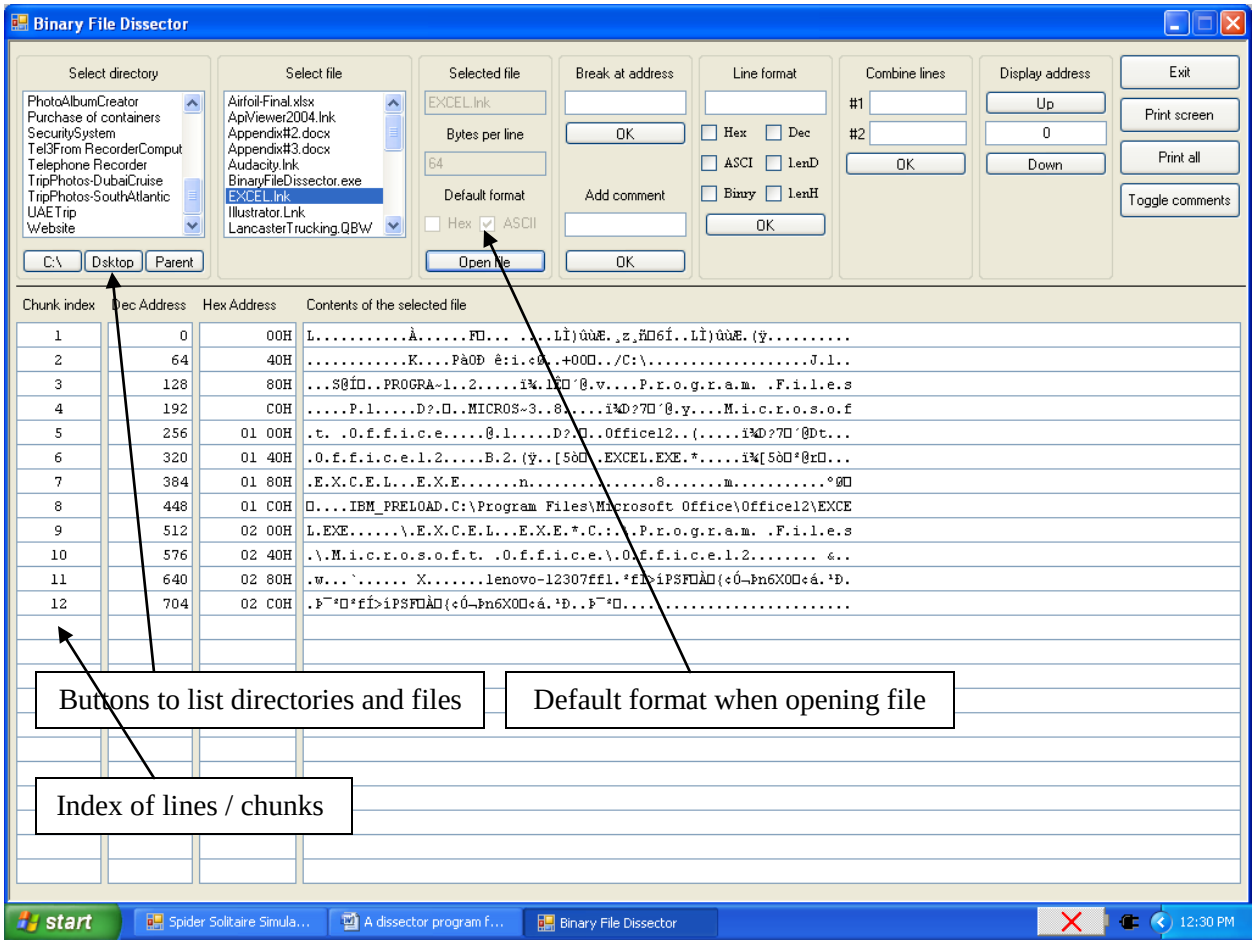
A Visual Basic program for dissecting binary files

If one spends much time looking at the contents or structure of binary files, there comes a time when it is useful to have a viewer which can do more than display the contents of the bytes in hexadecimal format. This paper describes such a program. The program is a single-form application written in Visual Basic 2010. The following screenshot shows the display at some point while viewing a shortcut (a .lnk file) from my desktop. In this case, the complete name of the file is "C:\Documents and Settings\User\Desktop\EXCEL.lnk".



The three control buttons at the upper-left, "C:\", "Dsktop" and "Parent", are used to navigate to the directory which contains the file of interest. They do as their names suggest and open the selected directory, whether the "C:\" drive, the desktop or the directory which is the parent of the directory being viewed, respectively. When a directory is selected, the listbox on the left, labeled "Select directory", lists the sub-directories in the directory and the listbox on the right, labeled "Select file", lists the files. Clicking on a sub-directory in the "Select directory" listbox drills down into the selected directory and shows its sub-directories and files on the left and right, respectively. Clicking on a file in the "Select file" listbox selects the file selected, but does not yet open it.

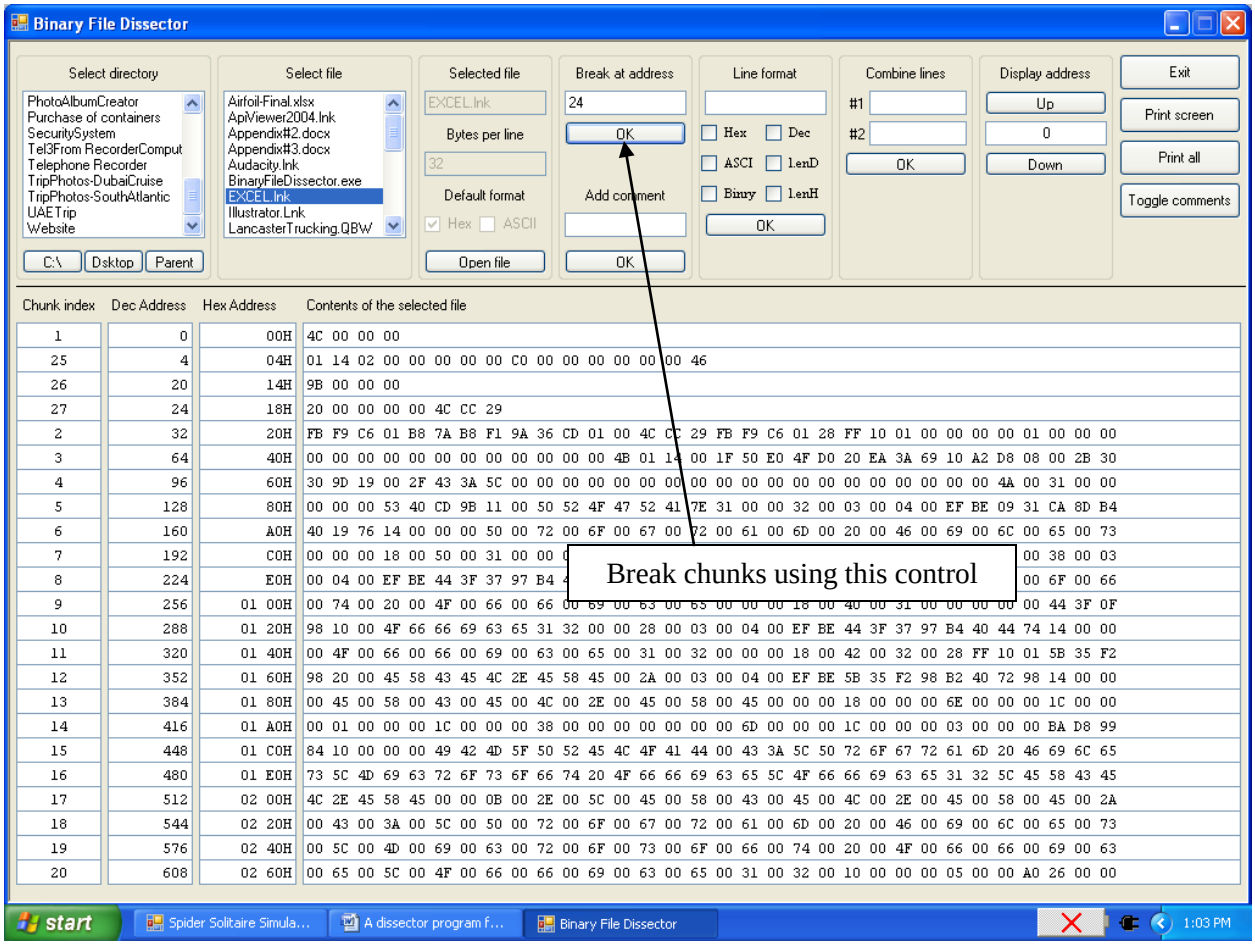
The next group of controls to the right, labeled "Selected file" will set out the name of the file to be opened. In the case shown, the short name of the file to be opened is "EXCEL.lnk". Beneath the name are the settings which will be used when the file is opened. As can be seen, I opened EXCEL.lnk with parameters which display 64 bytes per line in ASCII format. Bytes which are not printable are displayed

as periods. You can see from the text shown some of the data which the shortcut file uses to direct the computer's attention to the Excel program which it opens when you double-click on the shortcut icon.

The contents of the file are shown line-by-line or, as the index on the left-hand side of the main display area is labeled, chunk-by-chunk. In the program, I refer to the contents of a line as a "chunk". The next two columns of the main display area set out the address in the file of the first byte shown on that line. The addresses are shown in decimal format and then again in hexadecimal format. The bytes in a file are indexed from zero.

Incidentally, it will be observed that this file is not very long. 12 chunks are shown, each being 64 bytes long, so that a total of 768 bytes are shown. The file "Excel.lnk" happens to be 745 bytes long. The program has padded the last chunk with periods.

The whole point of a binary file dissector is to be able to make sense of the contents of the file. Much of the data contained in this file is not in ASCII format. The following screenshot shows the same file again, this time having been opened with 32 bytes per chunk and the display in hexadecimal format.

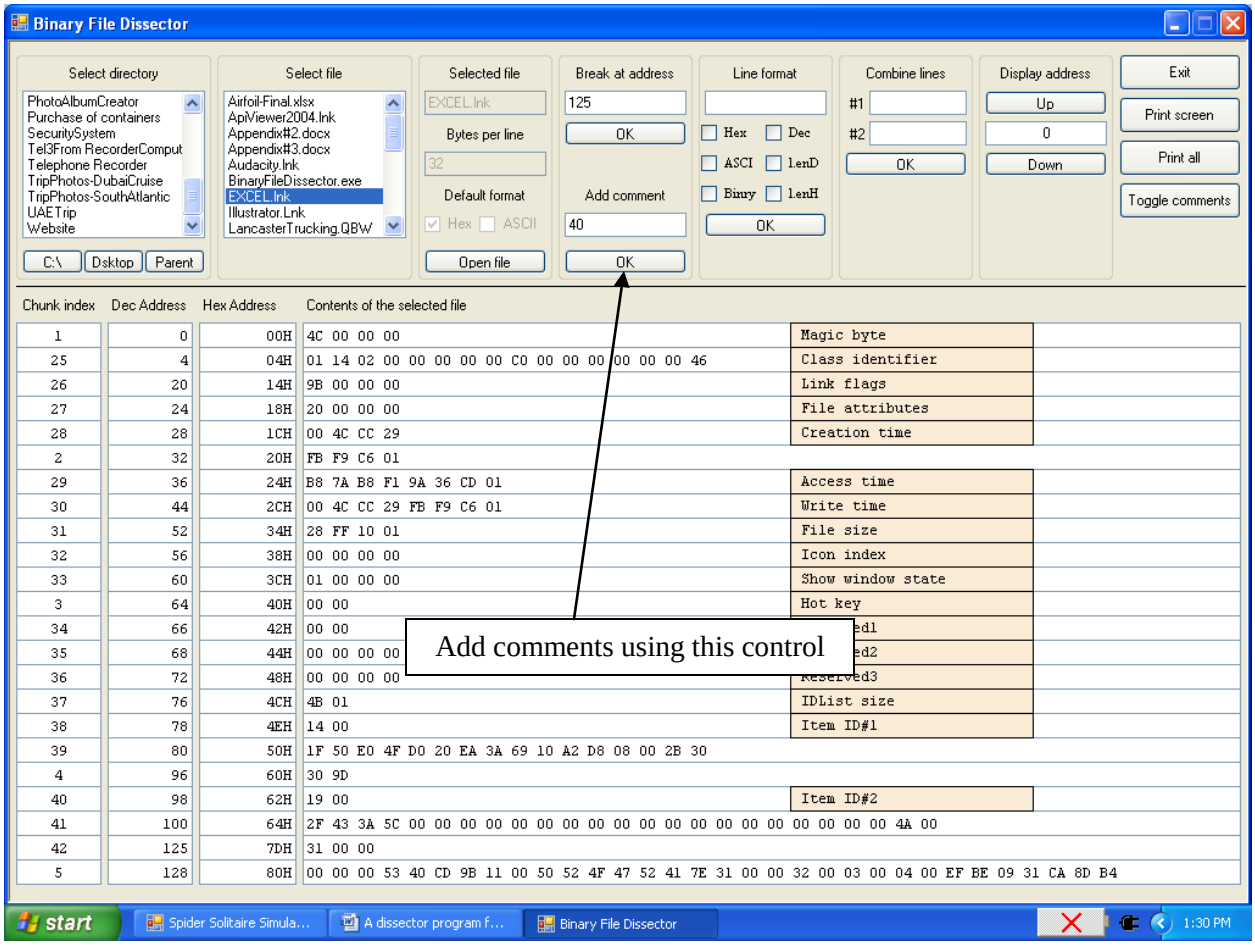


A stream of hexadecimal is even less informative than a stream of ASCII characters, but it does set the stage for cutting things up. For that purpose, the control labeled "Break at address" is used. I used it three times to produce this display: once at decimal address "4", a second time at address "20" and a third time at address "24". I did this because I know that the first four bytes of a shortcut file are what is called the "Magic" byte, which in this case identifies the file as a shortcut, or link. The next 16 bytes are the "Class identifier". For a shortcut file, the "Class identifier" must be exactly as shown. The next set of

