Eastman Software
Imaging

Professional Developer's Guide Update

August, 1999

Eastman Software, Inc., 296 Concord Road ■ Billerica, MA 01821-4130
Tel. (978) 313-7000 ■ http://www.kodak.com and www.eastmansoftware.com

Introduction

This document contains corrections and new information for the Eastman Software Imaging *Professional Developer's Guide*, part number 715-B017. The information is presented sequentially to correspond to the chapters in the book.

Chapter 3 — Automation Lexicon

Page 80

Remarks section

In the last sentence, the phrase "value is from 64 to 16384" should read "value is from 64 to 16392."

Page 87

PageRange Object Methods

Several entries in the Parameter, Data Type, and Description table are incorrect.

- The description for the StartPage parameter should read "First page in the range."
- The NumPages parameter should be changed to EndPage, and the description for the parameter should read "Last page in the range."

The corrected table is shown below.

Parameter	Data Type	Description
StartPage	Long	First page in the range.
EndPage	Long	Last page in the range.

Chapter 6 — Image-Enabling Web Pages

Page 246

Windows Explorer

Add the following note to this section:

To access the 1.x (NT) document manager from a Web browser, use the following syntax:

```
Image://server\database\cabinet\drawer\folder\document
```

Chapter 7 — Image Edit/Image Annotation Control

Page 322

ImageModified Property

In the Remarks section, the list of methods that set the **ImageModified** property to True should include the **LoadAnnotations** method.

Page 433

PrintImage Method

An entry in the Parameter, Data Type, and Setting table is incorrect. The data type for the OutputFormat parameter should be "Long (enumerated)", not "Integer (enumerated)."

Page 521

SelectionRectDrawn Event

In the Description section, the phrase "This event occurs immediately after the end user presses the left mouse button ..." is incorrect. It should read, "This event occurs immediately after the end user releases the mouse button after drawing a selection rectangle ..."

Chapter 8 — Image Admin Control

Page 552

CancelError Example - VC++

The following sample code replaces the example in the book.

```
void CAdmin1Dlg::OnPrint()
{
     // This will use the ShowPrintDialog method to allow
     // the user to specify parameters for printing. This
     // example will print the image that is displayed in
     // ImgEdit control.


    // Display an image
    ImgEdit1.SetImage ("D:\\image2\\4page.tif");
    ImgEdit1.Display();


     // Reset NumCopies in case user printed multiple copies
     // last time.
    ImgAdmin1.SetPrintNumCopies (1);


     // If CancelError is true, an error is generated if user
     // presses Cancel.  Trap the error to avoid trying to
     // print the file.
    ImgAdmin1.SetCancelError (TRUE);


     // Set filename to be printed to the displayed file.
     // If this property is not set the dialog box will
     // not display.
    ImgAdmin1.SetImage (ImgEdit1.GetImage());
    VARIANT vhWnd; V_VT(&vhWnd) =  VT_I4;
  V_I4(&vhWnd) = (long)m_hWnd;


  // Use try and catch around ShowPrintDialog method to
  //catch Cancel error.
  // If Cancel is pressed (or another error occurs), catch
  // will capture the error exception.
  try
  {
     ImgAdmin1.ShowPrintDialog (vhWnd);
```

```
    // Print the image using the parameters obtained from
    // the print dialog box (ex. start page, end page etc.).
    VARIANT vStart, vEnd, vOutputFormat, vAnnotations, evt;
    evt.vt = VT_ERROR;// set to error for optional parameter

    V_I4(&vStart) = ImgAdmin1.GetPrintStartPage();
    V_I4(&vEnd) = ImgAdmin1.GetPrintEndPage();
    V_I4(&vOutputFormat) =
    ➥ ImgAdmin1.GetPrintOutputFormat();
    V_I4(&vAnnotations) = ImgAdmin1.GetPrintAnnotations();
    ImgEdit1.PrintImage (vStart, vEnd, vOutputFormat ,
    ➥ vAnnotations, evt, evt, evt);
  }

  catch(COleDispatchException* e)
  {
    // Handle error in Print Dialog or in actual printing of
    // the image
    CString CStrErr;
    CStrErr += "Error Description: " + e->m_strDescription;
    AfxMessageBox( CStrErr );
  }
}
```

