

JAVA DEVELOPER'S GUIDE TO ASPRISE JT WAIN

Version 9

Last updated on January, 2006

ALL RIGHTS RESERVED BY LAB ASPRISE! © 1998, 2006.

Table of Contents

1	INTRODUCTION	4
1.1	ABOUT TWAIN	4
1.2	ABOUT JTWAIN	4
1.3	COMPONENTS OF JTWAIN	4
1.4	JTWAIN SDK INSTALLATION	5
1.5	FILE ORGANIZATION	5
1.6	DEVELOPMENT ENVIRONMENT SETUP	6
1.7	COMPATIBILITY.....	6
2	IMAGE ACQUISITION WITH JTWAIN	7
2.1	FOR THE IMPATIENT	7
3	CONTROL FLOW OF A TYPICAL IMAGE ACQUISITION PROCESS	9
3.1	GETTING A SOURCE.....	9
3.1.1	Gets the default Source	10
3.1.2	Lets the user select a Source	10
3.1.3	Selects source by its name.....	10
3.1.4	Get all the Sources available.....	10
3.1.5	Validating a Source:	11
3.2	HIDING THE USER INTERFACE.....	11
3.2.1	Hiding the "Select Source" UI:	11
3.2.2	Hiding the scanner/digital camera's acquisition UI:	11
3.2.3	Hiding the indicators' UI:.....	11
3.3	SETTING AND GETTING SOURCE CAPABILITIES.....	12
4	ACQUIRING IMAGES	14
4.1	AUTOMATIC DOCUMENT FEEDING (ADF)	14
4.2	SAVING ACQUIRED IMAGES INTO FILES.....	15
4.2.1	Built-in Image Saving Functions	15
4.2.2	Sample Code.....	15
4.3	ACQUIRING A SPECIFIED REGION ONLY	17
4.4	UPLOADING ACQUIRED IMAGES TO WEB SERVERS	18
4.5	ABOUT THE JTWAIN WEB APPLLET	19
5	LOW LEVEL API PROGRAMMING	20
5.1	JTWAIN API MODEL	20

5.2	EXTENDING THE SOURCE	20
5.3	TWAIN & JTWAIN MAPPING.....	22
5.3.1	Containers	22
5.3.2	Item Types.....	22
6	ADVANCED TOPICS	24
6.1	EXCEPTION HANDLING	24
6.2	USING JTWAIN IN THREADS.....	25
6.3	SOFTWARE PACKAGING AND DISTRIBUTION	25
7	DEPLOYMENT GUIDE.....	27
7.1	ORGANIZING YOUR DIRECTORY	27
7.2	CREATING A JAR FILE	28
7.3	CREATING SIGNED APPLETS	28
7.3.1	Creating a Certificate.....	29
7.3.2	Signing Jar Files	29
7.3.3	Launching the Applet.....	30
8	IMAGE ACQUISITION COMPONENTS	31
8.1	JIMAGEDIALOG	31
8.1.1	Advantages.....	32
8.1.2	Sample Uses	32
8.1.3	Supported Image Formats	34
8.1.4	Compatibility	35
8.1.5	Software Packaging and Distribution.....	35
8.2	JIMAGEFILECHOOSER	35
8.2.1	Sample Use	36
8.2.2	Supported Image Formats	36
8.2.3	Compatibility	36
8.2.4	Software Packaging and Distribution.....	36
9	SUPPORT AND PROFESSIONAL SERVICES	37
9.1	SUPPORT WEB SITE	37
9.2	BASIC SUPPORT.....	37
9.3	PREMIUM SUPPORT SERVICES	37
9.4	PROFESSIONAL SERVICES.....	37

1 Introduction

1.1 About TWAIN

The TWAIN initiative was originally launched in 1992 by leading industry vendors who recognized a need for a standard software protocol and applications programming interface (API) that regulates communication between software applications and imaging devices (the source of the data). TWAIN defines that standard. Most of scanners and digital cameras on market are TWAIN compatible. For more information, visit: www.twain.org

1.2 About JTwain

JTwain is the Java counterpart of TWAIN. It is a TWAIN suite developed by LAB Asprise! since 1998. It is 100% TWAIN (most updated version 1.9) compatible. JTwain enables Java developers to acquire images from scanners and digital cameras easily. Its universal APIs bridge Java and scanners, digital cameras tightly. With more than five years extensive development, LAB Asprise! proudly presents you the long waiting version 9 of JTwain.

1.3 Components of JTwain

JTwain comprises two components:

- ◆ A native library: *AspriseJTwain.dll*
- ◆ Several Java packages:
 - *com.asprise.util.jtwain* - main package; contains essential classes to perform image acquisition
 - *com.asprise.util.jtwain.lowlevel* – low-level APIs for advanced development of TWAIN applications
 - *com.asprise.util.ui* – optional UI components
 - *com.asprise.util.jtwain.web* – classes for uploading images to web servers

1.4 JTwain SDK Installation

First, make sure that you have already installed Java runtime version 1.2 or above on your system. Currently, JTwain only support the following OSs: Windows 98, NT, ME, 2000, XP and all Windows Server platforms. Other OSs are planed.

Download a copy of JTwain installation file from <http://www.asprise.com/product/jtwain>. Unzip the SDK kit to a folder, which we will refer as *JTWAIN_HOME*.

After the installation, double click *LaunchDemo.bat* to test your installation. Select 'TestJTwain' and click 'Launch!' button to test JTwain. You should see some output looks like this:

```
1. Executing TestJTwain ...
2. ----- JTWAIN TEST -----
3.
4. Testing results:
5. --- MESSG) System Java VM Version: 1.4.2_01-b06
6. --- MESSG) JTWAIN Supports Java 1.2 and above.
7.
8. --- MESSG) JTWAIN DLL version: 9 EVALUATION / LICENSED
9. --- OK) Source manager has been successfully loaded.
10.
11. --- OK) Source #0: Source: Microtek ScanWizard
12. --- OK) Source #1: Source: TWAIN_32 Sample Source
13.
14. ----- END OF TEST -----
```

A proper installation results no ERROR messages.