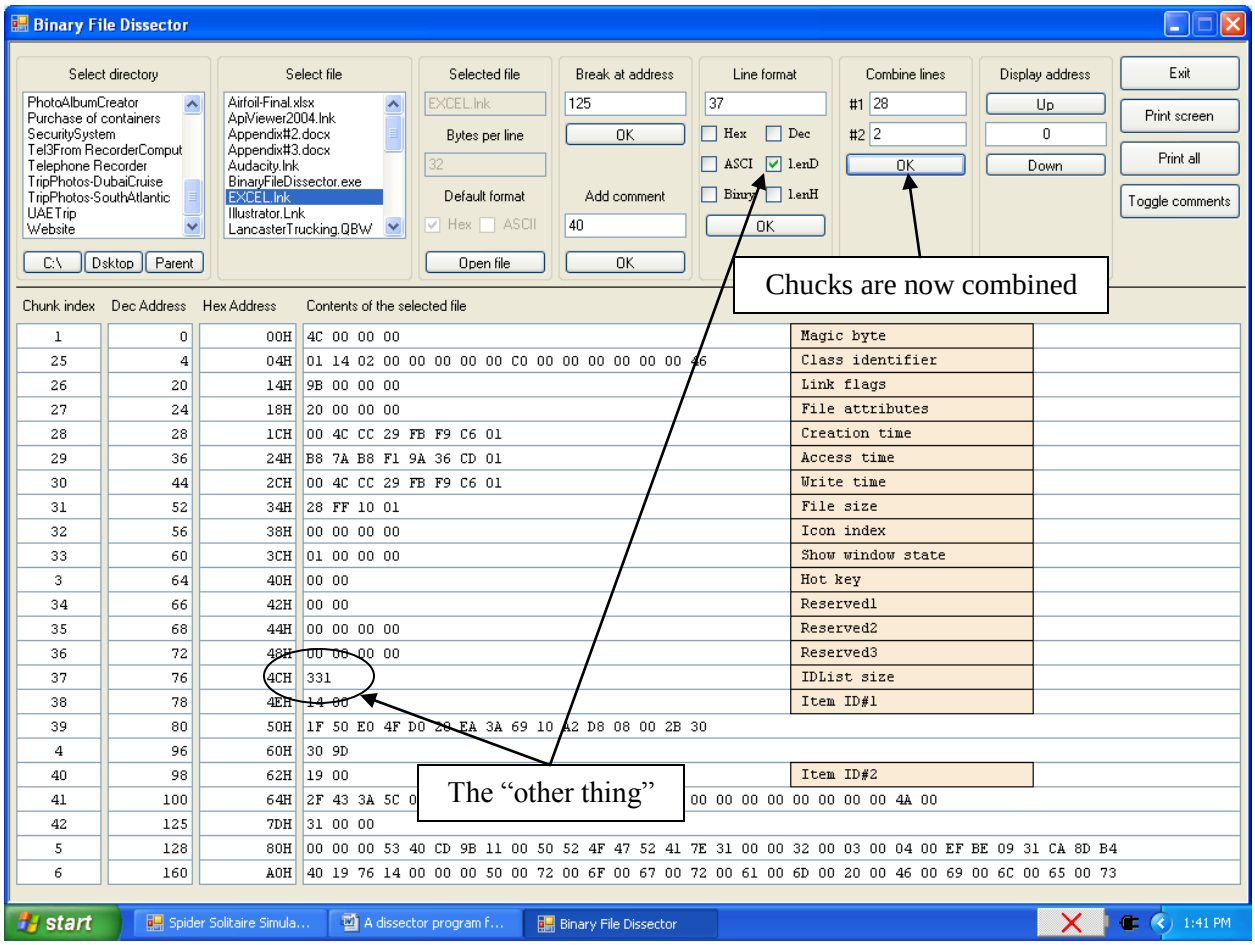
four bytes in a shortcut file are the “Link flags”, which is a set of 32 bits which describe what else follows in the file. Before looking at those in more detail, look at the fourth line, which now contains eight bytes. This happens to be the tag-end of the first chunk from when the file was opened. There were 32 bytes in the first chunk. I broke them up into three groups containing four bytes, 16 bytes and four bytes, respectively. That left eight bytes from the first chunk, which now comprise their own chunk.

The “chunk indices” in the left-most column tell the history of what happened. When the file was opened, its contents occupied 24 chunks (24 chucks multiplied by 32 bytes per chunk equals 768 bytes). When I made the first break, at address “4”, a new chunk was created to contain the bytes from address 4 through address 32. The new chunk was given index number 25. Another chunk (index number 26) was required when the “Class identifier” was segregated and a third (index number 27) was required for the “Link flags”. The remnant of the original chunk is now in that last chunk.

If there are many fields to be segregated, it soon becomes useful to be able to add a short description of each chunk. The following screenshot was taken after I continued to break out the fields in the header of this shortcut file. I added comments using the control identified by the arrow in the following screenshot.



Sometimes it is handy to be able to re-combine chunks. For example, the “Creation time” is supposed to be an eight-byte field, just like the “Access time” and the “Write time”. Because I opened the file using 32 bytes per chunk, the “Creation time” is now spread over two chunks. That is remedied using the control labeled “Combine lines”. After combing those two chunks (and doing one more thing), the screen looks like the following.



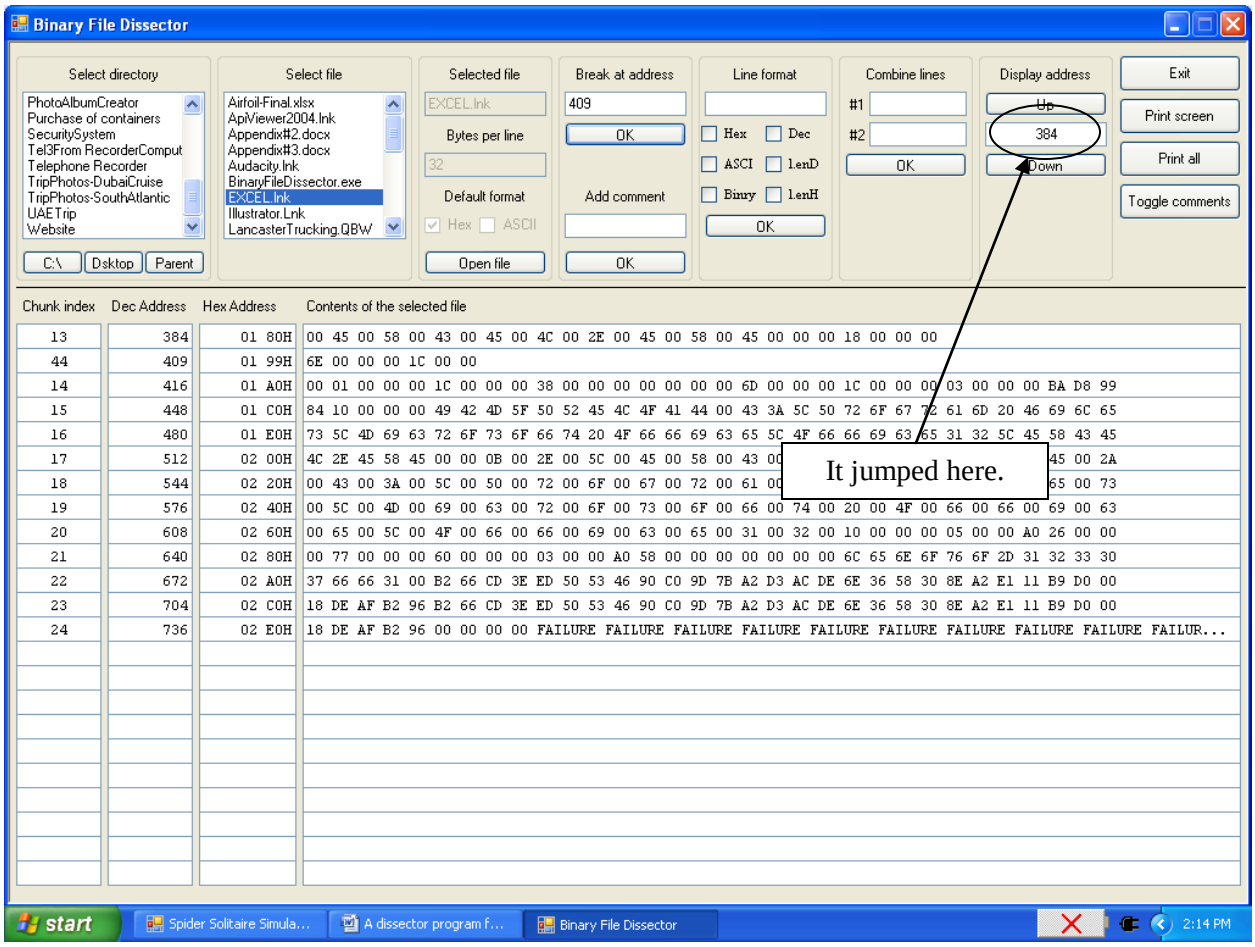
Let us look at the "other thing" I did. In a shortcut file, the information which starts at hexadecimal address 4C is the "IDList". The first field in that list is a two-byte field which contains the length of the "IDList", measured in bytes, not including the two bytes of the length field. The program can easily convert the format used to display a chunk. There are six available formats. We have already seen typical ASCII and Hex displays. In this screen, the "IDList size" have been displayed in something called "l.enD". This is a short way of describing "little-endian decimal". The little-endian format dates from the early days of personal computers and the culprit behind it is Intel Corporation. Little-endian gets its name because the less significant information (the "little end") in a sequence comes first. For example, the two bytes at hexadecimal address 4C are placed in the file (and stored in memory) in the order 4B 01. But, the meaningful order of the pair of bytes is 014B, which can be expanded as the sum of: (i) one 256, (ii) 4 16's plus (iii) B, or 11, ones, for a total of 256 + 64 + 11 = 331.

The four interesting formats available in the program are:

- "l.enD" is little-endian decimal,
- "l.enH" is little-endian hexadecimal,
- "Dec" is normally ordered decimal and
- "Binary" is ones and zeros (in the order in which they are stored).