Page 649

GetUniqueName Method

The following sample code replaces the example in the book.

```
cmdUniqueFile_Click()

  Dim strNewfile As String
  Dim strPath As String
  'Pass the directory, template, and file extension for
  'the new file
  'strPath = "c:\Images" example of local image path
  strPath = "Image://Groucho\demo:"
  strNewfile = ImgAdmin1.GetUniqueName(strPath, "test",
  ➥ "tif")
```

```
  With ImgEdit1
    .Image = strPath & "\" & strNewfile
    .DisplayBlankImage .Width, .Height
    .SaveAs (strPath & "\" & strNewfile)
  End With
End Sub
```

Pages 651, 219, 220

Use of Delimiters

In some cases, using a semicolon (;) as a delimiter does not work correctly. If you have encountered this problem, substitute a space for the semicolon. Refer to the following places in the book where the semicolon is used:

- page 651, under the **See Also** heading: "findfolders cabinet=*cabinetname*;drawer=*drawername*"
- page 219, in the description of the szQueryTerms parameter: "findfolders cabinet=" & mstrCabinet &";drawer=" & ...
- page 220, in the example:
  `mstrQuery = "findfolders cabinet=" & mstrCabinet &" ;drawer=" & ...`

Page 653-654

ImgQuery Method, VB Example

The following sample code replaces the example in the book.

```
Private Sub cmdQuery_Click()
    Dim objResults As Object
    Dim varItem As Variant
    Dim Msg As String
'Perform the query and put results in objResults.
    On Error GoTo ImgQueryErr
    ImgAdmin1.ImgQuery("srvrname\test_db:", "finddocs
    ➥ keyword = insurance", objResults)
    'Write results of the query to a listbox.
    For Each varItem In objResults
      lb_results.AddItem varItem
    Next
```

```
        'Release memory and resources.
        Set objResults = Nothing
        ImgAdmin1.ImgQueryEnd
    Exit Sub
ImgQueryErr:
    'Display error message
    Msg = "ImgQuery method failed" & vbCRLF & _
        "Error Number: " & Err.Number & vbCRLF & _
        "Error Description: " & Err.Description
    MsgBox Msg, vbExclamation
End Sub
```

Chapter 9 — Image OCR Control

Page 687

Image Property

To the **Remarks** section, add the following comment:

"The image defined in this property must be stored in a file, or the OCR will fail. For example, if you scan an image to display only, you must save it as a file before you can perform an OCR."

Chapter 10 — Image Scan Control

Page 744

Zoom Property

In the **Data Type** section, the value should be "Single" instead of "Long."

Page 767

SetPageTypeCompressionOpt Method

In the **Returns** section, the value should be "Integer" instead of "None." Possible Return values are:

Value	Description
0	Success
1	Invalid Compression Preference
2	Invalid Image Type
3	Invalid Compression Type
4	Invalid Compression Info

Page 770

SetScanCapability Method

In the **Returns** section, the value should be "Long" instead of "None." Possible Return values are:

Value	Description
0	Success
3	Bad capability
4	Invalid array passed in. Possible problems could be: – must be an array of floats – must be a one dimensional array – failed call – must have a lower limit of 0 – failed call – must have an upper limit of 4
5	Bad data value

Appendix B— Imaging ActiveX Tips and Tricks

Page 865

The current section, "**Catch errors properly when working with the ShowFileDialog method**," is replaced with the following section.

Catch errors properly when working with the Active X Controls

All the OCX controls (Image Admin, Image Edit, Image Annotation Tool Button, Image OCR, Image Scan and Image Thumbnail) and COM objects (Image Server Access) throw errors when a problem occurs. Unless these errors are properly handled, they will cause your program to quit. As an example, consider the Image Admin OCX control's **ShowFileDialog** method.

