Sherlock 7 Technical Resource

Teledyne DALSA IPD
www.teledynedalsa.com/ipd 978.670.2002 (U.S.A.)

Document Revision: August 9, 2012

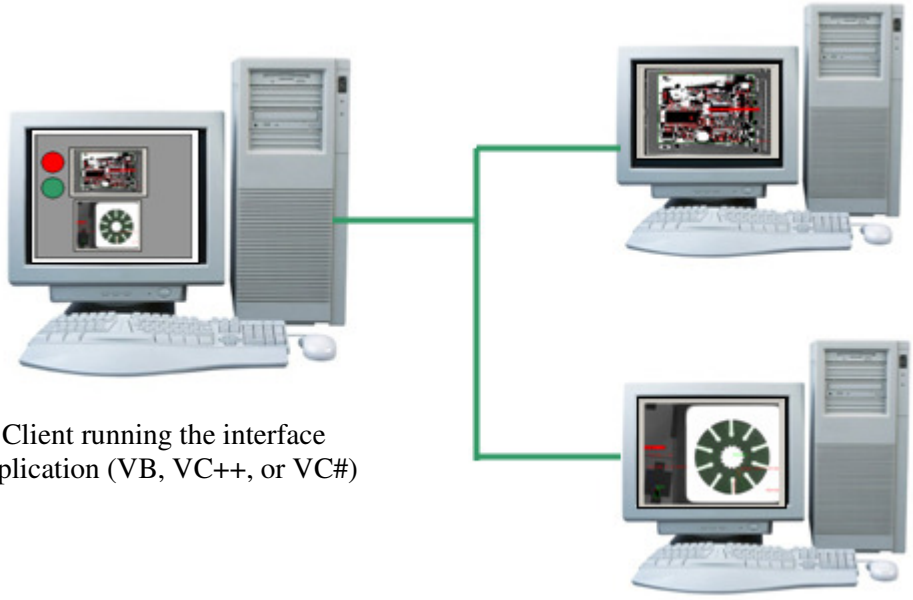
Remote Access from a Custom Interface

Although a Sherlock 7 investigation can be run directly from the Sherlock GUI, it is often desirable, and sometimes necessary, to hide the Sherlock GUI behind a custom interface. Separate documents explain how to create custom interface applications:

Developing a VB 6 interface for Sherlock 7.ppt
Developing a VB 2005 interface for Sherlock 7.ppt
Developing a VC# 2005 interface for Sherlock 7.ppt.

(The instructions in these documents can be adapted for Visual C++ 6.0, Visual C# 6.0 and Visual C++ 2005.)

These documents assume that the interface application is running on the same computer as the Sherlock investigation. However, it is possible for an interface application on one computer (the client) to launch and control one or more Sherlock investigations on different computers (the servers).



Client running the interface application (VB, VC++, or VC#)

Servers running the Sherlock investigations

Requirements

The tools for remote access to a Sherlock 7 investigation were first released with Sherlock 7.1.1.0. Earlier versions of Sherlock 7 do not support remote access.

The same release of Sherlock must be installed on both the client and the server. The client does not have to be licensed to run Sherlock.

Sherlock investigation

You do not have to modify in any way a Sherlock investigation that will be remotely accessed from an interface application running on a client; develop the investigation as you would if it and the interface application were going to run on the same computer.

Interface application

An interface application running on a client is virtually the same as one running on the server with the Sherlock investigation. The only necessary modification is the addition of a call to connect to the server. This call is a method of the **IpeEngCtrl** object:

```
.Connect(strServer, nPort)
```

where **strServer** is a string specifying the name or internet protocol address (IPA) of the server, and **nPort** is the port number on the server through which the two computers will communicate (5100 by default).

IpeServer

On the server, you must launch **IpeServer** before invoking the `.Connect` method on the client. IpeServer can be found in the **<Sherlock>\bin** directory.

Code examples

Examples are written in Visual Basic 2005, but they are easily ported to the other Visual Studio 6 and Visual Studio 2005 languages. These are not complete examples; they show only how to set up a connection to a server running Sherlock.

