```
1  / Fig. 16.1: LoadImageAndScale.java
2  // Load an image and display it in its original size
3  // and scale it to twice its original width and height.
4  // Load and display the same image as an ImageIcon.
5  import java.applet.Applet;
6  import java.awt.*;
7  import javax.swing.*;
8
9  public class LoadImageAndScale extends JApplet {
10    private Image logo1;
11    private ImageIcon logo2;
12
13    // load the image when the applet is loaded
14    public void init()
15    {
16       logo1 = getImage( getDocumentBase(), "logo.gif" );
17       logo2 = new ImageIcon( "logo.gif" );
18    }
19
20    // display the image
21    public void paint( Graphics g )
22    {
23       // draw the original image
24       g.drawImage( logo1, 0, 0, this );
25
26       // draw the image scaled to fit the width of the applet
27       // and the height of the applet minus 120 pixels
28       g.drawImage( logo1, 0, 120,
29                    getWidth(), getHeight() - 120, this );
30
31       // draw the icon using its paintIcon method
32       logo2.paintIcon( this, g, 180, 0 );
33    }
34 }
```

Fig. 16.1 Loading and displaying an image in an applet (part 1 of 2).

Fig. 16.1 Loading and displaying an image in an applet (part 2 of 2).

```
1  // Fig. 16.2: LoadAudioAndPlay.java
2  // Load an audio clip and play it.
3  import java.applet.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
```

Fig. 16.2 Loading and playing an **`AudioClip`** (part 1 of 3).

```
7
8  public class LoadAudioAndPlay extends JApplet {
9     private AudioClip sound1, sound2, currentSound;
10    private JButton playSound, loopSound, stopSound;
11    private JComboBox chooseSound;
12
13    // load the image when the applet begins executing
14    public void init()
15    {
16       Container c = getContentPane();
17       c.setLayout( new FlowLayout() );
18
19       String choices[] = { "Welcome", "Hi" };
20       chooseSound = new JComboBox( choices );
21       chooseSound.addItemListener(
22          new ItemListener() {
23             public void itemStateChanged( ItemEvent e )
24             {
25                currentSound.stop();
26
27                currentSound =
28                   chooseSound.getSelectedIndex() == 0 ?
29                      sound1 : sound2;
30             }
31          }
32       );
33       c.add( chooseSound );
34
35       ButtonHandler handler = new ButtonHandler();
36       playSound = new JButton( "Play" );
37       playSound.addActionListener( handler );
38       c.add( playSound );
39       loopSound = new JButton( "Loop" );
40       loopSound.addActionListener( handler );
41       c.add( loopSound );
42       stopSound = new JButton( "Stop" );
43       stopSound.addActionListener( handler );
44       c.add( stopSound );
45
46       sound1 = getAudioClip(
47                   getDocumentBase(), "welcome.wav" );
48       sound2 = getAudioClip(
49                   getDocumentBase(), "hi.au" );
50       currentSound = sound1;
```

```
51      }
52
53      // stop the sound when the user switches Web pages
54      // (i.e., be polite to the user)
55      public void stop()
56      {
57         currentSound.stop();
58      }
59
```

Fig. 16.2 Loading and playing an **`AudioClip`** (part 2 of 3).

```
60      private class ButtonHandler implements ActionListener {
61         public void actionPerformed( ActionEvent e )
62         {
63            if ( e.getSource() == playSound )
64               currentSound.play();
65            else if ( e.getSource() == loopSound )
66               currentSound.loop();
67            else if ( e.getSource() == stopSound )
68               currentSound.stop();
69         }
70      }
71   }
```

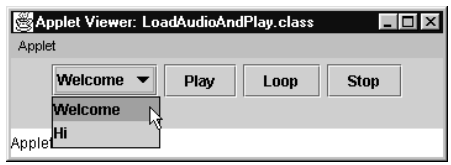


Fig. 16.2 Loading and playing an **`AudioClip`** (part 3 of 3).

```
1  // Fig. 16.3: MediaPlayerDemo.java
2  // Uses a Java Media Player to play media files.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.io.*;
6  import javax.swing.*;
7  import javax.media.*;
8
9  public class MediaPlayerDemo extends JFrame {
10    private Player player;
11    private File file;
12
13    public MediaPlayerDemo()
14    {
15       super( "Demonstrating the Java Media Player" );
16
17       JButton openFile = new JButton( "Open file to play" );
18       openFile.addActionListener(
19          new ActionListener() {
20             public void actionPerformed( ActionEvent e )
21             {
22                openFile();
23                createPlayer();
24             }
25          }
26       );
27       getContentPane().add( openFile, BorderLayout.NORTH );
28
29       setSize( 300, 300 );
30       show();
31    }
32
33    private void openFile()
34    {
35       JFileChooser fileChooser = new JFileChooser();
36
37       fileChooser.setFileSelectionMode(
38          JFileChooser.FILES_ONLY );
39       int result = fileChooser.showOpenDialog( this );
40
```