1.5 File Organization

The file organization of JTwain SDK distribution is as follows:

JTWAIN_HOME

- +--- AspriseJTwain-DevGuide.pdf [Developer's Guide]
- +--- api [Java docs]
- +--- AspriseJTwain.dll [The sole native library]
- +--- JTwain.jar [Contains all JTwain classes]
- +--- demo.jar [Contains binary class files and source code for demo programs]

+--- demo-src.jar [Contains the source code for all the demo programs]
+--- LaunchDemo.bat [Launches JTwain demo programs]
+--- applet.html [Launches the JTwain Web Demo applet]
+--- LICENSE-*.txt [License agreement]
+--- Purchase.htm [Click to order JTwain]

1.6 Development Environment Setup

After you have installed JTwain, you need to setup your development environment in order to develop Java applications with JTwain. You need:

- 1) Put JTwain.jar into your class path.
- 2) Put AspriseJTwain.dll into your system path, e.g., C:\Windows\System32.

If you only want to see JTwain demos, then you do not have to perform this step. For more information, please refer the 'Software Distribution' section.

1.7 Compatibility

Operating Systems: All Windows platforms are currently supported; other OSs planned.

Java Runtime: Version 1.2 or above.

2 Image Acquisition with JTwain

2.1 For the Impatient

The following code demonstrates the basic usage of JTwain:

```
1. try {
2.     Source source =
3.         SourceManager.instance().getDefaultSource();
4.     source.open();
5.     Image image = source.acquireImage();
6.     ... // Uses image here ...
7. }catch(Exception e) {
8.     e.printStackTrace();
9. }finally{
10.     SourceManager.closeSourceManager();
11. }
```

Line 2: *SourceManager* represents TWAIN source manager, which manages and controls all the data(image) sources, e.g. scanners, digital cameras, available on the operating system. There can be one and only one *SourceManager* at any time. A default Source is obtained by calling *SourceManager's getDefaultSource* method.

Line 3: Opens the Source.

Line 4: Acquires an Image from the opened Source. Now, the Image has been acquired, and it can be used as other Images in your applications.

Line 9 closes any open SourceManager (which closes any open Source) regardless whether there are exceptions thrown.

To execute the above lines, you have to import:

com.asprise.util.jtwain.SourceManager and *com.asprise.util.jtwain.Source*.

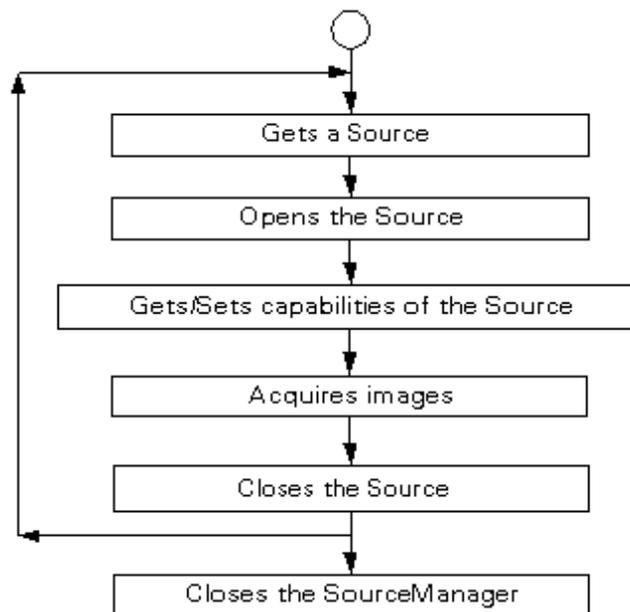
For a complete sample application, please refer to *DemoSimple.java* (in demo-src.jar).

Note: You need at least one source to run the demo programs. If you do not have any sources, download and install the sample TWAIN Source provided by twain.org at:

<http://www.twain.org/devfiles/twainkit.exe>

TIP: Source code for all demo programs are in the file demo-src.jar.

3 Control Flow of a Typical Image Acquisition Process



The figure above shows the control flow of a typical image acquisition process. In last section, basic code for image acquisition has been presented. You will notice that this flow clearly illustrates that code, except that:

- ◆ Getting and setting capabilities are not used – we will introduce this in later sections.
- ◆ In stead of explicitly closing the opened Source, SourceManager is closed – which causes any opened Source close.

3.1 Getting a Source

Source, or data source, is an abstraction of an image source – which can be a scanner, a digital camera or an image database.

There are many ways to get a Source from the SourceManager:

3.1.1 Gets the default Source

```
1. Source source = SourceManager.instance().getDefaultSource();
```

SourceManager will return the default Source or null if no data source exists.

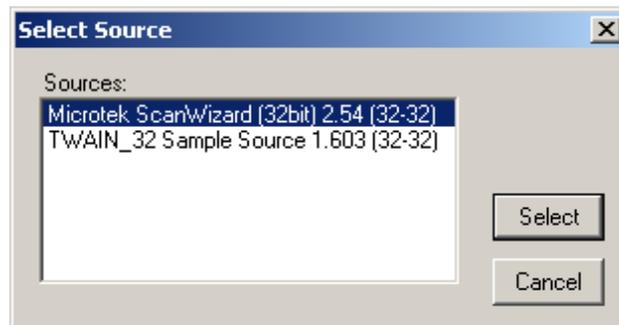
If you do not have any TWAIN compatible scanner or digital camera, you can try the sample TWAIN Source provided by twain.org at:

<http://www.twain.org/devfiles/twainkit.exe>

3.1.2 Lets the user select a Source

```
1. Source source = SourceManager.instance().selectSourceUI();
```

A dialog will pop up to enable the user to select a Source. Run `LaunchDemo.bat` and select `DemoSelectSourceDialog` to see the demo. You will see the following pop up (the contents may be different):



3.1.3 Selects source by its name

```
1. Source source = SourceManager.instance().selectSource("Big  
Camera");
```

A Source will be returned if there is a Source with the specified name, *null* otherwise.

3.1.4 Get all the Sources available

```
1. Source sources[] = SourceManager.getAllSources();
```

An array containing all the data sources will be returned.