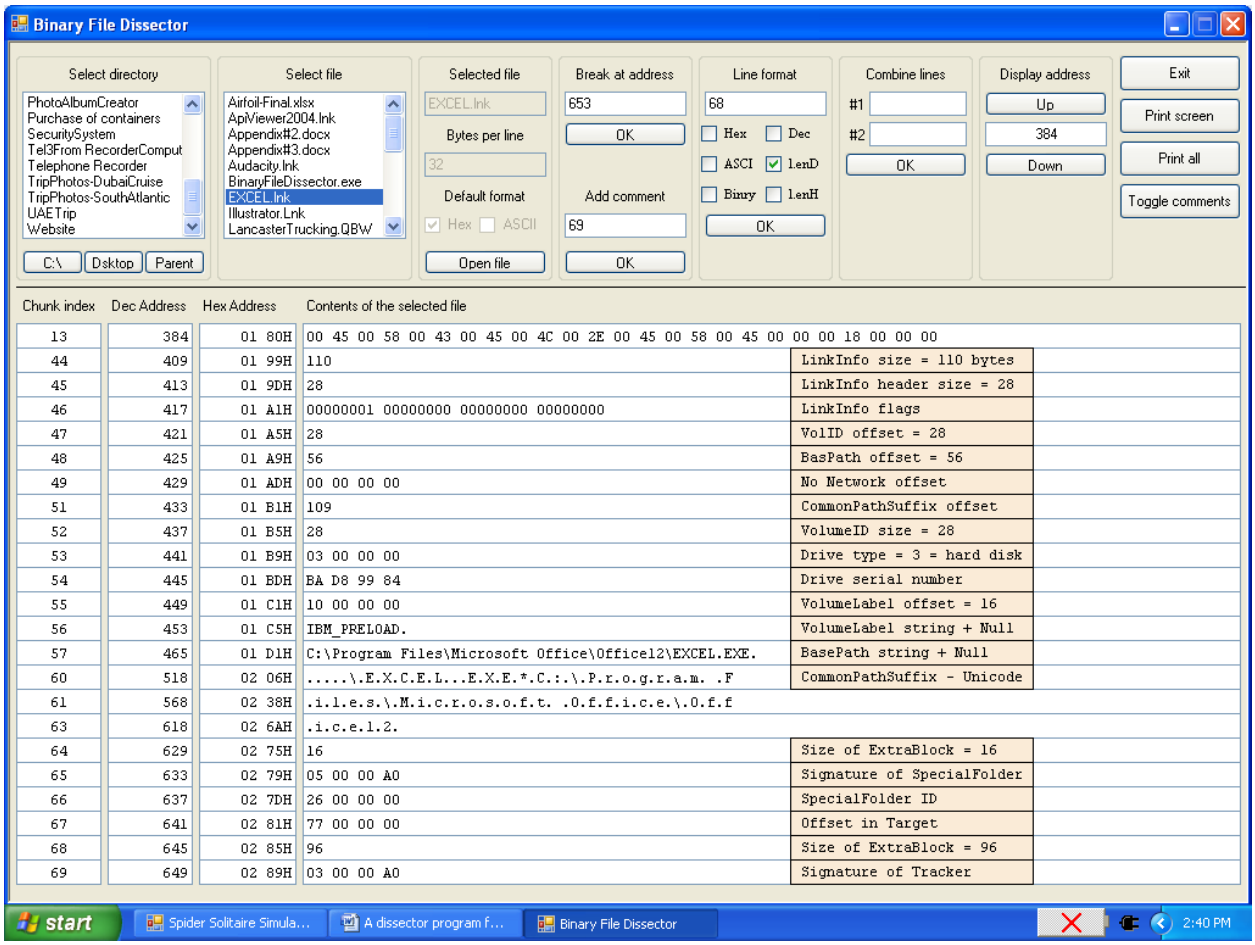
Now that we know that the "IDList" in the shortcut "EXCEL.lnk" is 331 bytes long, we can easily determine where it ends. Since the "IDList" starts at decimal address 78, the information which follows the "IDList" will start at decimal address 78 + 331 = 409. Let us skip down to that address. We can use

the controls labeled “Display address” in the upper-right of the screen. The buttons “Up” and “Down” move the focus up and down one line, respectively. Or, type an address into the textbox and hit Enter to skip to a desired address. The following is a screenshot taken after I skipped down to decimal address 409.



Although I typed in 409, the program jumped the display to decimal address 384. That is the address of the chunk containing the requested address. (Incidentally, there is no need to fret over the “FAILURE”s in the last line. They arise because the file came to an end, so there were no bytes with which to fill the last chunk.)

The information in “EXCEL.lnk” which starts at decimal address 409 is the “LinkInfo” structure. After breaking the structure into fields and formatting them in a useful way, the screen appears as follows. It can be seen that some “ExtraBlocks” follow the “LinkInfo” structure, including a “SpecialFolder” and a “Tracker”.



That completes the description of the controls, except for the buttons in the upper-right corner.

- "Print screen" will print the screen on the default printer.
- "Print all" will print the entire file, not just that portion which can be seen, on the default printer.
- "Toggle comments" hides the comments so that one can see the file contents which they cover up and, when clicked again, brings the comments back.

The following pages give a listing of the Visual Basic program.