Fig. 16.3 Demonstrating the Java Media Player (part 1 of 4).

```
41       // user clicked Cancel button on dialog
42       if ( result == JFileChooser.CANCEL_OPTION )
43          file = null;
44       else
45          file = fileChooser.getSelectedFile();
46    }
47
```

```
48      private void createPlayer()
49      {
50         if ( file == null )
51            return;
52
53         removePreviousPlayer();
54
55         try {
56            // create a new player and add listener
57            player = Manager.createPlayer( file.toURL() );
58            player.addControllerListener( new EventHandler() );
59            player.start();  // start player
60         }
61         catch ( Exception e ){
62            JOptionPane.showMessageDialog( this,
63               "Invalid file or location", "Error loading file",
64               JOptionPane.ERROR_MESSAGE );
65         }
66      }
67
68      private void removePreviousPlayer()
69      {
70         if ( player == null )
71            return;
72
73         player.close();
74
75         Component visual = player.getVisualComponent();
76         Component control = player.getControlPanelComponent();
77
78         Container c = getContentPane();
79
80         if ( visual != null )
81            c.remove( visual );
82
83         if ( control != null )
84            c.remove( control );
85      }
86
87      public static void main(String args[])
88      {
89         MediaPlayerDemo app = new MediaPlayerDemo();
90
```

Fig. 16.3 Demonstrating the Java Media Player (part 2 of 4).

```
91         app.addWindowListener(
92            new WindowAdapter() {
93               public void windowClosing( WindowEvent e )
94               {
95                  System.exit(0);
96               }
```

```
97             }
98          );
99       }
100
101      // inner class to handler events from media player
102      private class EventHandler implements ControllerListener {
103         public void controllerUpdate( ControllerEvent e ) {
104            if ( e instanceof RealizeCompleteEvent ) {
105               Container c = getContentPane();
106
107               // load Visual and Control components if they exist
108               Component visualComponent =
109                  player.getVisualComponent();
110
111               if ( visualComponent != null )
112                  c.add( visualComponent, BorderLayout.CENTER );
113
114               Component controlsComponent =
115                  player.getControlPanelComponent();
116
117               if ( controlsComponent != null )
118                  c.add( controlsComponent, BorderLayout.SOUTH );
119
120               c.doLayout();
121            }
122         }
123      }
124   }
```

Initial GUI at execution

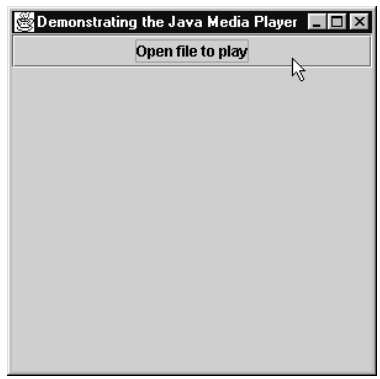
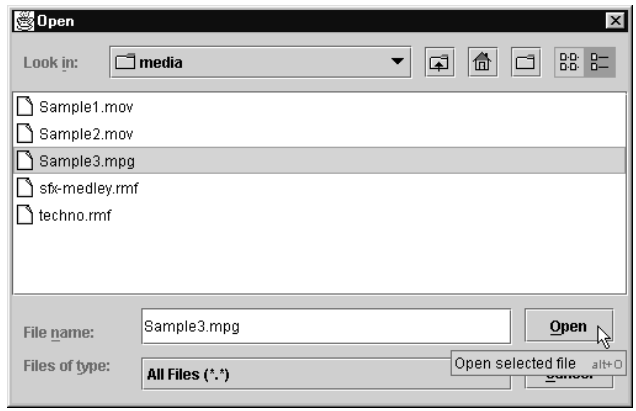


Fig. 16.3 Demonstrating the Java Media Player (part 3 of 4).