3.1.5 Validating a Source:

```
1. Source validateSource = SourceManager.selectSource(source);
```

If the source is a valid Source, a new Source object representing the same data Source is returned possible with more detailed information; otherwise, null will be returned.

3.2 Hiding the User Interface

3.2.1 Hiding the “Select Source” UI:

Instead of using method (3.1.2) in Getting a Source, try to use other methods, like (3.1.1), (3.1.3), (3.1.4).

3.2.2 Hiding the scanner/digital camera's acquisition

UI:

Before performing image acquisition, a Source should be called on the following method:

```
1. if(source.isUIEnabled())
2.     source.setUIEnabled(false);
```

3.2.3 Hiding the indicators' UI:

Some Sources may pop up indicator dialogs to show the progress of image acquisition. To turn them off:

```
1. source.setIndicators(false);
```

The above line set the source's capability CAP_INDICATORS to off. To learn more about capabilities, see next section.

Run DemoHiddenUI to see hidden-UI image acquisition from the default Source in action.

3.3 Setting and Getting Source Capabilities

One of TWAIN's benefits is it allows applications to easily interact with a variety of acquisition devices. Developers of applications need to be aware of a Source's capabilities and may influence the capabilities that the Source offers to the application's users. To do this, the application can perform capability negotiation. Some sample capabilities are: CAP_XFERCOUNT – indicating number of images the application is willing to accept, ICAP_XRESOLUTION - X-axis resolution of the Source.

When all the UIs have been hidden, setting and getting the Source's capabilities are extremely important. Setting and getting source capabilities should be done after the source has been opened and before performing the image acquisition. Thanks to JTWAIN, you do not have to go into dirty details of those capabilities. We have implemented proper operations for every capability that required by TWAIN 1.9 specification.

In last section, CAP_INDICATORS has been set by calling method setIndicators. Here, we use CAP_XFERCOUNT as an example to explain setting and getting capabilities in details:

Method in Source	Remarks
getTransferCount	Return the Capability (CAP_XFERCOUNT)'s valid value(s) including current and default values. [Corresponding to TWAIN: MSG_GET]
getCurrentTransferCount	Get the Capability's current value. [Corresponding to TWAIN: MSG_GETCURRENT]
getDefaultTransferCount	Get the Capability's preferred default value (Source specific). Corresponding to TWAIN: MSG_GETDEFAULT]
resetTransferCount	Change a Capability's current value to its TWAIN-defined default. [Corresponding to TWAIN: MSG_RESET]

Method in Source	Remarks
setTransferCount	Change a Capability's current and/or available value(s). [Corresponding to TWAIN: MSG_SET]

For a complete list of all capabilities, please refer the TWAIN specification at:
http://www.twain.org/docs/Spec1_9_197.pdf

Demo program DemoGetCapabilities prints all the capabilities that the selected Source supports.

4 Acquiring Images

Acquire an image using JTwain is as easy as:

```
1. java.awt.Image image = source.acquireImage();
```

To load the image completely:

```
1. MediaTracker tracker = new MediaTracker(this);
2. tracker.addImage(this.image, 1);
3.
4. try {
5.     tracker.waitForAll();
6. }catch(Exception e) {
7.     e.printStackTrace();
8. }
```

Then you can proceed to acquire another image. For a complete sample application, see `DemoSelectSourceDialog – DemoSelectSourceDialog.java` and `ImageDisplayer.java`.

New Features:

In addition to the `acquireImage` method, there is another one named `acquireImageAsBufferedImage`. The `acquireImageAsBufferedImage` method acquires an image into a `BufferedImage`.

4.1 Automatic Document Feeding (ADF)

JTwain makes it easy to perform automatic document feeding. The following code illustrates an automatic document feeding process:

```
1. source.open();
2. source.setUIEnabled(true);
3.
4. int counter = 1;
5. do {
6.     Image image = source.acquireImage();
```

```
7.     // Uses image here ...
8. }while(source.hasMoreImages());
```

If the data Source supports ADF, the user can enable ADF and set number of documents intending to acquire. Run DemoADF to see how ADF acquisition works.

To perform ADF without UI, replace the above code Line 2 with the following code:

```
1. source.setUIEnabled(false);
2. source.setFeederEnabled(true);
3. source.setAutoFeed(true);
4. source.setTransferCount(3);
```

Modify DemoADF.java to run ADF without UI.

Note: unfortunately, many scanners does not support ADF without UI. Please perform thorough test if you want to use ADF in UI disabled mode.

4.2 Saving Acquired Images into Files

4.2.1 Built-in Image Saving Functions

From version 8.2, you can using the following built-in image saving functions:

```
public InputStream outputLastAcquiredImageAsJPEG()
```

```
public File saveLastAcquiredImageIntoTemporaryFile()
```

```
public void saveLastAcquiredImageIntoFile(String destination)
```

```
public void saveLastAcquiredImageIntoFile(File destination)
```

4.2.2 Sample Code

```
1. public class DemoSaveJPEG extends JTwainDemoCode{
2.
3.     public DemoSaveJPEG() {
4.
```

```
5.     try {
6.
7.     Source source = SourceManager.instance().getDefaultSource();
8.
9.     if(source == null) {
10.        error("There is no (default) source on the system!");
11.        return;
12.    }
13.
14.    source.open();
15.
16.    Image image = source.acquireImage();
17.
18.    ImageDisplayer imageDisplayer =
19.        new ImageDisplayer("DemoSimple", image);
20.
21.    FileDialog fileDialog = new
22.        FileDialog(imageDisplayer.getFrame(), "Save the image acquired into
23.        a file: ", FileDialog.SAVE);
24.    fileDialog.show();
25.
26.    source.saveLastAcquiredImageIntoFile(new
27.        File(fileDialog.getDirectory(), fileDialog.getFile()));
28.
29.    source.close();
30.
31.    }catch(Exception e) {
32.        exception(e);
33.    }finally{
34.        SourceManager.closeSourceManager();
35.    }
36. }
37.
38. public static void main(String[] args) {
39.     new DemoSaveJPEG();
40. }
```