Jim Hawley
July 2012

An e-mail setting out errors and omissions would be appreciated.

```
Option Strict On
Option Explicit On

'////////////////////////////////////////////////////////////////////////////////
'// Dissector for binary files.
'////////////////////////////////////////////////////////////////////////////////

Public Class Main
    Inherits System.Windows.Forms.Form

    Private SCREENWPIXEL As Int32 = 1024            ' Width of screen in pixels
    Private SCREENHPIXEL As Int32 = 740             ' Height of screen in pixels
    Private CDIRECTORY As String = "C:\"
    Private DESKTOPDIRECTORY As String = "C:\Documents and Settings\User\Desktop\"
    Private CurrentDirectory As String
    Private CurrentFileName As String
    Private MAXNUMCHUNKS As Int32 = 10000           ' Chunks numbered from 1 to 10,000
    Private NumberOfChunks As Int32
    Private MAXNUMBYTESINFILE As Int32 = 10000001   ' 10MB maximum file size
    Private NumberOfBytesInFile As Int32
    Private MAXNUMBYTESPERCHUNK As Int32 = 10001    ' Bytes numbered from 0 to 10,000
    Private DEFAULTNUMBYTESPERCHUNK As Int32 = 256  ' Bytes numbered from 0 to 255
    Private DEFAULTFORMAT As Int32 = 1              ' Hex format
    Private NumberOfBytesInChunk As Int32
    Private MainInputStream As System.IO.FileStream
    Private MainReader As System.IO.BinaryReader
    Private MAXNUMLINESINDISPLAY As Int32 = 23
    Private ShowComments As Boolean   ' True if comments should be displayed

    Public Chunk(MAXNUMCHUNKS) As Chunks
    Public Structure Chunks
        Public Active As Boolean      ' False=retired from service
        Public StartAddress As Int32
        Public NumBytes As Int32
        Public EndAddress As Int32
        Public ByteStream() As Int32
        Public DisplayFormat As Int32 ' 1=Hex, 2=ASCII, 3=Bin, 4=Dec, 5=l.endD, 6=l.endH
        Public Comment As String
    End Structure

    Public Sub New()
        InitializeComponent()
        With Me
            Name = "Main"
            Text = "Binary File Dissector"
            FormBorderStyle = Windows.Forms.FormBorderStyle.Fixed3D
            Size = New Drawing.Size(SCREENWPIXEL, SCREENHPIXEL)
            CenterToScreen()
            MinimizeBox = True
            MaximizeBox = False
            Controls.Add(groupboxDirectories)
            groupboxDirectories.BringToFront()
            Controls.Add(groupboxFiles)
            groupboxFiles.BringToFront()
            Controls.Add(groupboxSelectedFile)
            groupboxSelectedFile.BringToFront()
            Controls.Add(groupboxBreakChunk)
            groupboxBreakChunk.BringToFront()
```

```
            Controls.Add(groupboxChunkFormat)
            groupboxChunkFormat.BringToFront()
            Controls.Add(groupboxCombineChunks)
            groupboxCombineChunks.BringToFront()
            Controls.Add(groupboxDisplayStartAddress)
            groupboxDisplayStartAddress.BringToFront()
            Controls.Add(buttonExit) : buttonExit.BringToFront()
            Controls.Add(buttonPrintScreen) : buttonPrintScreen.BringToFront()
            Controls.Add(buttonPrintAll) : buttonPrintAll.BringToFront()
            Controls.Add(buttonToggleComments) : buttonToggleComments.BringToFront()
            ' Controls for the DisplayArea.
            Controls.Add(panelSeparator)
            panelSeparator.BringToFront()
            Controls.Add(labelDisplayChunkIndex)
            labelDisplayChunkIndex.BringToFront()
            Controls.Add(labelDisplayDecStartAddress)
            labelDisplayDecStartAddress.BringToFront()
            Controls.Add(labelDisplayHexStartAddress)
            labelDisplayHexStartAddress.BringToFront()
            Controls.Add(labelDisplayFileContents)
            labelDisplayFileContents.BringToFront()
            Visible = True
            PerformLayout()
            BringToFront()
        End With
        Initialization()
    End Sub

    Private Sub Initialization()
        CreateDisplayTextboxes()
        ClearTheDisplayFromListboxDirectories()
        CurrentDirectory = ""
        CurrentFileName = ""
        textboxBytesPerChunk.Text = Str(DEFAULTNUMBYTESPERCHUNK)
        textboxBytesPerChunk.Enabled = False
        If (DEFAULTFORMAT = 1) Then
            checkboxDefaultHex.Checked = True
            checkboxDefaultASCII.Checked = False
        Else
            checkboxDefaultHex.Checked = False
            checkboxDefaultASCII.Checked = True
        End If
        checkboxDefaultHex.Enabled = False
        checkboxDefaultASCII.Enabled = False
        ShowComments = False
    End Sub

    '////////////////////////////////////////////////////////////////////////////////
    '////////////////////////////////////////////////////////////////////////////////
    '// Controls.

    Private gbWWide As Int32 = 160   ' Width of wide groupboxes
    Private gbWNarrow As Int32 = 110 ' Width of narrow groupboxes
    Private gbH As Int32 = 190       ' Height of groupboxes

    Private cntlWWide As Int32 = 160 - 10   ' Width of controls in wide groupboxes
    Private cntlWNarrow As Int32 = 110 - 10 ' Width of controls in narrow groupboxes
    Private cntlH As Int32 = 20             ' Height of labels, textboxes and buttons
```

```
Private listboxH As Int32 = 125         ' Height of listboxes

Private Row1 As Int32 = 10  ' Top row alignment of first row of controls
Private Row2 As Int32 = 35  ' Top row alignment of second row of controls
Private Row3 As Int32 = 60  ' Top row alignment of third row of controls
Private Row4 As Int32 = 85  ' Top row alignment of fourth row of controls
Private Row5 As Int32 = 110 ' Top row alignment of fifth row of controls
Private Row6 As Int32 = 135 ' Top row alignment of sixth row of controls
Private Row7 As Int32 = 165 ' Top row alignment of seventh row of controls
Private Col1 As Int32 = 5   ' Left column alignment for first column of controls

' Controls for listing directories.

Private groupboxDirectories As New Windows.Forms.GroupBox With _
    {.Size = New Drawing.Size(gbWWide, gbH), _
     .Location = New Drawing.Point(5, 5)}

Private labelDirectories As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWWide, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Select directory", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDirectories}

Private WithEvents listboxDirectories As New Windows.Forms.ListBox With _
    {.Size = New Drawing.Size(cntlWWide, listboxH), _
     .Location = New Drawing.Point(5, Row2), _
     .ScrollAlwaysVisible = True, .Parent = groupboxDirectories}

Private WithEvents buttonCDirectory As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(50, cntlH), _
     .Location = New Drawing.Point(5, Row2 + listboxH + 5), _
     .Text = "C:\", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDirectories}

Private WithEvents buttonDesktopDirectory As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(50, cntlH), _
     .Location = New Drawing.Point(5 + 50, Row2 + listboxH + 5), _
     .Text = "Dsktop", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDirectories}

Private WithEvents buttonUpDirectory As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(50, cntlH), _
     .Location = New Drawing.Point(5 + 50 + 50, Row2 + listboxH + 5), _
     .Text = "Parent", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDirectories}

' Controls for listing files in a directory.

Private groupboxFiles As New Windows.Forms.GroupBox With _
    {.Size = New Drawing.Size(gbWWide, gbH), _
     .Location = New Drawing.Point(5 + gbWWide + 5, 5)}

Private labelFiles As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWWide, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Select file", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxFiles}
```

```
Private WithEvents listboxFiles As New Windows.Forms.ListBox With _
    {.Size = New Drawing.Size(cntlWWide, listboxH), _
     .Location = New Drawing.Point(5, Row2), _
     .ScrollAlwaysVisible = True, .Parent = groupboxFiles}

' Controls for the selected file.

Private groupboxSelectedFile As New Windows.Forms.GroupBox With _
   {.Size = New Drawing.Size(gbWNarrow, gbH), _
    .Location = New Drawing.Point(5 + (2 * (gbWWide + 5)), 5)}

Private labelFileName As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Selected file", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}

Private textboxFileName As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row2), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Enabled = False, .Parent = groupboxSelectedFile}

Private labelBytesPerLine As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row3), _
     .Text = "Bytes per line", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}

Private WithEvents textboxBytesPerChunk As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row4), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxSelectedFile}

Private labelDefaultFormat As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row5), _
     .Text = "Default format", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}

Private WithEvents checkboxDefaultHex As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(45, cntlH), _
     .Location = New Drawing.Point(5, Row6), _
     .Text = "Hex", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}

Private WithEvents checkboxDefaultASCII As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(55, cntlH), _
     .Location = New Drawing.Point(50, Row6), _
     .Text = "ASCII", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}

Private WithEvents buttonOpenFile As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row7), _
     .Text = "Open file", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxSelectedFile}
```

```
' Controls for breaking a chunk.

Private groupboxBreakChunk As New Windows.Forms.GroupBox With _
   {.Size = New Drawing.Size(gbWNarrow, gbH), _
    .Location = New Drawing.Point(5 + (2 * (gbWWide + 5)) + gbWNarrow + 5, 5)}

Private labelBreakChunk As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Break at address", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxBreakChunk}

Private textboxBreakChunk As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row2), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxBreakChunk}

Private WithEvents buttonBreakChunk As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row3), _
     .Text = "OK", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxBreakChunk}

Private labelAddComment As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row5), _
     .Text = "Add comment", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxBreakChunk}

Private textboxAddComment As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row6), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxBreakChunk}

Private WithEvents buttonAddComment As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row7), _
     .Text = "OK", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxBreakChunk}

' Controls for changing the display format of a chunk.

Private groupboxChunkFormat As New Windows.Forms.GroupBox With _
   {.Size = New Drawing.Size(gbWNarrow, gbH), _
    .Location = New Drawing.Point( _
        5 + (2 * (gbWWide + 5)) + (2 * (gbWNarrow + 5)), 5)}

Private labelDisplayChunk As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Line format", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxChunkFormat}

Private textboxDisplayChunkNumber As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
```

```
     .Location = New Drawing.Point(5, Row2), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxChunkFormat}

Private TinyFont As New Drawing.Font("Times New Roman", 8)

Private WithEvents checkboxDisplayChunkHex As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(2, Row3), _
     .Text = "Hex", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents checkboxDisplayChunkASCII As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(2, Row4), _
     .Text = "ASCI", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents checkboxDisplayChunkBinary As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(2, Row5), _
     .Text = "Binry", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents checkboxDisplayChunkDec As New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(CInt(cntlWNarrow / 2) + 5, Row3), _
     .Text = "Dec", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents checkboxDisplayChunkLittleEndianDec As _
    New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(CInt(cntlWNarrow / 2) + 5, Row4), _
     .Text = "l.enD", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents checkboxDisplayChunkLittleEndianHex As _
    New Windows.Forms.CheckBox With _
    {.Size = New Drawing.Size(CInt(cntlWNarrow / 2) + 3, cntlH), _
     .Location = New Drawing.Point(CInt(cntlWNarrow / 2) + 5, Row5), _
     .Text = "l.enH", .TextAlign = ContentAlignment.MiddleLeft, _
     .Font = TinyFont, .Parent = groupboxChunkFormat}

Private WithEvents buttonDisplayChunk As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row6), _
     .Text = "OK", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxChunkFormat}

' Controls for combining two chunks.

Private groupboxCombineChunks As New Windows.Forms.GroupBox With _
   {.Size = New Drawing.Size(gbWNarrow, gbH), _
    .Location = New Drawing.Point( _
        5 + (2 * (gbWWide + 5)) + (3 * (gbWNarrow + 5)), 5)}

Private labelCombineChunks As New Windows.Forms.Label With _
```

```
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Combine lines", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxCombineChunks}

Private labelCombineChunksNumber1 As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(20, cntlH), _
     .Location = New Drawing.Point(5, Row2), _
     .Text = "#1", .TextAlign = ContentAlignment.MiddleLeft, _
     .Parent = groupboxCombineChunks}

Private textboxCombineChunksNumber1 As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow - 20, cntlH), _
     .Location = New Drawing.Point(5 + 20, Row2), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxCombineChunks}

Private labelCombineChunksNumber2 As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(20, cntlH), _
     .Location = New Drawing.Point(5, Row3), _
     .Text = "#2", .TextAlign = ContentAlignment.MiddleLeft, _
     .Parent = groupboxCombineChunks}