Selecting a media file to play from a `JFileChooser` dialog box



The `Sample3.mpg` file loaded and playing

Fig. 16.3 Demonstrating the Java Media Player (part 4 of 4).

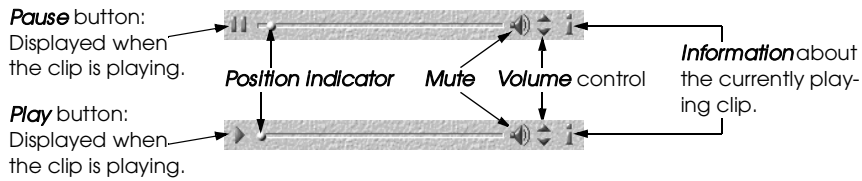


Fig. 16.4 Controls for the Java Media Player when an audio or video is playing.

```
1  // Fig. 16.5: LogoAnimator.java
2  // Animation a series of images
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LogoAnimator extends JPanel
8                            implements ActionListener {
9     protected ImageIcon images[];
10    protected int totalImages = 30,
11                  currentImage = 0,
12                  animationDelay = 50; // 50 millisecond delay
13    protected Timer animationTimer;
```

Fig. 16.5 Animating a series of images (part 1 of 3).

```
14
15    public LogoAnimator()
16    {
17       setSize( getPreferredSize() );
18
19       images = new ImageIcon[ totalImages ];
20
21       for ( int i = 0; i < images.length; ++i )
22          images[ i ] =
23             new ImageIcon( "images/deitel" + i + ".gif" );
24
25       startAnimation();
26    }
27
28    public void paintComponent( Graphics g )
29    {
30       super.paintComponent( g );
31
32       if ( images[ currentImage ].getImageLoadStatus() ==
33            MediaTracker.COMPLETE ) {
34          images[ currentImage ].paintIcon( this, g, 0, 0 );
35          currentImage = ( currentImage + 1 ) % totalImages;
36       }
37    }
38
39    public void actionPerformed( ActionEvent e )
40    {
41       repaint();
42    }
43
44    public void startAnimation()
45    {
46       if ( animationTimer == null ) {
47          currentImage = 0;
48          animationTimer = new Timer( animationDelay, this );
49          animationTimer.start();
```

```
50          }
51          else  // continue from last image displayed
52             if ( ! animationTimer.isRunning() )
53                animationTimer.restart();
54       }
55
56       public void stopAnimation()
57       {
58          animationTimer.stop();
59       }
60
61       public Dimension getMinimumSize()
62       {
63          return getPreferredSize();
64       }
65
```

Fig. 16.5 Animating a series of images (part 2 of 3).

```
66       public Dimension getPreferredSize()
67       {
68          return new Dimension( 160, 80 );
69       }
70
71       public static void main( String args[] )
72       {
73          LogoAnimator anim = new LogoAnimator();
74
75          JFrame app = new JFrame( "Animator test" );
76          app.getContentPane().add( anim, BorderLayout.CENTER );
77
78          app.addWindowListener(
79             new WindowAdapter() {
80                public void windowClosing( WindowEvent e )
81                {
82                   System.exit( 0 );
83                }
84             }
85          );
86
87          // The constants 10 and 30 are used below to size the
88          // window 10 pixels wider than the animation and
89          // 30 pixels taller than the animation.
90          app.setSize( anim.getPreferredSize().width + 10,
91                       anim.getPreferredSize().height + 30 );
92          app.show();
93       }
94    }
```

Fig. 16.5 Animating a series of images (part 3 of 3).

```
1  // Fig. 16.6: LogoAnimator.java
2  // Animating a series of images
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
```

