object. The
number of elements in the arrays aX and aY should at least the number of vertices in
the object. use MorphGetNumberOfVertices to obtain this number.

Parameters
nObjectID  Specifies the number of the object. Objects are numbered from 0 to
            (n – 1), where n is the total number of objects.

aX()  Defines a buffer into which the X-coordinates of the vertices will be read.

aY()  Defines a buffer into which the Y-coordinates of the vertices will be read.

Return values
aX()  The X-coordinates of the vertices will be read into this predefined buffer. (See
      Parameters.)

aY()  The Y-coordinates of the vertices will be read into this predefined buffer. (See
      Parameters.)

See also:  MorphGetNumberOfVertices

Continued on next page
For all the measured objects, print out some information obtained during measurement about each one. This code assumes object measurements were previously performed.

```visualbasic
Dim nObjects As Integer

MM.MorphGetNumberOfObjects nObjects

Dim i As Integer

For i = 0 To nObjects - 1
    MM.PrintMsg "Object " + Str(i) + " information:"

    Dim area As Long
    MM.MorphGetPixelArea i, area
    MM.PrintMsg " Pixel area is " + Str(area)

    Dim x As Single, y As Single
    MM.MorphGetCentroid i, x, y
    MM.PrintMsg " Centroid is at " + Str(x) + ", " + Str(y)

    Dim nx As Integer, ny As Integer
    MM.MorphGetCentroidPixel i, nx, ny
    MM.PrintMsg " Centroid pixel is at " + Str(nx) + ", " + Str(ny)

    MM.MorphGetInternalPoint i, nx, ny
    MM.PrintMsg " The coordinates of a point inside the object are " + Str(nx) + ", " + Str(ny)

    Dim n As Integer
    MM.MorphGetNumberOfVertices i, n
    MM.PrintMsg " The object has " + Str(n) + " vertices"

    Dim ax(n) As Integer, ay(n) As Integer
    MM.MorphGetVertexList i, ax, ay

    Dim j As Integer
    For j = 0 To n
        MM.PrintMsg " Vertex " + Str(j) + " is " + Str(ax(j)) + ", " + Str(ay(j))
    Next j
```

Continued on next page
8.6 Measuring Single Objects, continued

Single Object Data “Get” Function Programming Example
(continued)

```vbnet
MM.MorphGetNumberOfRuns i, n
MM.PrintMsg " There are " + Str(n) + " runs in the object"

Dim x1 As Integer, y1 As Integer, x2 As Integer, y2 As Integer
MM.MorphGetObjectBoundingRect i, x1, y1, x2, y2
MM.PrintMsg " The bounding rectangle coordinates are "
   + Str(x1) + ", " + Str(y1) + " and " + Str(x2) + ", "
   + Str(y2)
Next i
```
Index

—A—
Annotations
  Creating and assigning 54
  Reading 51
Area, measuring thresholded 106
Arrays 14
ASCII control codes 21
AutoEnhance 59
Autoscaling
  Current setting, reading 63
  Enabling and disabling 65
  Range maximum
    Configuring 66
    Reading 64
  Range minimum
    Configuring 67
    Reading 65

—B—
BinarizeImage 81
Bit-depth 51
Bounding rectangle 126
Brightness
  Autoenhancing 60
  Configuring 62
  Making changes permanent 60
  Reading the current setting 61
  Reverting to the previous setting 62

—C—
Centroids, obtaining coordinates 123
Class Modules 9
Classifiers
  Failed objects, drawing 111
  Measurement data, reading 119, 120
  Number of filters passed, determining for an object 124
  Settings
    Configuring 117
    Reading 116
CloneImage 37
CloseImage 38
Command Line text box 5
Contrast
  Autoenhancing 60
  Configuring 63
  Making changes permanent 60
  Reading the current setting 61
  Reverting to the previous setting 62
Coordinates
  Bounding rectangle, determining for 126
  Internal point of an object 125
  Nearest object, finding 124
  Vertices in an object, reading 128
CopyImage 38
CopyImagePlane 39
Copying images 37
CreateImage 39
CreateRectRegion 93