Private textboxCombineChunksNumber2 As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow - 20, cntlH), _
     .Location = New Drawing.Point(5 + 20, Row3), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Parent = groupboxCombineChunks}

Private WithEvents buttonCombineChunks As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row4), _
     .Text = "OK", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxCombineChunks}

' Controls for shifting the display range up or down.

Private groupboxDisplayStartAddress As New Windows.Forms.GroupBox With _
   {.Size = New Drawing.Size(gbWNarrow, gbH), _
    .Location = New Drawing.Point( _
        5 + (2 * (gbWWide + 5)) + (4 * (gbWNarrow + 5)), 5)}

Private labelDisplayStartAddress As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row1), _
     .Text = "Display address", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDisplayStartAddress}

Private WithEvents buttonStartAddressUp As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row2), _
     .Text = "Up", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDisplayStartAddress}

Private WithEvents textboxStartAddress As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row3), _
     .Text = "", .TextAlign = HorizontalAlignment.Center, _
```

```
     .Parent = groupboxDisplayStartAddress}

Private WithEvents buttonStartAddressDown As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, cntlH), _
     .Location = New Drawing.Point(5, Row4), _
     .Text = "Down", .TextAlign = ContentAlignment.MiddleCenter, _
     .Parent = groupboxDisplayStartAddress}

' Principal buttons.

Private WithEvents buttonExit As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, 30), _
     .Location = New Drawing.Point( _
         5 + (2 * (gbWWide + 5)) + (5 * (gbWNarrow + 5)), 10), _
     .Text = "Exit", .TextAlign = ContentAlignment.MiddleCenter}

Private WithEvents buttonPrintScreen As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, 30), _
     .Location = New Drawing.Point( _
         5 + (2 * (gbWWide + 5)) + (5 * (gbWNarrow + 5)), 45), _
     .Text = "Print screen", .TextAlign = ContentAlignment.MiddleCenter}

Private WithEvents buttonPrintAll As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, 30), _
     .Location = New Drawing.Point( _
         5 + (2 * (gbWWide + 5)) + (5 * (gbWNarrow + 5)), 80), _
     .Text = "Print all", .TextAlign = ContentAlignment.MiddleCenter}

Private WithEvents buttonToggleComments As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(cntlWNarrow, 30), _
     .Location = New Drawing.Point( _
         5 + (2 * (gbWWide + 5)) + (5 * (gbWNarrow + 5)), 115), _
     .Text = "Toggle comments", .TextAlign = ContentAlignment.MiddleCenter}

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// File content display.

Private CourierFont As New Drawing.Font("Courier New", 9)

Private DisplayAreaRow1 As Int32 = 200  ' Top row alignment for DisplayArea
Private DisplayAreaRow2 As Int32 = 205  ' Top row alignment of headers
Private DisplayAreaRow3 As Int32 = 230  ' Top row alignment for file contents

Private IndexW As Int32 = 70         ' Width of column with chunk indices
Private IndexCol As Int32 = 5        ' Left column alignment of chunk indices
Private DecAddressW As Int32 = 70    ' Width of decimal starting addresses
Private DecAddressCol As Int32 = 80  ' Left column alignment of decimal addresses
Private HexAddressW As Int32 = 80    ' Width of hex starting addresses
Private HexAddressCol As Int32 = 155 ' Left column alignment of hex addresses
Private ContentsW As Int32 = 770     ' Width of column with contents of the file
Private ContentsCol As Int32 = 240   ' Left column alignment of file contents
Private CommentW As Int32 = 200      ' Width of textboxes with line comments
Private CommentCol As Int32 = 640    ' Left column alignment for line comments
Private DisplayAreaH As Int32 = 470  ' Height of the DisplayArea

Private panelSeparator As New Windows.Forms.Panel With _
    {.Size = New Drawing.Size(SCREENWPIXEL - 15, 1), _
```

```
     .Location = New Drawing.Point(5, DisplayAreaRow1), _
     .BackColor = Color.Black}

Private labelDisplayChunkIndex As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(IndexW, cntlH), _
     .Location = New Drawing.Point(IndexCol, DisplayAreaRow2), _
     .Text = "Chunk index", .TextAlign = ContentAlignment.MiddleCenter}

Private labelDisplayDecStartAddress As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(DecAddressW, cntlH), _
     .Location = New Drawing.Point(DecAddressCol, DisplayAreaRow2), _
     .Text = "Dec Address", .TextAlign = ContentAlignment.MiddleLeft}

Private labelDisplayHexStartAddress As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(HexAddressW, cntlH), _
     .Location = New Drawing.Point(HexAddressCol, DisplayAreaRow2), _
     .Text = "Hex Address", .TextAlign = ContentAlignment.MiddleLeft}

Private labelDisplayFileContents As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(ContentsW, cntlH), _
     .Location = New Drawing.Point(ContentsCol, DisplayAreaRow2), _
     .Text = "Contents of the selected file", _
     .TextAlign = ContentAlignment.MiddleLeft}

Private tbDisplayChunkIndex(MAXNUMLINESINDISPLAY) As Windows.Forms.TextBox
Private tbDisplayAddressDec(MAXNUMLINESINDISPLAY) As Windows.Forms.TextBox
Private tbDisplayAddressHex(MAXNUMLINESINDISPLAY) As Windows.Forms.TextBox
Private tbDisplayChunkContents(MAXNUMLINESINDISPLAY) As Windows.Forms.TextBox
Private tbDisplayChunkComment(MAXNUMLINESINDISPLAY) As Windows.Forms.TextBox

Private Sub CreateDisplayTextboxes()
    With Me
        For I As Int32 = 1 To MAXNUMLINESINDISPLAY Step 1
            tbDisplayChunkIndex(I) = New Windows.Forms.TextBox
            With tbDisplayChunkIndex(I)
                .Size = New Drawing.Size(IndexW, 20)
                .Location = New Drawing.Point(IndexCol, _
                    DisplayAreaRow3 + ((I - 1) * 20))
                .Text = ""
                .TextAlign = HorizontalAlignment.Center
                .Font = CourierFont
            End With
            tbDisplayAddressDec(I) = New Windows.Forms.TextBox
            With tbDisplayAddressDec(I)
                .Size = New Drawing.Size(DecAddressW, 20)
                .Location = New Drawing.Point(DecAddressCol, _
                    DisplayAreaRow3 + ((I - 1) * 20))
                .Text = ""
                .TextAlign = HorizontalAlignment.Right
                .Font = CourierFont
            End With
            tbDisplayAddressHex(I) = New Windows.Forms.TextBox
            With tbDisplayAddressHex(I)
                .Size = New Drawing.Size(HexAddressW, 20)
                .Location = New Drawing.Point(HexAddressCol, _
                    DisplayAreaRow3 + ((I - 1) * 20))
                .Text = ""
                .TextAlign = HorizontalAlignment.Right
```

```
                .Font = CourierFont
            End With
            tbDisplayChunkContents(I) = New Windows.Forms.TextBox
            With tbDisplayChunkContents(I)
                .Size = New Drawing.Size(ContentsW, 20)
                .Location = New Drawing.Point(ContentsCol, _
                    DisplayAreaRow3 + ((I - 1) * 20))
                .Text = ""
                .TextAlign = HorizontalAlignment.Left
                .Font = CourierFont
            End With
            tbDisplayChunkComment(I) = New Windows.Forms.TextBox
            With tbDisplayChunkComment(I)
                .Size = New Drawing.Size(CommentW, 20)
                .Location = New Drawing.Point(CommentCol, _
                    DisplayAreaRow3 + ((I - 1) * 20))
                .Visible = False
                .Text = ""
                .TextAlign = HorizontalAlignment.Left
                .Font = CourierFont
                .BorderStyle = BorderStyle.FixedSingle
                .BackColor = Color.AntiqueWhite
                AddHandler tbDisplayChunkComment(I).KeyUp, _
                    AddressOf tbDisplayChunkComment_Key
            End With
            Controls.Add(tbDisplayChunkIndex(I))
            Controls.Add(tbDisplayAddressDec(I))
            Controls.Add(tbDisplayAddressHex(I))
            Controls.Add(tbDisplayChunkContents(I))
            Controls.Add(tbDisplayChunkComment(I))
        Next I
    End With
End Sub

Private Sub tbDisplayChunkComment_Key( _
    ByVal sender As Object, ByVal e As System.EventArgs)
    Dim SenderLine As Int32
    Dim SelectedChunk As Int32
    ' SenderLine is the line on the screen whose text has changed.
    For I As Int32 = 1 To MAXNUMLINESINDISPLAY Step 1
        If (sender.Equals(tbDisplayChunkComment(I))) Then
            SenderLine = I
        End If
    Next I
    ' SelectedChunk is the index of the chunk whose text has changed.
    SelectedChunk = CInt(Val(tbDisplayChunkIndex(SenderLine).Text))
    ' Save the changed text.
    Chunk(SelectedChunk).Comment = tbDisplayChunkComment(SenderLine).Text
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxDirectories.

Private Sub buttonCDirectory_Click() Handles buttonCDirectory.MouseClick
    ClearTheDisplayFromListboxDirectories()
    CurrentDirectory = CDIRECTORY
    PopulateListboxDirectories(CurrentDirectory)
```

```
    PopulateListboxFiles(CurrentDirectory)
End Sub

Private Sub buttonDesktopDirectory_Click() Handles buttonDesktopDirectory.MouseClick
    ClearTheDisplayFromListboxDirectories()
    CurrentDirectory = DESKTOPDIRECTORY
    PopulateListboxDirectories(DESKTOPDIRECTORY)
    PopulateListboxFiles(DESKTOPDIRECTORY)
End Sub

Private Sub buttonUpDirectory_Click() Handles buttonUpDirectory.MouseClick
    ClearTheDisplayFromListboxDirectories()
    If (CurrentDirectory = CDIRECTORY) Then
        Exit Sub
    End If
    If (CurrentDirectory = "") Then
        Exit Sub
    End If
    CurrentDirectory = Strings.Left(CurrentDirectory, Len(CurrentDirectory) - 1)
    Do While (Strings.Right(CurrentDirectory, 1) <> "\")
        CurrentDirectory = Strings.Left(CurrentDirectory, Len(CurrentDirectory) - 1)
    Loop
    PopulateListboxDirectories(CurrentDirectory)
    PopulateListboxFiles(CurrentDirectory)
End Sub

Private Sub listboxDirectories_Click() Handles listboxDirectories.MouseClick
    If (listboxDirectories.SelectedIndex < 0) Then
        Exit Sub
    End If
    CurrentDirectory = CurrentDirectory & "\" & _
        CType(listboxDirectories.SelectedItem, String)
    ClearTheDisplayFromListboxDirectories()
    PopulateListboxDirectories(CurrentDirectory)
    PopulateListboxFiles(CurrentDirectory)
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxFiles.

Private Sub listboxFiles_Click() Handles listboxFiles.MouseClick
    If (listboxFiles.SelectedIndex < 0) Then
        Exit Sub
    End If
    CurrentFileName = CType(listboxFiles.SelectedItem, String)
    ClearTheDisplayFromSelectedFile()
    textboxFileName.Text = CurrentFileName
    textboxBytesPerChunk.Enabled = True
    checkboxDefaultHex.Enabled = True
    checkboxDefaultASCII.Enabled = True
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxSelectedFile.

Private Sub textboxBytesPerChunk_Click(ByVal sender As Object, _
```

```
    ByVal e As KeyEventArgs) Handles textboxBytesPerChunk.KeyUp
    Dim tempBytesPerChunk As Int32
    tempBytesPerChunk = CInt(Val(textboxBytesPerChunk.Text))
    If (tempBytesPerChunk < 1) Then
        tempBytesPerChunk = 1
    End If
    If (tempBytesPerChunk > MAXNUMBYTESPERCHUNK) Then
        tempBytesPerChunk = MAXNUMBYTESPERCHUNK
    End If
    DEFAULTNUMBYTESPERCHUNK = tempBytesPerChunk
    textboxBytesPerChunk.Text = Trim(Str(tempBytesPerChunk))
End Sub

Private Sub checkboxDefaultHex_Click() Handles checkboxDefaultHex.CheckedChanged
    If (checkboxDefaultHex.Checked = True) Then
        DEFAULTFORMAT = 1
        checkboxDefaultASCII.Checked = False
    End If
End Sub

Private Sub checkboxDefaultASCII_Click() Handles checkboxDefaultASCII.CheckedChanged
    If (checkboxDefaultASCII.Checked = True) Then
        DEFAULTFORMAT = 2
        checkboxDefaultHex.Checked = False
    End If
End Sub

Private Sub buttonOpenFile_Click() Handles buttonOpenFile.MouseClick
    Dim tempFileName As String
    ClearTheDisplayFromBreakChunk()
    textboxBytesPerChunk.Enabled = False
    checkboxDefaultHex.Enabled = False
    checkboxDefaultASCII.Enabled = False
    If (CurrentDirectory = "") Then
        Exit Sub
    End If
    If (textboxFileName.Text = "") Then
        Exit Sub
    End If
    CurrentFileName = textboxFileName.Text
    tempFileName = CurrentDirectory & "\" & CurrentFileName
    ' Try to open the selected file.
    Try
        MainInputStream = New System.IO.FileStream(tempFileName, IO.FileMode.Open)
    Catch ex As Exception
        MsgBox("Could not open file " & tempFileName & ex.ToString)
        Exit Sub
    End Try
    ' Inactivate all of the Chunk() structures.
    For I As Int32 = 1 To MAXNUMCHUNKS Step 1
        Chunk(I).Active = False
    Next I
    ' Read the file and store in Chunk() structures.
    NumberOfChunks = 0
    NumberOfBytesInFile = 0
    Dim CurrentAddress As Int32 = 0
    Do
        If (NumberOfChunks = MAXNUMCHUNKS) Then
```

```
            ' Do not continue if MAXNUMCHUNKS have been filled.
            Exit Do
        End If
        If (MainInputStream.Position >= MainInputStream.Length) Then
            ' Do not continue if we are past the end of the file.
            Exit Do
        End If
        ' Increment the index and instantiate a new chunk.
        NumberOfChunks = NumberOfChunks + 1
        InstantiateChunkStructure(NumberOfChunks)
        ' Set some of the elements of the structure of the new chunk.
        Chunk(NumberOfChunks).StartAddress = CurrentAddress
        Chunk(NumberOfChunks).NumBytes = 0
        Chunk(NumberOfChunks).Active = True
        Chunk(NumberOfChunks).DisplayFormat = DEFAULTFORMAT
        Chunk(NumberOfChunks).Comment = ""
        ' Read byte-by-byte from the file.
        For I As Int32 = 0 To (DEFAULTNUMBYTESPERCHUNK - 1) Step 1
            If (MainInputStream.Position <= MainInputStream.Length) Then
                ' Do not read if we are past the end of the file.
                Chunk(NumberOfChunks).ByteStream(I) = MainInputStream.ReadByte
                Chunk(NumberOfChunks).NumBytes = Chunk(NumberOfChunks).NumBytes + 1
                CurrentAddress = CurrentAddress + 1
                NumberOfBytesInFile = NumberOfBytesInFile + 1
            Else
                Chunk(NumberOfChunks).ByteStream(I) = 0