Fig. 16.6 Customizing an applet via the **param** HTML tag (part 1 of 5).

```
7  public class LogoAnimator extends JPanel
8                            implements ActionListener {
9     protected ImageIcon images[];
10    protected int totalImages = 30,
11                  currentImage = 0,
12                  animationDelay = 50; // 50 millisecond delay
13    protected String imageName = "deitel";
14    protected Timer animationTimer;
15
16    public LogoAnimator()
17    {
18       initializeAnim();
19    }
20
21    // new constructor to support customization
22    public LogoAnimator( int num, int delay, String name )
23    {
24       totalImages = num;
25       animationDelay = delay;
26       imageName = name;
27
28       initializeAnim();
29    }
30
31    private void initializeAnim()
32    {
33       images = new ImageIcon[ totalImages ];
34
35       for ( int i = 0; i < images.length; ++i )
36          images[ i ] = new ImageIcon( "images/" +
37                                       imageName + i + ".gif" );
38
39       // moved here so getPreferredSize can check the size of
40       // the first loaded image.
41       setSize( getPreferredSize() );
42
43       startAnimation();
44    }
45
46    public void paintComponent( Graphics g )
47    {
48       super.paintComponent( g );
49
```

```
50          if ( images[ currentImage ].getImageLoadStatus() ==
51               MediaTracker.COMPLETE ) {
52             images[ currentImage ].paintIcon( this, g, 0, 0 );
53             currentImage = ( currentImage + 1 ) % totalImages;
54          }
55       }
56
57       public void actionPerformed( ActionEvent e )
58       {
```

Fig. 16.6 Customizing an applet via the **param** HTML tag (part 2 of 5).

```
59          repaint();
60       }
61
62       public void startAnimation()
63       {
64          if ( animationTimer == null ) {
65             currentImage = 0;
66             animationTimer = new Timer( animationDelay, this );
67             animationTimer.start();
68          }
69          else  // continue from last image displayed
70             if ( ! animationTimer.isRunning() )
71                animationTimer.restart();
72       }
73
74       public void stopAnimation()
75       {
76          animationTimer.stop();
77       }
78
79       public Dimension getMinimumSize()
80       {
81          return getPreferredSize();
82       }
83
84       public Dimension getPreferredSize()
85       {
86          return new Dimension( images[ 0 ].getIconWidth(),
87                                images[ 0 ].getIconHeight() );
88       }
89
90       public static void main( String args[] )
91       {
92          LogoAnimator anim = new LogoAnimator();
93
94          JFrame app = new JFrame( "Animator test" );
95          app.getContentPane().add( anim, BorderLayout.CENTER );
96
97          app.addWindowListener(
98             new WindowAdapter() {
99                public void windowClosing( WindowEvent e )
100               {
```

```
                System.exit( 0 );
             }
          }
       );

       app.setSize( anim.getPreferredSize().width + 10,
                    anim.getPreferredSize().height + 30 );
       app.show();
    }
}
```

Fig. 16.6 Customizing an applet via the **param** HTML tag (part 3 of 5).

```
// Fig. 16.6: LogoApplet.java
// Customizing an applet via HTML parameters
//
// HTML parameter "animationdelay" is an int indicating
// milliseconds to sleep between images (default 50).
//
// HTML parameter "imagename" is the base name of the images
// that will be displayed (i.e., "deitel" is the base name
// for images "deitel0.gif," "deitel1.gif," etc.). The applet
// assumes that images are in an "images" subdirectory of
// the directory in which the applet resides.
//
// HTML parameter "totalimages" is an integer representing the
// total number of images in the animation. The applet assumes
// images are numbered from 0 to totalimages - 1 (default 30).

import java.awt.*;
import javax.swing.*;

public class LogoApplet extends JApplet{
   public void init()
   {
      String parameter;

      parameter = getParameter( "animationdelay" );
      int animationDelay = ( parameter == null ? 50 :
                             Integer.parseInt( parameter ) );

      String imageName = getParameter( "imagename" );

      parameter = getParameter( "totalimages" );
      int totalImages = ( parameter == null ? 0 :
                          Integer.parseInt( parameter ) );

      // Create an instance of LogoAnimator
      LogoAnimator animator;

      if ( imageName == null || totalImages == 0 )
         animator = new LogoAnimator();
```

```
150         else
151            animator = new LogoAnimator( totalImages,
152                           animationDelay, imageName );
153
154         setSize( animator.getPreferredSize().width,
155                  animator.getPreferredSize().height );
156         getContentPane().add( animator, BorderLayout.CENTER );
157
158         animator.startAnimation();
159      }
160   }
```

Fig. 16.6 Customizing an applet via the **`param`** HTML tag (part 4 of 5).