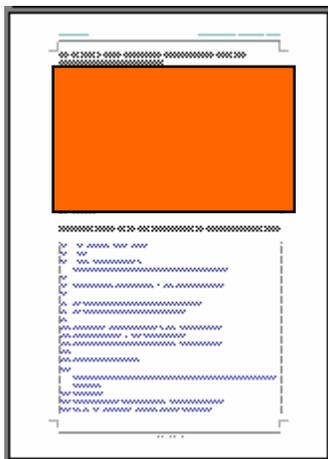
The code above can run on any JRE with version 1.2 and above. Additionally, if you are using Java version 1.4 or about, you can use the *ImageIO* class to write the acquired images into various formats. For example,

```
1. import java.awt.*;
2. import java.awt.image.*;
```

```
3. import java.io.*;
4. import javax.imageio.*;
5.
6. import com.asprise.util.jtwain.*;
7.
8. ...
9.
10. try {
11.
12.     Source source = SourceManager.instance().getDefaultSource();
13.     source.open();
14.
15.     BufferedImage image = source.acquireImageAsBufferedImage(); //
    Acquire the image
16.
17.     // Save the image as a PNG file
18.     ImageIO.write(image, "png", new File("C:\\\\test.png"));
19.
20.     // Save it as a JPEG file
21.     ImageIO.write(image, "jpg", new File("C:\\\\test.jpg"));
22.
23. }catch(Exception e) {
24.     e.printStackTrace();
25. }finally{
26.     SourceManager.closeSourceManager();
27. }
```

4.3 Acquiring a Specified Region Only

In some cases, you want to acquire a region/part of the page only:



You use the *setRegion* method to tell the scanning device to scan the specified region only:

```
1. source.setRegion(0, 0, source.getPhysicalWidth()/2,
   source.getPhysicalHeight()/2);
```

If your scanner does not support this feature, you have to scan the whole page and crop it after you acquire it.

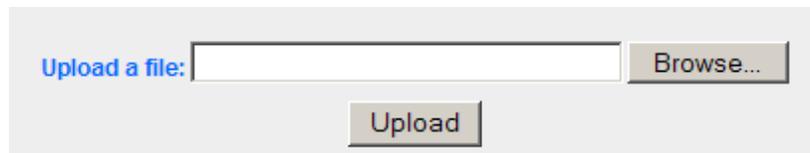
4.4 Uploading Acquired Images to Web Servers

Warning: this is a feature of JTwain Web Applet, you must purchase a proper license for it before you can start using it. You can either purchase JTwain Web Applet license or JTwain Site Developer license.

For the demo of JTwain acquisition and uploading in action, please visit:

<http://asprise.com/product/jtwain/applet/>

A typical uploading web page is shown below:



The user browses a file and clicks the upload button to upload the selected file. The HTML code behind it is:

```
1. <form method='post' enctype='multipart/form-data'
2.   action='/product/jtwain/applet/fileupload.php?method=upload'>
3.
4.   <input type='file' name='file[]' style='width: 300'>
5.   <input type='hidden' name='testParam' value='testValue'>
6.   <input type='submit' value='Upload'></td>
7.
8. </form>
```

With JTwain, you can use the code below to perform the exact same uploading action:

```
1. // Acquire image first
2. ...
```

```
3. File acquiredImage =
   source.saveLastAcquiredImageIntoTemporaryFile();
4.
5. FileUploader fileUploader = new FileUploader();
6.
7. // fileUploader.setProxyHost(proxyHost);
8. // fileUploader.setProxyPort(port);
9.
10. Properties extraParameters = new Properties();
11. extraParameters = new Properties();
12. extraParameters.put("testParam", "testValue");
13.
14. fileUploader.upload(
15.     "http://asprise.com/product/jtwain/applet/fileupload.php?method=
        upload",
16.     "file[]",
17.     "scanned.jpg", acquiredImage, extraParameters);
18. // If no exception thrown, upload succeeds.
```

4.5 About the JTwain Web Applet

On the first tab "JTwain Configuration", you can check your JTwain dll version or install the JTwain dll file to your system.

If the JTwain version is fine, you can proceed to the second tab to acquired images;

After an image has been acquired, you can use the third tab to upload it to your web server.

Specify your extra parameters if any in the text field next to "Extra Params" in the following format:

param1=value1;param2=value2;param3=value3

5 Low level API Programming

5.1 JTwain API Model

JTwain has provided all TWAIN specification v1.9 mandatory capabilities negotiation operations. Besides them, some useful optional capabilities negotiation operations have been implemented too. If you need to deal with some less frequently used capabilities or you prefer to have total control on capabilities negotiation, the good news that JTwain exposes low level APIs which you can use to accomplish your tasks.

5.2 Extending the Source

Suppose you need to do ICAP_BITDEPTHREDUCTION capability negotiation. (ICAP_BITDEPTHREDUCTION is the capability controlling Reduction Method the Source should use to reduce the bit depth of the data.)

On page 9-425 of Twain specification v1.9:

Values

Type: TW_UINT16
Default Value: No Default
Allowed Values: TWBR_THRESHOLD 0
TWBR_HALFTONES 1
TWBR_CUSTHALFTONE 2
TWBR_DIFFUSION 3
Container for MSG_GET: TW_ENUMERATION
TW_ONEVALUE
Container for MSG_SET: TW_ENUMERATION
TW_ONEVALUE