The **New()** routine is a good place to connect to the server.

```
Dim WithEvents hSherlock As IpeEngCtrl.Engine
Dim nErr As IpeEngCtrlLib.I_ENG_ERROR
Dim SherlockType As Type

Public Sub New()
  ' This call is required by the Windows Form Designer.
  InitializeComponent()

  ' Add any initialization after the InitializeComponent() call.
  SherlockType = Type.GetTypeFromProgID("IpeEngCtrl.Engine")

  'Create an instance of the Sherlock object
  hSherlock = Activator.CreateInstance(SherlockType)
  If Not (hSherlock Is Nothing) Then
      ' Connect to the server by IPA
      nErr = hSherlock.Connect("192.168.10.121", 5100)
      ' You can also connect to the server by name
      ' nErr = hSherlock.Connect("NS-550025", 5100)
      If Not (nErr = IpeEngCtrl.I_ENG_ERROR.I_OK) Then
         MsgBox("Error connecting to server.", MsgBoxStyle.Critical, "Error")
         Me.Close()
      End If
      'Initialize the Sherlock engine
      nErr = hSherlock.EngInitialize()
  Else
      MsgBox("Error creating Sherlock object.", MsgBoxStyle.Critical, "Error")
      Me.Close()
      End
  End If

  ' Load the investigation from the server's C drive
  nErr = hSherlock.InvLoad("C:\MyApps\Bottle top.ivs")
.
.
```

```
.
End Sub
```

Connecting to multiple servers

To connect to multiple servers from one client interface, you must create a separate instance of the Sherlock object for each server.

```
Dim WithEvents hSherlock_A As IpeEngCtrl.Engine
Dim WithEvents hSherlock_B As IpeEngCtrl.Engine

Public Sub New()
.
.
.
  'Create two instances of the Sherlock object
  hSherlock_A = Activator.CreateInstance(SherlockType)
  hSherlock_B = Activator.CreateInstance(SherlockType)
.
.
.
  nErr = hSherlock_A.Connect("192.168.10.120", 5100)
  nErr = hSherlock_B.Connect("192.168.10.132", 5100)
.
.
.
  nErr = hSherlock_A.InvLoad("C:\LineA\Cap placement.ivs")
  nErr = hSherlock_B.InvLoad("D:\Labeler\Label position.ivs")
.
.
.
End Sub()
```

Image display

Compression

A client interface application can connect to and display image windows in a server investigation just as when the interface application and the investigation are on the same computer. However, transmitting images across a network consumes considerable bandwidth. Since an interface application pauses upon every call to the **IpeEngCtrl RunCompleted( )** method (which is automatically invoked at the end of every iteration through the investigation) until all the requested data and images have been transferred, an interface that displays every image acquired into an image window will not run at the speed of the investigation running by itself.

To decrease the amount of data being transmitted, you can compress the contents of Sherlock's image windows on the server before they are transmitted across the network, and decompress them on the client before they are displayed. This is achieved by invoking the IpeEngCtrl **Set Remote Display Compression** method:

.SetRemDispCompression(bEnable, nCompression)

where **nCompression** is a number from 0 to 100

0 : images not transmitted

1 : high compression, worst image quality

…

100 : no compression, best image quality
and **bEnable** is a Boolean value. If bEnable is True, compression is performed; if it is False, compression is not performed.

```
nErr = hSherlock.SetRemDispCompression(True, 50)
```

Note that only the contents of image windows that are connected to display controls on the interface application are transmitted. If the Sherlock investigation has image windows imgA, imgB, and imgC, but only imgB is connected to a display control on the interface application, only the contents of imgB are transmitted.

A complementary method returns the compression settings:

```
.GetRemDispCompression(bEnable, nCompression)
```

If compression is enabled, True is returned in **bEnable**; if compression is not enabled, False is returned. The current compression factor is returned in **nCompression**, whether or not compression is enabled.

```
Dim bEnable as Boolean
Dim nCompression as Integer
nErr = hSherlock.GetRemDispCompression(bEnable, nCompression)
```

Graphics

To further decrease image transmission time, you can disable transmission of an image window's overlay graphics (ROIs and annotations) by invoking the IpeEngCtrl **Set Remote Display Graphics** method:

```
.SoSetRemDispGraphics(strImageWindow, bEnable)
```

where **strImageWindow** is a string specifying the name of the image window, and **bEnable** is a Boolean value. If bEnable is True, the graphics are transmitted with the image; if it is False, the graphics are not transmitted.

```
nErr = hSherlock.SoSetRemDispGraphics("imgA", True)
nErr = hSherlock.SoSetRemDispGraphics("imgD", False)
```

A complementary method returns the graphics setting:

```
.SoGetRemDispGraphics(strImageWindow, bEnable)
```

where **strImageWindow** is a string specifying the name of the image window, and **bEnable** is a Boolean in which the current setting for the image window is returned.

```
Dim bEnable as Boolean
nErr = hSherlock.SoGetRemDispGraphics("imgB", bEnable)
```

Acquiring without processing

To acquire and display images without processing them, call the method

```
.SoLiveSet(strImageWindow, bEnable)
```

where **strImageWindow** is a string specifying the name of the image window, and **bEnable** is a Boolean value. If bEnable is True, live acquisition without processing is started; if it is False, live acquisition is halted.

```
nErr = hSherlock.SoLiveSet("imgA", True)
```

 In Visual Basic 6, the Boolean parameters in the methods described above are passed and received as **Long** values, where 0 = False and 1 = True.