—D—
DestroyRegion 94
DIGetFirst 23
DIGetIOStatus 23
DIGetLineCount 24
DIGetLineState 25
DIGetName 25
DIGetNext 26
Digital I/O
  High vs. Low state, reading 25
  Input vs. output status of a line, reading 23
  Number of lines, reading 24
  Receiving signals from a device 27
  Sending signals to a device 26
DoCommand 6, 8

—E—
Edgelists
  Coordinates for a region, reading 101
  Vertices
    Coordinates, reading for an object 128
    Number of, reading 126
Exclusive thresholding 78, 79

—F—
FixImage 60
ForceCloseImage 40
Functions required by MetaMorph 7

—G—
GetActivePlane 51
GetActiveRegion 95
GetAutoScale 64
GetBrightness 61
GetContrast 61
GetCurrentImage 44
GetDepth 51
GetFunctionHandle 30
GetHeight 51
GetImage 44
GetImageAnnotation  52
GetImageName  52
GetImageWindowPosition  47
GetImageWindowSize  47
GetLut  69
GetLutModel  70
GetMaxScale  65
GetMinScale  66
GetNumberOfImages  45
GetNumberOfPlanes  53
GetNumberOfRegions  96
GetRegion  96
GetRegionArea  99
GetRegionAverageValue  105
GetRegionDistance  99
GetRegionMaximumValue  106
GetRegionMinimumValue  105
GetRegionPosition  100
GetRegionSize  100
GetRegionStdDeviation  106
GetRegionThresholdArea  107
GetThresholdRange  77
GetThresholdState  78
GetWidth  53
GetZoom  53

Handles, finding  44
Height, reading  51
Loading  41
Name, reading  52
New images, creating  39
Number of loaded images, finding  45
Overwriting  38
Renaming  56
Saving  41
Selecting for measurement  115
Setting the timestamp  56
Updating display  59
Width, reading  53

Inclusive thresholding  78, 79

Intensity

Average  105
Maximum  106
Measuring  107
Minimum  105
Reading
  From a column of pixels  82, 83
  From a row of pixels  84, 85
  From a single pixel  84
Standard deviation  106
Writing values
  To a column of pixels  86, 87
  To a row of pixels  89, 90
  To a single pixel  88

IsValidImage  46
IsValidRegion  97

Journals, running  33

Keep Program in Memory After Execution check box  6

Labeling an image  91
LoadImage  41

Look-up tables

Assigning a LUT model  73
Current LUT model in use, determining  70
Reading a table’s elements  69
Writing to a table’s elements  72

Mask images  115
MaximizeImageWindow  48

Measurement

Centroids  123

Data
Filling an array  125
Reading  119, 120
Measuring objects in an image  121
Number of pixels in an object, measuring  128
Parameter data, reading for an object  127
Recalculating  122
Vertices in an object, reading  128
Measurement images  115
MeasureRegion  107
Message windows
  Configuring size and position  34
  Displaying  30
MetaDevices
  Obtaining a handle  23
  Obtaining a name from a handle  25
MinimizeImageWindow  49
MM variable  7
MorphDeleteObject  118
MorphFindParmIndex  112
MorphGetCentroid  123
MorphGetCentroidPixel  123
MorphGetClassifiersPassed  124
MorphGetClosestObject  124
MorphGetFilter  116
MorphGetInternalPoint  125
MorphGetMeasurements  125
MorphGetNumberOfObjects  118
MorphGetNumberOfParms  113
MorphGetNumberOfRuns  126
MorphGetNumberOfVertices  126
MorphGetObjectBoundingRect  126
MorphGetParmAccumSummary  119
MorphGetParmDescription  113
MorphGetParmMeasurement  127
MorphGetParmName  114
MorphGetParmSummary  120
MorphGetPixelArea  128
MorphGetVertexList  128
MorphMeasureObjects  121
MorphRecalc  122
MorphSetDrawFailedObjectsFlag  111
MorphSetFillHoleFlag  111
MorphSetFilter  117
MorphSetupMeasurements  115