Now, you can extend the Source class to add this capability handling.

```
1. public class MySource extends com.asprise.util.jtwain.Source
2. {
3.     public MySource(Source source)
4.     {
```

```
5.     if(source == null ||
6.         (source.getSourceName() == null && source.getIdentity ==
7.         null))
8.         throw new IllegalArgumentException("source should not be
9.         null!");
10.
11.     this.sourceName = source.getSourceName();
12.     this.identity = source.getIdentity();
13. }
14.
15. // Get value(s).
16. public int[] getBitDepthReduction()
17.     throws InvalidStateException, OperationException
18. {
19.     Object ret = getCapability(MSG_GET, ICAP_BITDEPTHREDUCTION,
20.                               TWON_DONOTCARE16);
21.     return getIntArray(ret);
22. }
23.
24. // Set one value.
25. public void setBitDepthReduction(int value)
26.     throws InvalidStateException, OperationException
27. {
28.     ValueContainerOneValue c = new ValueContainerOneValue();
29.     ItemTypeInteger item = new ItemTypeInteger(TWTY_UINT16, value);
30.     c.setItem(item);
31.     setCapability(ICAP_BITDEPTHREDUCTION, c);
32. }
33.
34. // Set multiple values.
35. public void setBitDepthReduction(int[] values)
36.     throws InvalidStateException, OperationException
37. {
38.     if(values == null || values.length == 1)
39.         throw new IllegalArgumentException("Empty values!");
40.
41.     ValueContainerEnumeration c = new ValueContainerEnumeration();
42.     for(int i=0; i<values.length; i++)
43.         c.pushItem(new ItemTypeInteger(TWTY_UINT16, values[i]));
44.     setCapability(ICAP_BITDEPTHREDUCTION, c);
45. }
46. // Sample use.
```

```

47. public static void main(String args[]) throws Exception
48. {
49.     Source source = SourceManager.instance().selectSourceUI();
50.     source.open();
51.
52.     MySource mine = new MySource(source);
53.     mine.setBitDepthReduction(1);
54.     Image image = mine.acquireImage();
55.
56.     ...
57. }

```

Line 17 *getCapability* returns the capability inquiry result. Some examples: if the container is of the type *ValueContainerOneValue* and item type is *TWTY_UINT16*, then a *Long* will be returned; For a single value container, the returned object may be of the following types: Long, Double, String in case of multi-value containers, the returned object may be of the following type: Long[], Double[], String[].

5.3 TWAIN & JTwain Mapping

5.3.1 Containers

TWAIN	JTWAIN
TW_ONEVALUE	ValueContainerOneValue
TW_ARRAY	ValueContainerArray
TW_ENUMERATION	ValueContainerEnumeration
TW_RANGE	ValueContainerRange

5.3.2 Item Types

TWAIN	JTWAIN
TW_BOOL	ItemTypeInteger(TWTY_BOOL, value[1 or 0])
TW_INT8	ItemTypeInteger(TWTY_INT8, value)
TW_INT16	ItemTypeInteger(TWTY_INT16, value)
TW_UINT32	ItemTypeInteger(TWTY_UINT32, value)

TW_FIX32	ItemTypeFix32(value)
TW_STR32	ItemTypeString(TWTY_STR32, value)
TW_STR1024	ItemTypeString(TWTY_STR1024, value)
TW_FRAME	ItemTypeFrame

6 Advanced Topics

6.1 Exception Handling

According to TWAIN specification, a data Source does not have to support all the capabilities. Further more, there are some poor designed TWAIN Sources which are not TWAIN-compatible. Exceptions could be thrown everywhere especially in the middle of getting/setting capabilities.

Asprise JTwain package has been carefully designed to relieve developers from handling complicated multiple exceptions. By default, a minimum set of exceptions will be thrown. To enable a Source to throw all possible exceptions, one can enable or disable this minimum exception option by executing the following code:

```
1. source.setMinimumExceptionEnabled(true); // Only a minimum set of
   exceptions will be thrown. By default, minimum exception is enabled.
2. source.setMinimumExceptionEnabled(false); // Throw as many as
   possible exceptions.
```

The “minimum set of exceptions” does not include exceptions thrown during capability getting/setting. Most of critical exceptions are included in this minimum exception set.

Note that although minimum exception option can be turned on, the developer still has to write code to catch exceptions – there could be some exceptions to be thrown, although just a minimum set.

A typical exception handling example:

```
1. try {
2.     Source source = SourceManager.instance().getDefaultSource();
3.     source.open();
4.     source.setXResolution(999); // Invalid value!
5.
6.     Image image = source.acquireImage();
7.
8.     new ImageDisplayer("DemoSimple", image);
```

```
9.  
10. }catch(Exception e) {  
11.     e.printStackTrace();  
12. }finally{  
13.     SourceManager.closeSourceManager();  
14. }
```

Note that line 4 set the capability ICAP_XRESOLUTION to value 999 – very likely an invalid value (typical resolution values are 72, 300, 600, etc). Even if the Source does not support this capability or it rejects the value setting request, no exception will be thrown – because minimum exception option is on by default. The image will be acquired using Source's default resolution.

If this capability is very important, one can turn off minimum exception option by inserting the following line before line 4:

```
1. source.setMinimumExceptionEnabled(false)
```

Browse any of the demo programs to see exception handling at work.

If the user cancels scanning during single image acquisition, Source's acquireImage will throw a JTwainException exception.

6.2 Using JTwain in Threads

The TWAIN is not thread-safe, its Java counterpart, JTwain inherits the same property. Once a Source has been opened, it has to be used in the same thread until it is closed. If a Source is used in application's main thread, a fatal error in native method will be generated if AWT or Swing event handling thread tries to access it.

To avoid fatal errors, always call SourceManager.instance().close() at the end of image acquisition.

6.3 Software Packaging and Distribution

So you have successfully developed your Java applications with JTwain. It's time to distribute your programs to end users. First, make sure you are an authorized licensee registered with LAB Asprise!. To purchase a license, please visit:

<http://www.asprise.com/product/jtwain>

There are two files about JTwain you need to distribute along with your own binary code. One is JTwain.jar, which is like any other java library, you can just copy it and put it in the class path. The other one is AspriseJTwain.dll, the native library. There are many ways to 'install' this dll file, you can:

Add the folder containing the native library to the system path, or

Copy the native library to jre/bin directory – 'install' the library to the JVM, or

Copy the native library to a specific location, e.g. C:\AspriseJTwain.dll, before calling

SourceManage.instance(), call:

```
SourceManage.setLibraryPath(" C:\AspriseJTwain.dll ");
```

7 Deployment Guide

Jar files are the recommended java binary code distribution format. This section will guide you to put *.class files into jar file.