When invoked, **ShowFileDialog** displays an Open or Save As dialog box to end users. When users click Cancel on one of these dialog boxes, a "Cancel button is pressed" error condition occurs. Other Image Admin error conditions can also occur as the procedure containing the **ShowFileDialog** method continues its processing.

It is important to understand the difference between catching the "Cancel button is pressed" error condition and any other Image Admin (or other Active X control) error condition that may occur.

Note: While the examples below catch the "Cancel button is pressed" error condition to illustrate the appropriate techniques for catching errors, you do not need to catch this particular error if you do not wish to do so. To prevent this error from being thrown, see the Image Admin CancelError property.

In Visual Basic — The following code snippet from the Eastman Sample application demonstrates the recommended way of handling ShowFileDialog error conditions, as well as those for all the Imaging Active X controls and COM objects. To trap any errors, put an On Error statement such as the one below before the control code in question. This particular statement says that the code should resume processing at the statement labeled OCXError. (Other On Error options are available. See the Visual Basic help text or any good book on Visual Basic).

```
'Trap errors that may be thrown by the OCX
   On Error GoTo OCXError

   'Display open dialog
   kdkImgAdmin1.InitDir = "C:\My Documents"
   kdkImgAdmin1.Image = ""


   'Set this property to False so no error is thrown
   'when Cancel is pressed
   'kdkimgadmin1.CancelError = False
```

```
    kdkImgAdmin1.ShowFileDialog OpenDlg, Form1.hWnd

    'Display image selected from open dialog
    kdkImgEdit1.Image = kdkImgAdmin1.Image
    kdkImgEdit1.Display
    Exit Sub

OCXError:
' If the Cancel button was pressed, or if a different
    'error occurred, declare a message box and exit the
    'subroutine.
    MsgBox "Error: " + Str(Err.Number) + vbCrLf + _
        "Description: " + Err.Description + vbCrLf + _
        "ImgAdmin error: " + Hex(kdkImgAdmin1.StatusCode), _
        ➥ vbCritical, "Error"
     Exit Sub
```

When users click Cancel on the Open or Save As dialog box while executing the **ShowFileDialog** method, an error is thrown. Based on the instructions in the On Error statement, the code resumes execution at the line labeled OCXError. The Number property in Visual Basic's Err object contains the literal value for the "Cancel button is pressed" error condition. The Description property of the Err object contains a text description of the error. The **StatusCode** property is the error code returned by the Image Admin control. In this case the "Cancel button is pressed" error, causes a dialog to be displayed followed by an Exit from the subroutine.

In Visual C++ — Errors thrown by the Active X controls are handled in similar fashion in Visual C++ using try….catch logic. The following code snippet shows how this is done. The code to display the Open dialog box as well as the selected image resides inside of a try block. If the user presses Cancel in the Open dialog or commits another trappable error, the code jumps to the catch block where an error message is displayed.

```
// Trap errors that may be thrown by the OCX
  try
  {
    // Display open dialog
    m_ImgAdmin.SetInitDir( "C:\\My Documents" );
    m_ImgAdmin.SetImage( "" );

    // Set this property to False so no error is thrown
    // when Cancel is pressed.
    // m_ImgAdmin.SetCancelError( false );

    VARIANT vhWnd;
    V_VT( &vhWnd ) = VT_I4;
    V_I4( &vhWnd ) = ( long )m_hWnd;

    m_ImgAdmin.ShowFileDialog( 0, vhWnd );

    // Display the image selected from open dialog
    m_ImgEdit.SetImage( m_ImgAdmin.GetImage() );
    m_ImgEdit.Display();
  }

  catch ( COleDispatchException *e )
  {
    // Handle error in Open dialog or in displaying
    // the image
    CString CstrErr;// strings to hold error message

    CstrErr += "Description:     " + e->m_strDescription;
    AfxMessageBox( CstrErr );
}
```