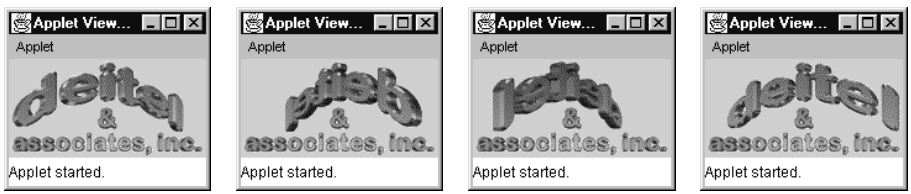


Fig. 16.6 Customizing an applet via the **`param`** HTML tag (part 5 of 5).

```
1  // Fig. 16.7: ImageMap.java
2  // Demonstrating an image map.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class ImageMap extends JApplet {
8     private ImageIcon mapImage;
9     private int width, height;
10
11    public void init()
12    {
13       addMouseListener(
14          new MouseAdapter() {
15             public void mouseExited( MouseEvent e )
16             {
17                showStatus( "Pointer outside applet" );
18             }
19          }
20       );
21
22       addMouseMotionListener(
23          new MouseMotionAdapter() {
24             public void mouseMoved( MouseEvent e )
25             {
26                showStatus( translateLocation( e.getX() ) );
27             }
28          }
29       );
30
31       mapImage = new ImageIcon( "icons2.gif" );
32       width = mapImage.getIconWidth();
33       height = mapImage.getIconHeight();
34       setSize( width, height );
35    }
36
37    public void paint( Graphics g )
38    {
39       mapImage.paintIcon( this, g, 0, 0 );
40    }
41
```

Fig. 16.7 Demonstrating an image map (part 1 of 2).

```
42    public String translateLocation( int x )
43    {
44       // determine width of each icon (there are 6)
45       int iconWidth = width / 6;
46
47       if ( x >= 0 && x <= iconWidth )
48          return "Common Programming Error";
49       else if ( x > iconWidth && x <= iconWidth * 2 )
50          return "Good Programming Practice";
```

```
51          else if ( x > iconWidth * 2 && x <= iconWidth * 3 )
52             return "Performance Tip";
53          else if ( x > iconWidth * 3 && x <= iconWidth * 4 )
54             return "Portability Tip";
55          else if ( x > iconWidth * 4 && x <= iconWidth * 5 )
56             return "Software Engineering Observation";
57          else if ( x > iconWidth * 5 && x <= iconWidth * 6 )
58             return "Testing and Debugging Tip";
59
60          return "";
61       }
62    }
```

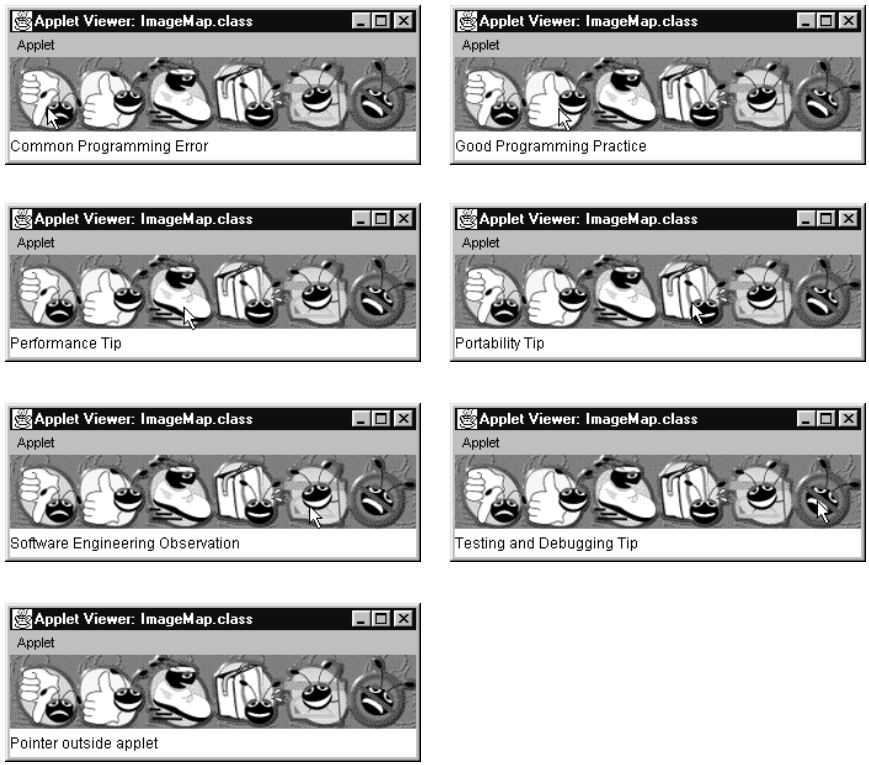


Fig. 16.7 Demonstrating an image map (part 2 of 2).