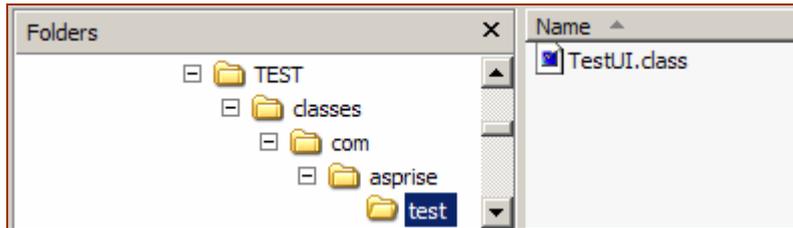
7.1 Organizing your directory

It is important to organize your project directory properly first. Let's say you have the following Java class:

```
2. package com.asprise.test;
3. ...
4. public class TestUI extends JApplet {
5.
6.     public TestUI() {
7.         getContentPane().setBackground(Color.pink);
8.         getContentPane().setLayout(new BorderLayout());
9.         JLabel label = new JLabel("Testing Message.");
10.        label.setHorizontalAlignment(JLabel.CENTER);
11.        getContentPane().add(label, BorderLayout.CENTER);
12.    ...
13. }
14.
15.
16. public static void main(String[] args) {
17.     JFrame frame = new JFrame("TestUI");
18.     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.     frame.getContentPane().add(new TestUI());
20.     frame.setSize(400, 300);
21.     frame.setVisible(true);
22. }
23. }
```

The class name is *TestUI* and it is in package ***com.asprise.test***.

For this class you should organize the directory as following:



[Assuming *TEST* is your root folder. The absolute path of *TEST* is *E:\eclipse-301\workspace\TEST*]

7.2 Creating a Jar File

Now, you change directory to the *classes* folder, and type the following command to put all the files in the *classes* directory into a jar file named *program.jar*.¹

```
E:\eclipse-301\workspace\TEST\classes>jar cvf program.jar *
```

Now, all the files in the *classes* directory has been put into:
E:\eclipse-301\workspace\TEST\classes\program.jar

Now, move the *program.jar* to the *E:\eclipse-301\workspace\TEST* folder.

```
E:\eclipse-301\workspace\TEST\classes>move program.jar ..
```

Note: If your class is in the default package (i.e., you did not specify any package information in the source code), you should put it in the *classes* folder. Otherwise, you should create the corresponding package directory under the *classes* folder and put the class file into the package directory as shown above.

7.3 Creating Signed Applets

Only signed applets can be granted with all the permissions. To enable an applet to access dll files (e.g., JTwain) or other native services, you need to sign the all the jar files required by the applet with the same certificate. This section provides a step-by-step guide.

First, if you have *.class* files, make sure you put them into a jar as shown in the first

¹ In this document, we use Microsoft Windows as the hosting OS. You can easily convert the commands onto other platforms, like Mac OSX, Linux, etc.

chapter.

In our sample project, we have a jar named *program.jar* containing our program binary class files. We also have a library file named *JTwain.jar*.

Name	Size	Type
classes		File Folder
src		File Folder
.classpath	1 KB	CLASSPATH File
.project	1 KB	PROJECT File
program.jar	2 KB	Executable Jar File
JTwain.jar	66 KB	Executable Jar File

7.3.1 Creating a Certificate

Before you can sign jar files, you need a certificate. If you already have one, you can skip this procedure.

First, change directory to the root directory of the project:

```
E:\eclipse-301\workspace\TEST
```

Run the following command:

```
E:\eclipse-301\workspace\TEST>keytool -genkey -dname "cn=YOUR NAME,  
ou=ORG UNIT, o=COMPANY, c=US" -alias test -keypass testpass -validity 999  
-keystore test -storepass testpass
```

A file named *test* containing the certificate is generated under the *TEST* folder.

7.3.2 Signing Jar Files

Use the following command to sign each jar file:

```
E:\eclipse-301\workspace\TEST>jarsigner -keystore test -storepass  
testpass -keypass testpass program.jar test  
  
E:\eclipse-301\workspace\TEST>jarsigner -keystore test -storepass  
testpass -keypass testpass JTwain.jar test  
  
...
```

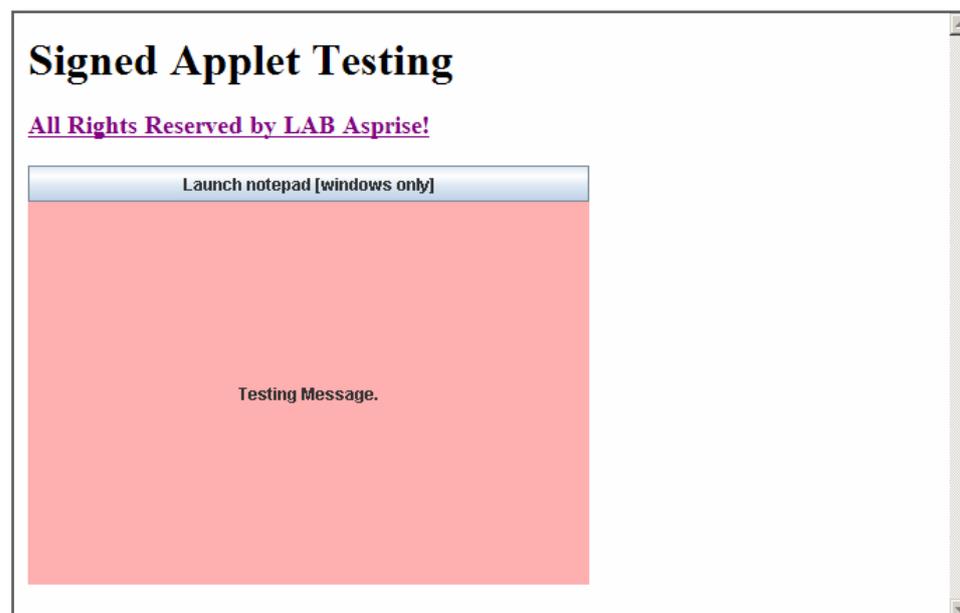
Now, all the jar files have been signed. You can launch the applet with a proper HTML page.