            End If
        Next I
        Chunk(NumberOfChunks).EndAddress = CurrentAddress - 1
        If (NumberOfBytesInFile > MAXNUMBYTESINFILE) Then
            ' Do not proceed if we have read more than MAXNUMBYTESINFILE bytes.
            Exit Do
        End If
        ' Display progress to the user through the StartAddress textbox.
        Application.DoEvents()
        textboxStartAddress.Text = Str(CurrentAddress)
    Loop
    ' Close the file.
    MainInputStream.Close()
    MainInputStream.Dispose()
    ' Zero the starting address for the display.
    textboxStartAddress.Text = Trim(Str(0))
    ' Render the display.
    RenderTheDisplay()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxBreakChunk.

Private Sub buttonBreakChunk_Click() Handles buttonBreakChunk.MouseClick
    Dim StartOfNewChunk As Int32 = CInt(Val(textboxBreakChunk.Text))
    Dim IndexOfSourceChunk As Int32
    Dim AbsoluteAddressOfFirstByteToTransfer As Int32
    Dim AbsoluteAddressOfLastByteToTransfer As Int32
    Dim NumberOfBytesToTransfer As Int32
    Dim SourceByteIndex As Int32
    Dim DestinationByteIndex As Int32
```

```
    ' Update the display textbox to show the user what was received.
    textboxBreakChunk.Text = Trim(Str(StartOfNewChunk))
    ' Check validity of the address.
    If ((StartOfNewChunk < 0) Or (StartOfNewChunk > (NumberOfBytesInFile - 1))) Then
        Exit Sub
    End If
    IndexOfSourceChunk = FindChunkContainingAddress(StartOfNewChunk)
    ' Deal with the inability to find a straddling chunk.
    If (IndexOfSourceChunk = -1) Then
        MsgBox("Error: Could not find the chunk containing address " & _
            Trim(Str(StartOfNewChunk)) & ".")
        Exit Sub
    End If
    ' If the starting address is already at the start of a chunk, do nothing.
    If (Chunk(IndexOfSourceChunk).StartAddress = StartOfNewChunk) Then
        Exit Sub
    End If
    ' Initialize a new chunk structure for the remainder of this chunk.
    NumberOfChunks = NumberOfChunks + 1
    Chunk(NumberOfChunks).ByteStream = New Int32(MAXNUMBYTESPERCHUNK) {}
    ' Copy from the source chunk into the new chunk.
    AbsoluteAddressOfFirstByteToTransfer = StartOfNewChunk
    AbsoluteAddressOfLastByteToTransfer = Chunk(IndexOfSourceChunk).EndAddress
    NumberOfBytesToTransfer = AbsoluteAddressOfLastByteToTransfer - _
        AbsoluteAddressOfFirstByteToTransfer
    SourceByteIndex = StartOfNewChunk - (Chunk(IndexOfSourceChunk).StartAddress + 1)
    DestinationByteIndex = -1
    Do
        SourceByteIndex = SourceByteIndex + 1
        If (SourceByteIndex > Chunk(IndexOfSourceChunk).EndAddress) Then
            Exit Do
        End If
        DestinationByteIndex = DestinationByteIndex + 1
        Chunk(NumberOfChunks).ByteStream(DestinationByteIndex) = _
            Chunk(IndexOfSourceChunk).ByteStream(SourceByteIndex)
    Loop
    ' Update the parameters in both chunk's structures.
    Chunk(IndexOfSourceChunk).EndAddress = StartOfNewChunk - 1
    Chunk(IndexOfSourceChunk).NumBytes = Chunk(IndexOfSourceChunk).EndAddress - _
        Chunk(IndexOfSourceChunk).StartAddress + 1
    Chunk(NumberOfChunks).StartAddress = StartOfNewChunk
    Chunk(NumberOfChunks).NumBytes = NumberOfBytesToTransfer + 1
    Chunk(NumberOfChunks).EndAddress = StartOfNewChunk + NumberOfBytesToTransfer
    Chunk(NumberOfChunks).Active = True
    Chunk(NumberOfChunks).DisplayFormat = 1
    ' Refresh the display.
    RenderTheDisplay()
End Sub

Private Sub buttonAddComment_Click() Handles buttonAddComment.MouseClick
    Dim IndexOfThisChunk As Int32 = CInt(Val(textboxAddComment.Text))
    ' Update the display textbox to show the user what was received.
    textboxAddComment.Text = Trim(Str(IndexOfThisChunk))
    ' Check validity of the index.
    If ((IndexOfThisChunk < 0) Or (IndexOfThisChunk > NumberOfChunks)) Then
        Exit Sub
    End If
    ' Check that this chunk is still active.
```

```
    If (Chunk(IndexOfThisChunk).Active = False) Then
        Exit Sub
    End If
    ' Check that this chunk is currently being displayed.
    Do
        For I As Int32 = 1 To MAXNUMLINESINDISPLAY Step 1
            If (Val(tbDisplayChunkIndex(I).Text) = IndexOfThisChunk) Then
                Exit Do
            End If
        Next I
        Exit Sub
    Loop
    ' Turn on the ShowComments flag.
    ShowComments = True
    ' Display a blank textbox.
    Chunk(IndexOfThisChunk).Comment = " "
    ' Render the display.
    RenderTheDisplay()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxFormatChunk.

Private Sub checkboxDisplayChunkHex_Click() Handles _
    checkboxDisplayChunkHex.CheckedChanged
    If (checkboxDisplayChunkHex.Checked = True) Then
        checkboxDisplayChunkASCII.Checked = False
        checkboxDisplayChunkBinary.Checked = False
        checkboxDisplayChunkDec.Checked = False
        checkboxDisplayChunkLittleEndianDec.Checked = False
        checkboxDisplayChunkLittleEndianHex.Checked = False
    End If
End Sub

Private Sub checkboxDisplayChunkASCII_Click() Handles _
    checkboxDisplayChunkASCII.CheckedChanged
    If (checkboxDisplayChunkASCII.Checked = True) Then
        checkboxDisplayChunkHex.Checked = False
        checkboxDisplayChunkBinary.Checked = False
        checkboxDisplayChunkDec.Checked = False
        checkboxDisplayChunkLittleEndianDec.Checked = False
        checkboxDisplayChunkLittleEndianHex.Checked = False
    End If
End Sub

Private Sub checkboxDisplayChunkBinary_Click() Handles _
    checkboxDisplayChunkBinary.CheckedChanged
    If (checkboxDisplayChunkBinary.Checked = True) Then
        checkboxDisplayChunkHex.Checked = False
        checkboxDisplayChunkASCII.Checked = False
        checkboxDisplayChunkDec.Checked = False
        checkboxDisplayChunkLittleEndianDec.Checked = False
        checkboxDisplayChunkLittleEndianHex.Checked = False
    End If
End Sub

Private Sub checkboxDisplayChunkDec_Click() Handles _
```

```
    checkboxDisplayChunkDec.CheckedChanged
    If (checkboxDisplayChunkDec.Checked = True) Then
        checkboxDisplayChunkHex.Checked = False
        checkboxDisplayChunkASCII.Checked = False
        checkboxDisplayChunkBinary.Checked = False
        checkboxDisplayChunkLittleEndianDec.Checked = False
        checkboxDisplayChunkLittleEndianHex.Checked = False
    End If
End Sub

Private Sub checkboxDisplayChunkLittleEndianDec_Click() Handles _
    checkboxDisplayChunkLittleEndianDec.CheckedChanged
    If (checkboxDisplayChunkLittleEndianDec.Checked = True) Then
        checkboxDisplayChunkHex.Checked = False
        checkboxDisplayChunkASCII.Checked = False
        checkboxDisplayChunkBinary.Checked = False
        checkboxDisplayChunkDec.Checked = False
        checkboxDisplayChunkLittleEndianHex.Checked = False
    End If
End Sub

Private Sub checkboxDisplayChunkLittleEndianHex_Click() Handles _
    checkboxDisplayChunkLittleEndianHex.CheckedChanged
    If (checkboxDisplayChunkLittleEndianHex.Checked = True) Then
        checkboxDisplayChunkHex.Checked = False
        checkboxDisplayChunkASCII.Checked = False
        checkboxDisplayChunkBinary.Checked = False
        checkboxDisplayChunkDec.Checked = False
        checkboxDisplayChunkLittleEndianDec.Checked = False
    End If
End Sub

Private Sub buttonDisplayChunkOK_Click() Handles buttonDisplayChunk.MouseClick
    Dim SelectedChunk As Int32
    ' Do not proceed if no format has been selected.
    If ((checkboxDisplayChunkHex.Checked = False) And _
        (checkboxDisplayChunkASCII.Checked = False) And _
        (checkboxDisplayChunkBinary.Checked = False) And _
        (checkboxDisplayChunkDec.Checked = False) And _
        (checkboxDisplayChunkLittleEndianDec.Checked = False) And _
        (checkboxDisplayChunkLittleEndianHex.Checked = False)) Then
        Exit Sub
    End If
    ' Do not proceed if no chunk number has been specified.
    If (textboxDisplayChunkNumber.Text = "") Then
        Exit Sub
    Else
        textboxDisplayChunkNumber.Text = _
            Trim(Str(CInt(Val(textboxDisplayChunkNumber.Text))))
        Me.Refresh()
    End If
    ' Do not proceed if the chunk number is invalid.
    SelectedChunk = CInt(Val(textboxDisplayChunkNumber.Text))
    If ((SelectedChunk < 0) Or (SelectedChunk > NumberOfChunks)) Then
        Exit Sub
    End If
    ' Do not proceed if the chunk has been de-activated.
    If (Chunk(SelectedChunk).Active = False) Then
```

```
            Exit Sub
        End If
        ' Update the structure of the chunk.
        If (checkboxDisplayChunkHex.Checked = True) Then
            Chunk(SelectedChunk).DisplayFormat = 1
        Else
            If (checkboxDisplayChunkASCII.Checked = True) Then
                Chunk(SelectedChunk).DisplayFormat = 2
            Else
                If (checkboxDisplayChunkBinary.Checked = True) Then
                    Chunk(SelectedChunk).DisplayFormat = 3
                Else
                    If (checkboxDisplayChunkDec.Checked = True) Then
                        Chunk(SelectedChunk).DisplayFormat = 4
                    Else
                        If (checkboxDisplayChunkLittleEndianDec.Checked = True) Then
                            Chunk(SelectedChunk).DisplayFormat = 5
                        Else
                            If (checkboxDisplayChunkLittleEndianHex.Checked = True) Then
                                Chunk(SelectedChunk).DisplayFormat = 6
                            Else
                                Chunk(SelectedChunk).DisplayFormat = 0
                            End If
                        End If
                    End If
                End If
            End If
        End If
        ' Update the display.
        RenderTheDisplay()
    End Sub

    '////////////////////////////////////////////////////////////////////////////////
    '////////////////////////////////////////////////////////////////////////////////
    '// Handlers for groupboxCombineChunks.

    Private Sub buttonCombineChunks_Click() Handles buttonCombineChunks.MouseClick
        Dim IndexOfChunk1 As Int32 = CInt(Val(textboxCombineChunksNumber1.Text))
        Dim IndexOfChunk2 As Int32 = CInt(Val(textboxCombineChunksNumber2.Text))
        Dim IndexOfByteToTransfer As Int32
        ' Update the display textbox to show the user what was received.
        textboxCombineChunksNumber1.Text = Trim(Str(IndexOfChunk1))
        textboxCombineChunksNumber2.Text = Trim(Str(IndexOfChunk2))
        ' Check validity of the chunks.
        If ((IndexOfChunk1 < 0) Or (IndexOfChunk1 > NumberOfChunks)) Then
            Exit Sub
        End If
        If ((IndexOfChunk2 < 0) Or (IndexOfChunk2 > NumberOfChunks)) Then
            Exit Sub
        End If
        If (IndexOfChunk1 = IndexOfChunk2) Then
            Exit Sub
        End If
        ' Check the adjacency of the two chunks.
        If ((Chunk(IndexOfChunk1).EndAddress + 1) <> _
            Chunk(IndexOfChunk2).StartAddress) Then
            Exit Sub
        End If
```

```
    ' Copy from the second chunk into the first chunk.
    For I As Int32 = 1 To Chunk(IndexOfChunk2).NumBytes Step 1
        IndexOfByteToTransfer = Chunk(IndexOfChunk1).NumBytes + I
        Chunk(IndexOfChunk1).ByteStream(IndexOfByteToTransfer - 1) = _
            Chunk(IndexOfChunk2).ByteStream(I - 1)
    Next I
    ' Update the parameters in both chunk's structures.
    Chunk(IndexOfChunk1).EndAddress = Chunk(IndexOfChunk2).EndAddress
    Chunk(IndexOfChunk1).NumBytes = Chunk(IndexOfChunk1).EndAddress - _
        Chunk(IndexOfChunk1).StartAddress + 1
    Chunk(IndexOfChunk2).Active = False
    ' Refresh the display.
    RenderTheDisplay()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for groupboxDisplayStartAddress.

Private Sub buttonStartAddressUp_Click() Handles buttonStartAddressUp.MouseClick
    Dim FirstAddressInDisplay As Int32
    Dim IndexOfBoundingChunk As Int32
    ' Determine the index of the chunk which starts at FirstAddressinDisplay.
    FirstAddressInDisplay = CInt(Val(textboxStartAddress.Text))
    IndexOfBoundingChunk = FindChunkContainingAddress(FirstAddressInDisplay)
    If ((IndexOfBoundingChunk < 0) Or (IndexOfBoundingChunk > MAXNUMCHUNKS)) Then
        MsgBox("Error: Could not find the chunk bounding address " & _
            Trim(Str(FirstAddressInDisplay)) & ".")
        Exit Sub
    End If
    ' Determine the index of the preceding chunk.
    If (Chunk(IndexOfBoundingChunk).StartAddress > 0) Then
        IndexOfBoundingChunk = FindChunkContainingAddress( _
            Chunk(IndexOfBoundingChunk).StartAddress - 1)
        FirstAddressInDisplay = Chunk(IndexOfBoundingChunk).StartAddress
    End If
    textboxStartAddress.Text = Trim(Str(FirstAddressInDisplay))
    ' Render the display.
    ClearTheDisplayFromChunkIndices()
    RenderTheDisplay()
End Sub

Private Sub buttonStartAddressDown_Click() Handles buttonStartAddressDown.MouseClick
    Dim FirstAddressInDisplay As Int32
    Dim IndexOfBoundingChunk As Int32
    ' Determine the index of the chunk which starts at FirstAddressinDisplay.
    FirstAddressInDisplay = CInt(Val(textboxStartAddress.Text))
    IndexOfBoundingChunk = FindChunkContainingAddress(FirstAddressInDisplay)
    If ((IndexOfBoundingChunk < 0) Or (IndexOfBoundingChunk > MAXNUMCHUNKS)) Then
        MsgBox("Error: Could not find the chunk bounding address " & _
            Trim(Str(FirstAddressInDisplay)) & ".")
        Exit Sub
    End If
    ' Increase to the starting address of the following chunk.
    If (Chunk(IndexOfBoundingChunk).EndAddress < (NumberOfBytesInFile - 1)) Then
        FirstAddressInDisplay = Chunk(IndexOfBoundingChunk).EndAddress + 1
    End If
    textboxStartAddress.Text = Trim(Str(FirstAddressInDisplay))
```

```
    ' Render the display.
    ClearTheDisplayFromChunkIndices()
    RenderTheDisplay()
End Sub

Private Sub textboxStartAddress_Click(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles textboxStartAddress.KeyUp
    Dim EntryIsHex As Boolean = False
    Dim tempStartAddress As Int32
    ' Only process the enter key.
    If ((e.KeyCode <> Keys.Enter) And (e.KeyCode <> Keys.Tab)) Then
        Exit Sub
    End If
    ' Determine if the user's entry is a hex address.
    If ((Strings.InStr(textboxStartAddress.Text, "H") <> 0) Or _
        (Strings.InStr(textboxStartAddress.Text, "h") <> 0) Or _
        (Strings.InStr(textboxStartAddress.Text, "0x") <> 0)) Then
        EntryIsHex = True