—N—
New images, creating  39

—O—
Object linking and embedding  7
Objects
  Measuring  121
  Nearest object, finding  124
  Number last measured, finding  118
  Parameter data, reading  127
  Pixels in an object, measuring  128
  Remeasuring  122
  Removing from a measurement list  118
  Rows in an object, measuring  126
  Vertices, determining number of  126
OLE  7

—P—
Palettes, configuring number of entries  74
Parameters
  Description, obtaining  113
  Name, obtaining  114
  Number of measurable parameters, determining  113
  Parameter number, determining from a parameter's name  112
Perimeter, determining for a region  99
Planes
  Active plane, setting  55
  Copying  39
  Number of planes in stack, reading  53
  Plane number, reading  51
Preferences
  Failed objects, drawing  111
  Holes, filling  111
PrintMsg  30
Program Name drop-down list  5
Public variants  9

—R—
ReadColumn  82
ReadColumnEx  83
ReadPixel  84
ReadRow  84
ReadRowEx  85
RegionGetEdgePixelCoordinates  101
RegionGetNumEdgePixels  101
Regions
  Active region, setting  98
  Area, determining  99
  Creating  93
  Handles, finding  95, 96
  Intensity, measuring  107
  Number of regions, determining  96
  Numbers, switching  104
  Perimeter, determining
    Calibrated units  99
    Pixel units  101
  Position
    Configuring  103
    Reading  100
  Removing  94
  Size
    Configuring  103
    Reading  100
  Thresholded area, measuring  107
  Remove command button  6
ResetContrast 62
Run User Program
  Dialog box 5
  Options 5
RunFunction 31
RunFunctionEx 32
RunJournal 33
Running a function 31
Running a journal 33
RUNUSER drop-in 5

—S—
SaveImage 41
SendSerialData 18
Serial communication
  ASCII control codes 21
  Receiving a data stream from a device 19
  Sending a data stream to a device 18
  Syntax rules 20
Set 9
SetActivePlane 55
SetActiveRegion 98
SetAutoScale 66
SetBrightness 62
SetContrast 63
SetDigitalIO 26
SetDisplayImagesWhenCreated 42
SetFunctionVariable 33
SetImageAnnotation 55
SetImageName 56
SetImageTimestamp 56
SetImageWindowPosition 49
SetImageWindowSize 50
SetLut 72
SetLutModel 73
SetMaxScale 67
SetMinScale 68
SetNumPaletteEntries 74
SetPrintMsgWindowPositionAndSize 34
SetRegionPosition 103
SetRegionSize 103
SetThresholdRange 79
SetThresholdState 79
SetZoom 57
ShowImage 43
Shutdown 6, 8
Standard area 115
Startup 6, 8
Summary data, recalculating 122
SwapRegionNumbers 104

—T—
Text, writing on an image 91
Thresholding
  Area, measuring thresholded 107
  Holes, filling 111
  Range
    Configuring 79
    Reading 77
  State
    Configuring 79
    Reading 78

—U—
UpdateDisplay 59
User programs
  Compiling 12
  Creating
    With Visual Basic 4.0 10
    With Visual Basic 5.0 11
  Overview 7
  Registering and unregistering 6
UserMethods 9

—V—
Variables required by MetaMorph 7
Variables, setting values for 33
Vertices
  Determining number in an object 126
  Reading coordinates from an object 128

—W—
WaitForDigitalIO 27
WaitForSerialData 19
Width, image 53
WriteColumn 86
WriteColumnEx 87
WritePixel 88
WriteRow 89
WriteRowEx 90
WriteText 91

—Z—
Zoom factor
  Configuring 57
  Reading 53