7.3.3 Launching the Applet

Now, you can use HTML code like the following to invoke the applet:

```
1. <html>
2. <head>
3. <title>TestUI</title>
4. </head>
5. <body>
6. <h1>Signed Applet Testing</h1>
7. <h3><a href="http://asprise.com">All Rights Reserved by LAB
   Asprise!</a></h3>
8.
9. <applet code="com.asprise.test.TestUI.class" codebase="."
   archive="program.jar, JTwain.jar" width="400" height="300">
10. Oops, Your browser does not support Java applet!
11. </applet>
12.
13. </body>
14. </html>
```

The screenshot:

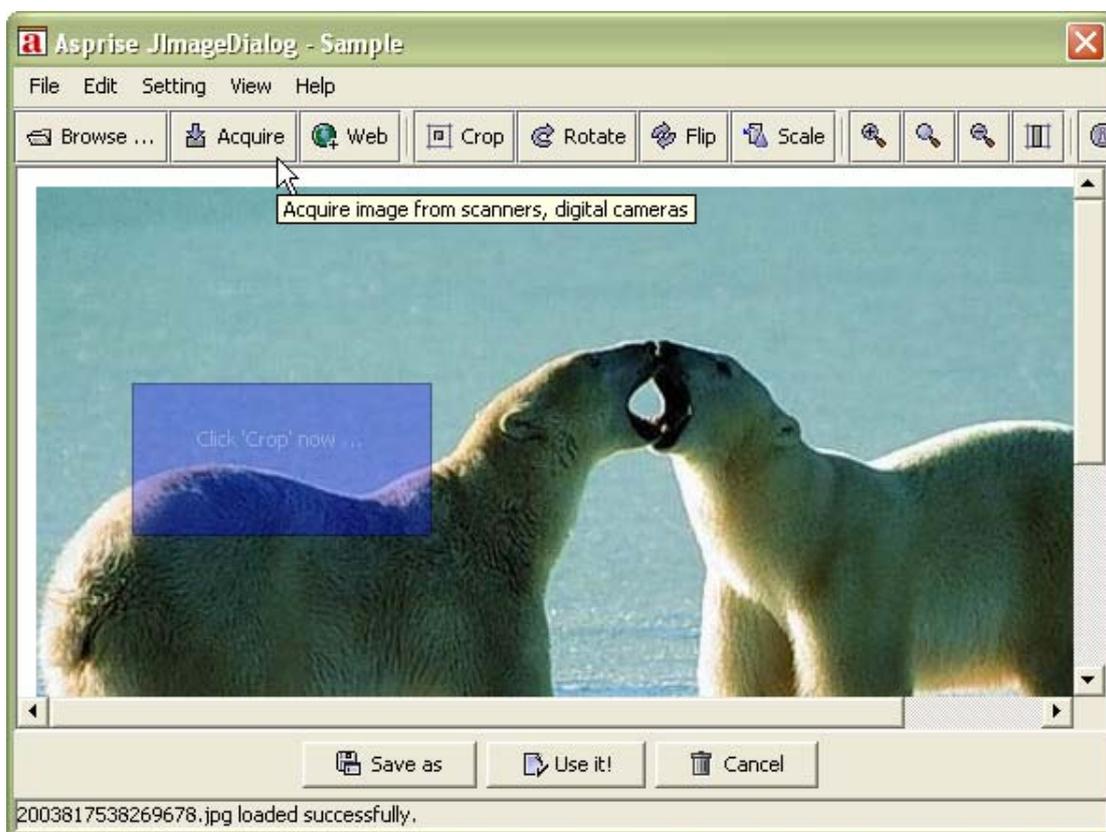


8 Image Acquisition Components

The image acquisition UI components are not part of Asprise OCR library. However, based on our customers' experience, if you need to build a front-end for OCR, they are invaluable and could save you a lot of time. Otherwise, you may skip this chapter.

8.1 JImageDialog

JImageDialog is an image acquisition UI component that allows the user to load images and to perform basic image editing tasks. If you are developing some applications that require the user to select/edit/input images, then JImageDialog will make your life extremely easy – and more importantly, the user experience will be improved dramatically.



Let say you want to build an album application, the user is required to supply photos

(i.e. images). You put a button on your panel. When the user click the button, JImageDialog is brought up – now the user can select existing pictures files from his or her computer or acquire images from digital cameras or scanners. And the user can edit images before putting it into the album.

8.1.1 Advantages

- ◆ Multiple image sources supported: local computer, digital cameras, scanners and the web;
- ◆ Multiple image formats: read and write BMP, PNG, JPG, GIF, PCT, PSD and many other formats;
- ◆ Platform/Virtual machine independent: Any platform, any Java virtual machine (version 1.3 or above);
- ◆ Powerful features: rotation, flipping, scaling, clipping, etc.
- ◆ User friendly as well as developer friendly

The user can load images from local computer or the web, he or she can also acquire images from digital cameras and scanners. After the image has been loaded, the user can rotate, clip, flip, and scale the image. The image has been loaded and edited, the user can save the image or select the image - which will be used in your applications.

8.1.2 Sample Uses

8.1.2.1 Modal (synchronous) mode

```
15. JImageDialog dialog = new JImageDialog(frame, "Sample", true); //  
    Modal dialog  
16. BufferedImage image = dialog.showDialog();  
17. ...
```

Line 1 constructs the image dialog.

Line 2 brings up the image dialog and waiting for user's selection/acquisition.

Besides using JImageDialog in synchronous mode, you can also use it in:

8.1.2.2 Asynchronous mode

```
18. public class JImageDialogSample extends JPanel implements
    JImageDialogListener {
19.     ...
20.     BufferedImage image;
21.
22.     // Displays selected image if any.
23.     public void paintComponent(Graphics g) {
24.         super.paintComponent(g); // Paint background.
25.         if(image != null)
26.             g.drawImage(image, 0, 0, null);
27.     }
28.
29.     // Sets image and refreshes the panel.
30.     public void setImage(BufferedImage image) {
31.         this.image = image;
32.         setPreferredSize(getPreferredSize());
33.         revalidate();
34.         repaint();
35.     }
36.
37.     // Methods in JImageDialogListener
38.     // When the user presses cancel button, this method will be called.
39.     public void onCancel() {
40.         setImage(null);
41.     }
42.
43.     // When the user presses the selection button, will be invoked.
44.     public void onImageSet(BufferedImage image) {
45.         setImage(image);
46.     }
47. }
48.
49. ...
50. JImageDialogSample imagePanel = new JImageDialogSample();
51.
52. JImageDialog dialog = new JImageDialog();
53. dialog.addImageDialogListener(imagePanel);
54. dialog.showDialog();
```

Line 1-30 implements a *JImageDialogListener*.