    End If
    ' Determine the starting address.
    If (EntryIsHex = True) Then
        tempStartAddress = ConvertHexStringToInt(textboxStartAddress.Text)
    Else
        tempStartAddress = CInt(Val(Trim(textboxStartAddress.Text)))
    End If
    ' Validate the starting address.
    If (tempStartAddress < 0) Then
        tempStartAddress = 0
    End If
    If (tempStartAddress > NumberOfBytesInFile) Then
        tempStartAddress = NumberOfBytesInFile - 8
    End If
    ' Show what was understood.
    textboxStartAddress.Text = Trim(Str(tempStartAddress))
    ' Render the display.
    ClearTheDisplayFromChunkIndices()
    RenderTheDisplay()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Handlers for prinicpal buttons.

Private Sub buttonExit_Click() Handles buttonExit.MouseClick
    Application.Exit()
End Sub

Private WithEvents printDocument1 As New System.Drawing.Printing.PrintDocument
Private memoryImage As Bitmap

Private Sub buttonPrintScreen_Click() Handles buttonPrintScreen.MouseClick
    Dim myGraphics As Graphics = Me.CreateGraphics()
    Dim s As Size = Me.Size
    memoryImage = New Bitmap(s.Width, s.Height, myGraphics)
    Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
    memoryGraphics.CopyFromScreen(Me.Location.X, Me.Location.Y, 0, 0, s)
    printDocument1.Print()
End Sub
```

```
Private Sub printDocument1_PrintPage(ByVal sender As System.Object, _
    ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
    Handles printDocument1.PrintPage
    e.Graphics.DrawImage(memoryImage, 0, 0, _
        CInt(8 * 96), CInt(SCREENHPIXEL * 8 * 96 / SCREENWPIXEL))
End Sub

Private Sub buttonToggleComments_Click() Handles buttonToggleComments.MouseClick
    If (ShowComments = True) Then
        ShowComments = False
        RenderTheDisplay()
    Else
        ShowComments = True
        RenderTheDisplay()
    End If
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Application-specific subroutines.
'// 1. PopulateListboxDirectories(String)
'// 2. PopulateListboxFiles(String)
'// 3. InstantiateChunkStructure(Int32)
'// 4. FindChunkFollowingAddress(Int32) As Int32
'// 5. FindChunkContainingAddress(Int32) as Int32
'// 6. FindChunkPrecedingAddress(Int32) as Int32

Private Sub PopulateListboxDirectories(ByVal dirname As String)
    listboxDirectories.Items.Clear()
    Dim dirinfo As New System.IO.DirectoryInfo(dirname)
    Dim di As System.IO.DirectoryInfo()
    di = dirinfo.GetDirectories()
    Dim onedi As System.IO.DirectoryInfo
    For Each onedi In di
        listboxDirectories.Items.Add(onedi.Name)
    Next onedi
End Sub

Private Sub PopulateListboxFiles(ByVal dirname As String)
    listboxFiles.Items.Clear()
    Dim dirinfo As New System.IO.DirectoryInfo(dirname)
    Dim fi As System.IO.FileInfo()
    fi = dirinfo.GetFiles()
    Dim onefi As System.IO.FileInfo
    For Each onefi In fi
        listboxFiles.Items.Add(onefi.Name)
    Next onefi
End Sub

Private Sub InstantiateChunkStructure(ByVal chunknumber As Int32)
    Chunk(chunknumber).ByteStream = New Int32(MAXNUMBYTESPERCHUNK) {}
End Sub

Private Function FindChunkFollowingAddress(ByVal Address As Int32) As Int32
    ' This function returns the index of the chunk which follows the chunk
    ' containing address Address.  The function returns -1 if Address is
    ' greater than the number of bytes in the file.  The function returns
```

```
    ' -2 if no chunk is found.  The function returns -3 if Address is in
    ' the final chunk, so there is no following chunk.
    Dim tempChunkIndex1 As Int32
    Dim tempChunkIndex2 As Int32
    Dim tempStartingAddress As Int32
    If (Address > (NumberOfBytesInFile - 1)) Then
        FindChunkFollowingAddress = -1
        Exit Function
    End If
    tempChunkIndex1 = FindChunkContainingAddress(Address)
    If (tempChunkIndex1 = -2) Then
        FindChunkFollowingAddress = -2
        Exit Function
    End If
    tempStartingAddress = Chunk(tempChunkIndex1).EndAddress + 1
    If (tempStartingAddress > (NumberOfBytesInFile - 1)) Then
        FindChunkFollowingAddress = -3
        Exit Function
    End If
    tempChunkIndex2 = FindChunkContainingAddress(tempStartingAddress)
    If (tempChunkIndex2 = -2) Then
        FindChunkFollowingAddress = -2
    End If
    FindChunkFollowingAddress = tempChunkIndex2
End Function

Private Function FindChunkContainingAddress(ByVal Address As Int32) As Int32
    ' This function returns the index of the chunk which contains the address
    ' Address.  The function returns -1 if Address is greater than the number
    ' of bytes in the file.  The functions returns -2 if no chunk is found.
    If (Address > (NumberOfBytesInFile - 1)) Then
        FindChunkContainingAddress = -1
        Exit Function
    End If
    For I As Int32 = 1 To NumberOfChunks Step 1
        If (Chunk(I).Active = True) Then
            If ((Chunk(I).StartAddress <= Address) And _
                (Chunk(I).EndAddress >= Address)) Then
                FindChunkContainingAddress = I
                Exit Function
            End If
        End If
    Next I
    FindChunkContainingAddress = -2
End Function

Private Function FindChunkPrecedingAddress(ByVal Address As Int32) As Int32
    ' This function returns the index of the chunk which precedes the chunk
    ' containing address Address.  The function returns -1 if Address is
    ' greater than the number of bytes in the file.  The function returns
    ' -2 if no chunk is found.  The function returns -3 if Address is in
    ' the first chunk, so there is no preceding chunk.
    Dim tempChunkIndex1 As Int32
    Dim tempChunkIndex2 As Int32
    Dim tempEndingAddress As Int32
    If (Address > (NumberOfBytesInFile - 1)) Then
        FindChunkPrecedingAddress = -1
        Exit Function
```

```
    End If
    tempChunkIndex1 = FindChunkContainingAddress(Address)
    If (tempChunkIndex1 = -2) Then
        FindChunkPrecedingAddress = -2
        Exit Function
    End If
    tempEndingAddress = Chunk(tempChunkIndex1).StartAddress - 1
    If (tempEndingAddress < 0) Then
        FindChunkPrecedingAddress = -3
        Exit Function
    End If
    tempChunkIndex2 = FindChunkContainingAddress(tempEndingAddress)
    If (tempChunkIndex2 = -2) Then
        FindChunkPrecedingAddress = -2
    End If
    FindChunkPrecedingAddress = tempChunkIndex2
End Function

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// Subroutines for display purposes.
'// 1. ClearTheDisplayFromListboxDirectories()
'// 2. ClearTheDisplayFromSelectedFile()
'// 3. ClearTheDisplayFromBreakChunk()
'// 4. ClearTheDisplayFromDisplayStartAddress()
'// 5. ClearTheDisplayFromChunkIndices()
'// 6. RenderTheDisplay()
'// 7. FormatContentsOfOneChunk(Int32) As String

Private Sub ClearTheDisplayFromListboxDirectories()
    listboxDirectories.Items.Clear()
    listboxFiles.Items.Clear()
    ClearTheDisplayFromSelectedFile()
End Sub

Private Sub ClearTheDisplayFromSelectedFile()
    textboxFileName.Text = ""
    ClearTheDisplayFromBreakChunk()
End Sub

Private Sub ClearTheDisplayFromBreakChunk()
    textboxBreakChunk.Text = ""
    textboxAddComment.Text = ""
    textboxDisplayChunkNumber.Text = ""
    checkboxDisplayChunkHex.Checked = False
    checkboxDisplayChunkASCII.Checked = False
    checkboxDisplayChunkBinary.Checked = False
    checkboxDisplayChunkDec.Checked = False
    checkboxDisplayChunkLittleEndianDec.Checked = False
    checkboxDisplayChunkLittleEndianHex.Checked = False
    textboxCombineChunksNumber1.Text = ""
    textboxCombineChunksNumber2.Text = ""
    ClearTheDisplayFromDisplayStartAddress()
End Sub

Private Sub ClearTheDisplayFromDisplayStartAddress()
    textboxStartAddress.Text = ""
    ClearTheDisplayFromChunkIndices()
```

```
End Sub

Private Sub ClearTheDisplayFromChunkIndices()
    For I As Int32 = 1 To MAXNUMLINESINDISPLAY Step 1
        tbDisplayChunkIndex(I).Text = ""
        tbDisplayAddressDec(I).Text = ""
        tbDisplayAddressHex(I).Text = ""
        tbDisplayChunkContents(I).Text = ""
        tbDisplayChunkComment(I).Text = ""
        tbDisplayChunkComment(I).Visible = False
        tbDisplayChunkComment(I).SendToBack()
    Next I
End Sub

Private Sub RenderTheDisplay()
    Dim CurrentAddress As Int32
    Dim CurrentChunk As Int32
    ' Clear the previous display.
    ClearTheDisplayFromChunkIndices()
    ' Retrieve the starting address for the display.
    CurrentAddress = CInt(Val(textboxStartAddress.Text))
    If (CurrentAddress < 0) Then
        CurrentAddress = 0
    End If
    If (CurrentAddress > NumberOfBytesInFile) Then
        CurrentAddress = NumberOfBytesInFile
    End If
    ' Do not proceed if the file is empty.
    If (NumberOfBytesInFile = 0) Then
        MsgBox("The file contains zero bytes.")
        Exit Sub
    End If
    ' Find the index of the chunk containing the starting address.
    CurrentChunk = FindChunkContainingAddress(CurrentAddress)
    If (CurrentChunk = -1) Then
        MsgBox("Error: Logic error at test point #1.")
        Exit Sub
    End If
    If (CurrentChunk = -2) Then
        MsgBox("Error: Could not find the chunk which starts the display.")
        Exit Sub
    End If
    ' Update the starting address.
    CurrentAddress = Chunk(CurrentChunk).StartAddress
    textboxStartAddress.Text = Trim(Str(CurrentAddress))
    ' Main loop to process lines in the display.
    Do
        For I = 1 To MAXNUMLINESINDISPLAY Step 1
            ' Process the current chunk.
            tbDisplayChunkIndex(I).Text = Trim(Str(CurrentChunk))
            tbDisplayAddressDec(I).Text = Trim(Str(CurrentAddress))
            tbDisplayAddressHex(I).Text = ConvertIntToHexString(CurrentAddress)
            tbDisplayChunkContents(I).Text = FormatContentsOfOneChunk(CurrentChunk)
            ' Display the comment if requested.
            If (ShowComments = True) Then
                If (Chunk(CurrentChunk).Comment <> "") Then
                    tbDisplayChunkComment(I).Text = Chunk(CurrentChunk).Comment
                    tbDisplayChunkComment(I).Visible = True
```

```
                    tbDisplayChunkComment(I).BringToFront()
                End If
            End If
            ' Find the following chunk.
            CurrentChunk = FindChunkFollowingAddress(CurrentAddress)
            If (CurrentChunk = -3) Then
                Exit Do
            End If
            If (CurrentChunk = -1) Then
                MsgBox("Error: Logic error at test point #2.")
                Exit Sub
            End If
            If (CurrentChunk = -2) Then
                MsgBox("Error: Could not find the chunk for line " & Trim(Str(I)))
                Exit Sub
            End If
            CurrentAddress = Chunk(CurrentChunk).StartAddress
        Next I
        Exit Do
    Loop
    Me.Refresh()
End Sub

Private Function FormatContentsOfOneChunk(ByVal ChunkNumber As Int32) As String
    ' This function composes the contents of structure Chunk(ChunkNumber) as
    ' a string for display purposes.
    Dim ReturnString As String = ""
    Dim NextByte As Int32
    If (Chunk(ChunkNumber).DisplayFormat = 5) Then
        ' The display is in LittleEndiamDecimal format.
        ' This is only permitted if the chunk is two or four bytes long.
        ' If the chunk has any other length, it is re-formated as Hex.
        If ((Chunk(ChunkNumber).NumBytes <> 2) And _
            (Chunk(ChunkNumber).NumBytes <> 4)) Then
            Chunk(ChunkNumber).DisplayFormat = 1
        Else
            If (Chunk(ChunkNumber).NumBytes = 2) Then
                FormatContentsOfOneChunk = FormatTwoBytesAsLittleEndianDecimal( _
                    Chunk(ChunkNumber).ByteStream(0), _
                    Chunk(ChunkNumber).ByteStream(1))
                Exit Function
            Else
                FormatContentsOfOneChunk = FormatFourBytesAsLittleEndianDecimal( _
                    Chunk(ChunkNumber).ByteStream(0), _
                    Chunk(ChunkNumber).ByteStream(1), _
                    Chunk(ChunkNumber).ByteStream(2), _
                    Chunk(ChunkNumber).ByteStream(3))
                Exit Function
            End If
        End If
    End If
    If (Chunk(ChunkNumber).DisplayFormat = 6) Then
        ' The display is in LittleEndianHex format.
        ' This is only permitted if the chink is two or four bytes long.
        ' If the chunk has any other length, it is re-formated as Hex.
        If ((Chunk(ChunkNumber).NumBytes <> 2) And _
            (Chunk(ChunkNumber).NumBytes <> 4)) Then
            Chunk(ChunkNumber).DisplayFormat = 1
```

```
    Else
        If (Chunk(ChunkNumber).NumBytes = 2) Then
            FormatContentsOfOneChunk = _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(1)) & _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(0))
            Exit Function
        Else
            FormatContentsOfOneChunk = _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(3)) & _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(2)) & _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(1)) & _
                FormatByteAsHex(Chunk(ChunkNumber).ByteStream(0))
            Exit Function
        End If
    End If
End If
If (Chunk(ChunkNumber).DisplayFormat = 1) Then
    ' The display is in Hex format.
    For I As Int32 = 1 To Chunk(ChunkNumber).NumBytes Step 1
        NextByte = Chunk(ChunkNumber).ByteStream(I - 1)
        ReturnString = ReturnString & FormatByteAsHex(NextByte) & " "
    Next I
    If (Len(ReturnString) > 108) Then
        ReturnString = Strings.Left(ReturnString, 105) & "..."
    End If
    FormatContentsOfOneChunk = ReturnString
    Exit Function
End If
If (Chunk(ChunkNumber).DisplayFormat = 2) Then
    ' The display is in ASCII format.
    For I As Int32 = 1 To Chunk(ChunkNumber).NumBytes Step 1
        NextByte = Chunk(ChunkNumber).ByteStream(I - 1)
        ' Circumvent control codes and the delete code.
        If ((NextByte > 31) And (NextByte <> 177)) Then
            ReturnString = ReturnString & FormatByteAsASCII(NextByte)
        Else
            ReturnString = ReturnString & "."
        End If
    Next I
    If (Len(ReturnString) > 108) Then
        ReturnString = Strings.Left(ReturnString, 105) & "..."
    End If
    FormatContentsOfOneChunk = ReturnString
    Exit Function
End If
If (Chunk(ChunkNumber).DisplayFormat = 3) Then
    ' The display is in Binary format.
    For I As Int32 = 1 To Chunk(ChunkNumber).NumBytes Step 1