The Java Plug-in HTML Converter. This program allows you to convert all the HTML files in one directory or select a specific file. In this case, **One File** is selected.

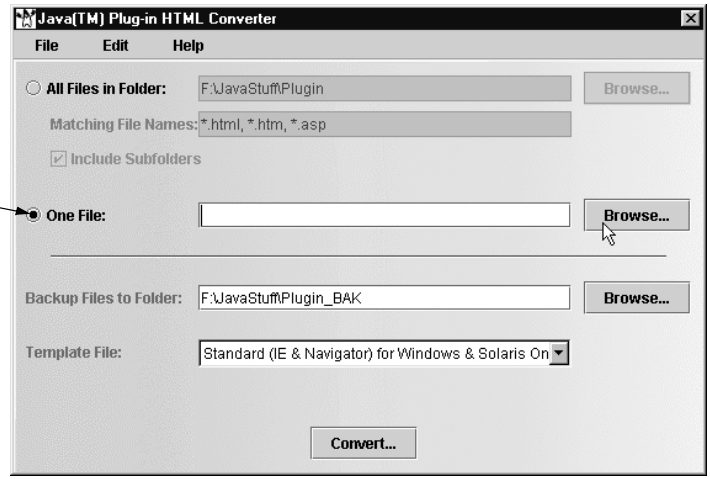


Fig. 16.8 The Java Plug-in HTML Converter (part 1 of 4).

This dialog allows you to select the file that will be converted.

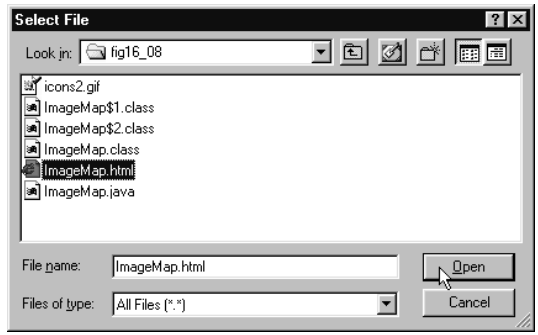
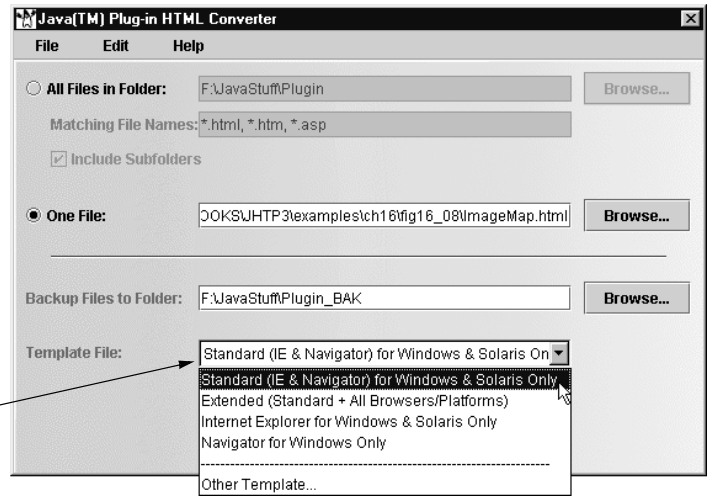


Fig. 16.8 The Java Plug-in HTML Converter (part 2 of 4).

The HTML Converter window after a file is selected for conversion.



The **Template File** combo box allows you to choose the browsers in which the plug-in should be used.

Fig. 16.8 The Java Plug-in HTML Converter (part 3 of 4).

The confirmation dialog showing that one applet was found in the HTML file and converted.

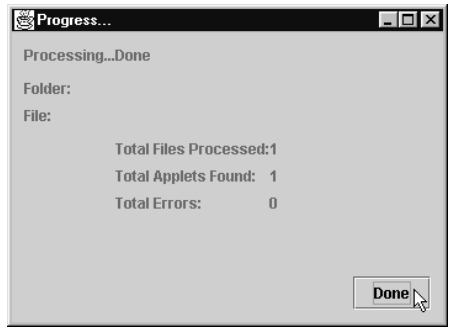


Fig. 16.8 The Java Plug-in HTML Converter (part 4 of 4).