Line 33 constructs the listener.

Line 35 constructs the dialog.
 Line 36 registers the listener the dialog
 Line 37 brings up the dialog

When the user acquires an image and selects it, JimageDialog's listeners will be notified. In this case, *imagePanel.onImageSet(BufferedImage image)* will be called and thus the panel will display the selected image. If the user cancels the selection, *onCancel()* will be called instead.

Sample application: *com.asprise.util.ui.JImageDialogSample*

8.1.3 Supported Image Formats

The following table shows image formats supported by JImageDialog:

Formats	File extensions	READ	WRITE
Adobe Photoshop	*.psd	Y	Y
Bitmap, Windows/OS2	*.bmp, *.dib	Y	Y
Cursor	*.cur	Y	
Graphics Interchange Format	*.gif	Y	
Icon	*.ico	Y	
JPEG	*.jpg, *.jpeg	Y	Y
Macintosh PICT Format	*.pict, *.pct	Y	Y
PCX Format	*.pcx	Y	Y
Portable Network Graphics	*.png	Y	Y
Sun Raster Format	*.ras	Y	
Tag Image File Format	*.tif, *.tiff	Y	
Targa	*.tga	Y	Y
X Bitmap	*.xbm	Y	Y
X PixMap	*.xpm	Y	Y

On any Java platforms (version 1.3 or above), JImageDialog supports the above formats (using its own library to read/write image files). JImageDialog intelligently selects the best way to read or write files – e.g. on Java 1.4, it may invoke *ImageIO* to see whether a file can be read or written; if the *ImageIO* can do the job then JImageDialog will let it do; otherwise, JImageDialog will use its own library to access the file.

Note: You can only read/write image files from the JImageDialog UI component with unlicensed image acquisition UI component package. If you want to access image files

from your Java code and/or to perform other advanced operations, you need to obtain an affordable license from LAB Asprise!.

8.1.4 Compatibility

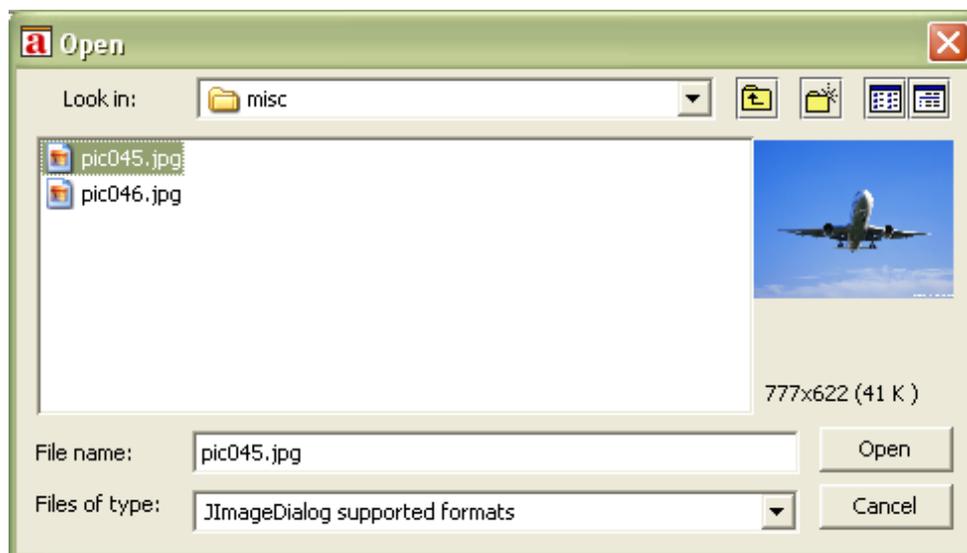
All Java runtimes with version 1.3 or above.

8.1.5 Software Packaging and Distribution

Mandatory: `jid.jar`, `JTwain.jar`

8.2 JImageFileChooser

JImageFileChooser is an extended *JFileChooser* that supports image preview and image information extraction.



When the user clicks an image file, its preview and associated information will be displayed to assist the user to select the proper image.

8.2.1 Sample Use

```
55. JFileChooser fc = new JImageFileChooser(lastDirectory);
56. fc.addChoosableFileFilter(JImageFileChooser.getImageFileFilter()
    );
57. int returnVal = fc.showOpenDialog(frame);
58. ...
```

Line 1 creates the image file chooser;

Line 2 set the file filter.

You can use it as normal JFileChooser, and it improves the user experience greatly.

8.2.2 Supported Image Formats

Please refer to Supported Image Formats in JImageDialog section.

Note: You can only preview image files from the JImageFileChooser UI component with unlicensed image acquisition UI component package. If you want to read/write image files from your Java code with the package and/or to perform other advanced operations, you need to obtain an affordable license from LAB Asprise!.

8.2.3 Compatibility

All operating systems;

All Java runtimes with version 1.2 or above.

8.2.4 Software Packaging and Distribution

Mandatory: **jid.jar**

9 Support and professional services

9.1 Support Web Site

<http://www.asprise.com/product/jtwain>

9.2 Basic Support

Our team provides basic support for general Asprise JTwain developers. Email your technical questions to support@asprise.com

Advice: You are strongly recommended to subscribe our premium support service in order to get your problems solved quickly.

9.3 Premium Support Services

Free fixed period of premium support services subscription comes with every license purchased. You may optionally extend premium support services after your subscription expires. For more details, visit the order page.

9.4 Professional Services

Our team are ready to help you to develop various applications, components. Please send your query to info@asprise.com