        NextByte = Chunk(ChunkNumber).ByteStream(I - 1)
        ReturnString = ReturnString & FormatByteAsBinary(NextByte) & " "
    Next I
    If (Len(ReturnString) > 108) Then
        ReturnString = Strings.Left(ReturnString, 105) & "..."
    End If
    FormatContentsOfOneChunk = ReturnString
    Exit Function
End If
If (Chunk(ChunkNumber).DisplayFormat = 4) Then
```

```
        ' The display is in Decimal format.
        For I As Int32 = 1 To Chunk(ChunkNumber).NumBytes Step 1
            NextByte = Chunk(ChunkNumber).ByteStream(I - 1)
            ReturnString = ReturnString & FormatByteAsDecimal(NextByte) & " "
        Next I
        If (Len(ReturnString) > 108) Then
            ReturnString = Strings.Left(ReturnString, 105) & "..."
        End If
        FormatContentsOfOneChunk = ReturnString
        Exit Function
    End If
    FormatContentsOfOneChunk = "Dummy return"
End Function

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'// General-purpose subroutines.
'// 1. FormatByteAsHex(Int32) As String
'// 2. FormatByteAsASCII(Int32) As String
'// 3. FormatByteAsBinary(Int32) As String
'// 4. FormatByteAsDecimal(Int32) As String
'// 5. FormatTwoBytesAsLittleEndianDecimal(Int32, Int32) As String
'// 6. FormatFourBytesAsLittleEndianDecimal(Int32, Int32, Int32, Int32) As String
'// 7. ConvertIntToHexString(Int32) As String
'// 8. ConvertHexStringToInt(String) As Int32

Private Function FormatByteAsHex(ByVal ByteValue As Int32) As String
    ' This function returns a two-character string containing the hex
    ' representation of ByteValue.  This function returns "FAILURE"
    ' if ByteValue is not in the range 0-255.
    Dim HighNibble As Int32
    Dim LowNibble As Int32
    Dim HighNibbleString As String
    Dim LowNibbleString As String
    If ((ByteValue < 0) Or (ByteValue > 255)) Then
        FormatByteAsHex = "FAILURE"
        Exit Function
    End If
    HighNibble = CInt(Math.Floor(ByteValue / 16))
    LowNibble = ByteValue - (HighNibble * 16)
    Select Case HighNibble
        Case 0
            HighNibbleString = "0"
        Case 1
            HighNibbleString = "1"
        Case 2
            HighNibbleString = "2"
        Case 3
            HighNibbleString = "3"
        Case 4
            HighNibbleString = "4"
        Case 5
            HighNibbleString = "5"
        Case 6
            HighNibbleString = "6"
        Case 7
            HighNibbleString = "7"
        Case 8
```

```
            HighNibbleString = "8"
        Case 9
            HighNibbleString = "9"
        Case 10
            HighNibbleString = "A"
        Case 11
            HighNibbleString = "B"
        Case 12
            HighNibbleString = "C"
        Case 13
            HighNibbleString = "D"
        Case 14
            HighNibbleString = "E"
        Case 15
            HighNibbleString = "F"
        Case Else
            HighNibbleString = "*"
    End Select
    Select Case LowNibble
        Case 0
            LowNibbleString = "0"
        Case 1
            LowNibbleString = "1"
        Case 2
            LowNibbleString = "2"
        Case 3
            LowNibbleString = "3"
        Case 4
            LowNibbleString = "4"
        Case 5
            LowNibbleString = "5"
        Case 6
            LowNibbleString = "6"
        Case 7
            LowNibbleString = "7"
        Case 8
            LowNibbleString = "8"
        Case 9
            LowNibbleString = "9"
        Case 10
            LowNibbleString = "A"
        Case 11
            LowNibbleString = "B"
        Case 12
            LowNibbleString = "C"
        Case 13
            LowNibbleString = "D"
        Case 14
            LowNibbleString = "E"
        Case 15
            LowNibbleString = "F"
        Case Else
            LowNibbleString = "*"
    End Select
    FormatByteAsHex = HighNibbleString & LowNibbleString
End Function

Private Function FormatByteAsASCII(ByVal ByteValue As Int32) As String
```

```
    ' This function returns a single-character string containing the ASCII
    ' representation of ByteValue.  This function returns "FAILURE"
    ' if ByteValue is not in the range 0-255.
    If ((ByteValue < 0) Or (ByteValue > 255)) Then
        FormatByteAsASCII = "FAILURE"
        Exit Function
    End If
    FormatByteAsASCII = Microsoft.VisualBasic.ChrW(ByteValue)
End Function

Private Function FormatByteAsBinary(ByVal ByteValue As Int32) As String
    ' This function returns an eight-digit string containing the binary
    ' representation of ByteValue.  This function returns "FAILURE"
    ' if ByteValue is not in the range 0-255.
    Dim TestBit As Int32
    Dim ReturnString As String = ""
    If ((ByteValue < 0) Or (ByteValue > 255)) Then
        FormatByteAsBinary = "FAILURE"
        Exit Function
    End If
    TestBit = 128
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 64
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 32
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 16
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 8
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 4
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 2
    If ((TestBit And ByteValue) = TestBit) Then
```

```
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    TestBit = 1
    If ((TestBit And ByteValue) = TestBit) Then
        ReturnString = ReturnString & "1"
    Else
        ReturnString = ReturnString & "0"
    End If
    FormatByteAsBinary = ReturnString
End Function

Private Function FormatByteAsDecimal(ByVal ByteValue As Int32) As String
    ' This function returns a string containing the decimal representation
    ' of ByteValue.  This function returns "FAILURE" if ByteValue is not
    ' in the range 0-255.
    If ((ByteValue < 0) Or (ByteValue > 255)) Then
        FormatByteAsDecimal = "FAILURE"
        Exit Function
    End If
    FormatByteAsDecimal = Trim(Str(ByteValue))
End Function

Private Function FormatTwoBytesAsLittleEndianDecimal( _
    ByVal ByteValue1 As Int32, ByVal ByteValue2 As Int32) As String
    ' This function returns a string containing the decimal representation
    ' of the two ByteValues, in reverse order.  This function returns
    ' "FAILURE" if either of the ByteValues is not in the range 0-255.
    Dim tempInteger As Int32
    If ((ByteValue1 < 0) Or (ByteValue1 > 255)) Then
        FormatTwoBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
    If ((ByteValue2 < 0) Or (ByteValue2 > 255)) Then
        FormatTwoBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
    tempInteger = (ByteValue2 * 256) + ByteValue1
    FormatTwoBytesAsLittleEndianDecimal = _
        Trim(FormatNumber(tempInteger, 0, TriState.True, , TriState.True))
End Function

Private Function FormatFourBytesAsLittleEndianDecimal( _
ByVal ByteValue1 As Int32, ByVal ByteValue2 As Int32, _
ByVal ByteValue3 As Int32, ByVal ByteValue4 As Int32) As String
    ' This function returns a string containing the decimal representation
    ' of the four ByteValues, in reverse order.  This function returns
    ' "FAILURE" if any of the ByteValues is not in the range 0-255.
    Dim tempInteger As Int32
    If ((ByteValue1 < 0) Or (ByteValue1 > 255)) Then
        FormatFourBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
    If ((ByteValue2 < 0) Or (ByteValue2 > 255)) Then
        FormatFourBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
```

```
    If ((ByteValue3 < 0) Or (ByteValue3 > 255)) Then
        FormatFourBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
    If ((ByteValue4 < 0) Or (ByteValue4 > 255)) Then
        FormatFourBytesAsLittleEndianDecimal = "FAILURE"
        Exit Function
    End If
    tempInteger = (ByteValue2 * 256) + ByteValue1
    tempInteger = (ByteValue3 * 256 * 256) + tempInteger
    tempInteger = (ByteValue4 * 256 * 256 * 256) + tempInteger
    FormatFourBytesAsLittleEndianDecimal = _
        Trim(FormatNumber(tempInteger, 0, TriState.True, , TriState.True))
End Function

Private Function ConvertIntToHexString(ByVal WholeNumber As Int32) As String
    ' This function returns a string containing the hex representation of
    ' WholeNumber.  This function returns "FAILURE" if WholeNumber is not
    ' in the range 0-2,147,483,647.  The string returned has the form
    ' "xx xx xx xxH", where leading pairs which are "00" are suppressed.
    Dim Byte4 As Int32            ' Most significant byte
    Dim Byte3 As Int32
    Dim Byte2 As Int32
    Dim Byte1 As Int32            ' Least significant byte
    Dim Byte4String As String
    Dim Byte3String As String
    Dim Byte2String As String
    Dim Byte1String As String
    Dim ReturnString As String = ""
    If ((WholeNumber < 0) Or (WholeNumber > 2147483647)) Then
        ConvertIntToHexString = "FAILURE"
        Exit Function
    End If
    Byte4 = CInt(Math.Floor(WholeNumber / 16777216))      ' 256^3 = 16777216
    WholeNumber = WholeNumber - (Byte4 * 16777216)
    Byte3 = CInt(Math.Floor(WholeNumber / 65536))         ' 256^2 = 65536
    WholeNumber = WholeNumber - (Byte3 * 65536)
    Byte2 = CInt(Math.Floor(WholeNumber / 256))           ' 256^1 = 256
    Byte1 = WholeNumber - (Byte2 * 256)
    Byte4String = FormatByteAsHex(Byte4)
    Byte3String = FormatByteAsHex(Byte3)
    Byte2String = FormatByteAsHex(Byte2)
    Byte1String = FormatByteAsHex(Byte1)
    ReturnString = Byte1String & "H"
    If ((Byte4String = "00") And (Byte3String = "00") And (Byte2String = "00")) Then
        ConvertIntToHexString = ReturnString
        Exit Function
    End If
    ReturnString = Byte2String & " " & ReturnString
    If ((Byte4String = "00") And (Byte3String = "00")) Then
        ConvertIntToHexString = ReturnString
        Exit Function
    End If
    ReturnString = Byte3String & " " & ReturnString
    If (Byte4String = "00") Then
        ConvertIntToHexString = ReturnString
        Exit Function
    End If
```

```
        ReturnString = Byte4String & " " & ReturnString
        ConvertIntToHexString = ReturnString
    End Function

    Private Function ConvertHexStringToInt(ByVal HexString As String) As Int32
        ' This function returns an integer with is the decimal representation of
        ' HexString.  This function returns -1 on failure.  This function removes
        ' all blank spaces from HexString.  It also removes any occurrence of "H"
        ' or "h" or "0x".
        Dim HexNibble As String
        Dim ReturnValue As Int32 = 0
        HexString = Strings.Replace(HexString, " ", "")
        HexString = Strings.Replace(HexString, "H", "")
        HexString = Strings.Replace(HexString, "h", "")
        HexString = Strings.Replace(HexString, "0x", "")
        HexString = Trim(HexString)
        If (Len(HexString) = 0) Then
            ConvertHexStringToInt = 0
            Exit Function
        End If
        Try
            For I As Int32 = 1 To Len(HexString) Step 1
                HexNibble = Strings.Mid(HexString, I, 1)
                Select Case HexNibble
                    Case "0"
                        ReturnValue = 0 + (16 * ReturnValue)
                    Case "1"
                        ReturnValue = 1 + (16 * ReturnValue)
                    Case "2"
                        ReturnValue = 2 + (16 * ReturnValue)
                    Case "3"
                        ReturnValue = 3 + (16 * ReturnValue)
                    Case "4"
                        ReturnValue = 4 + (16 * ReturnValue)
                    Case "5"
                        ReturnValue = 5 + (16 * ReturnValue)
                    Case "6"
                        ReturnValue = 6 + (16 * ReturnValue)
                    Case "7"
                        ReturnValue = 7 + (16 * ReturnValue)
                    Case "8"
                        ReturnValue = 8 + (16 * ReturnValue)
                    Case "9"
                        ReturnValue = 9 + (16 * ReturnValue)
                    Case "A"
                        ReturnValue = 10 + (16 * ReturnValue)
                    Case "B"
                        ReturnValue = 11 + (16 * ReturnValue)
                    Case "C"
                        ReturnValue = 12 + (16 * ReturnValue)
                    Case "D"
                        ReturnValue = 13 + (16 * ReturnValue)
                    Case "E"
                        ReturnValue = 14 + (16 * ReturnValue)
                    Case "F"
                        ReturnValue = 15 + (16 * ReturnValue)
                    Case Else
                        ConvertHexStringToInt = -1
```

```
                    Exit Function
                End Select
            Next I
        Catch ex As Exception
            ConvertHexStringToInt = -1
        End Try
        ConvertHexStringToInt = ReturnValue
    End Function

    '////////////////////////////////////////////////////////////////////////////////
    '////////////////////////////////////////////////////////////////////////////////
    '// Subroutine for debugging purposes.

    Private Sub DebugDisplayTwoStructures( _
        ByVal Index1 As Int32, ByVal Index2 As Int32)
        Dim DisplayString As String = ""
        If ((Index1 < 0) Or (Index1 > NumberOfChunks)) Then
            Exit Sub
        End If
        DisplayString = DisplayString & _
            "Structure(index=" & Trim(Str(Index1)) & ")" & vbCrLf & _
            "  .StartAddress=" & Trim(Str(Chunk(Index1).StartAddress)) & vbCrLf & _
            "  .Length=" & Trim(Str(Chunk(Index1).NumBytes)) & vbCrLf & _
            "  .EndAddress=" & Trim(Str(Chunk(Index1).EndAddress)) & vbCrLf & _
            "  .Active=" & Chunk(Index1).Active.ToString & vbCrLf & _
            "  .DisplayFormat=" & Trim(Str(Chunk(Index1).DisplayFormat) & vbCrLf & _
            "  .Comment=" & Chunk(Index1).Comment)
        If ((Index2 > 0) And (Index2 <= NumberOfChunks)) Then
            DisplayString = DisplayString & vbCrLf & _
            "Structure(index=" & Trim(Str(Index2)) & ")" & vbCrLf & _
            "  .StartAddress=" & Trim(Str(Chunk(Index2).StartAddress)) & vbCrLf & _
            "  .Length=" & Trim(Str(Chunk(Index2).NumBytes)) & vbCrLf & _
            "  .EndAddress=" & Trim(Str(Chunk(Index2).EndAddress)) & vbCrLf & _
            "  .Active=" & Chunk(Index2).Active.ToString & vbCrLf & _
            "  .DisplayFormat=" & Trim(Str(Chunk(Index2).DisplayFormat) & vbCrLf & _
            "  .Comment=" & Chunk(Index2).Comment)
        End If
        MsgBox(DisplayString)
    End Sub

End Class
```