

Ziath



ZIATH DATAPAQ REMOTE CONTROL USER MANUAL[®]



Version 1.28



Ziath Ltd.

Tel +44 (0)1223 655577 • Fax +44 (0)1223 790069

www.ziath.com • sales@ziath.com • support@ziath.com

DATAPAQ REMOTE CONTROL USER MANUAL © Copyright 2008, 2009, 2010, 2011 Ziath Ltd.

All Rights Reserved. **Ziath** and **DataPaq** are registered trademarks of Ziath Ltd. No part of this publication, in either its printed or electronic format, may be copied, reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose involving resale for profit or gain, through any form of paid, membership or subscription service, without the express permission of Ziath Ltd.

Revised 21/06/2011

THIS IS VERSION 1.28 OF THE DATAPAQ REMOTE CONTROL USER MANUAL.

DataPaq Remote Control

Introduction

The headless enhancement to **DataPaq** allows for the interaction of a separate computer program to control the scanner and retrieve the results. This is enabled by a separate program entitled *Server.exe* which is provided. This will interact with **DataPaq** and control the scanner without a Graphical User Interface showing on the screen.

There are two modes in which the server can be started up: (i) command line mode, and (ii) server mode.

Activating Demonstration Mode

DataPaq can operate in a demonstration mode; in this mode it will load a prescanned image to decode. This has the advantage that an integrator does not need a **DataPaq** scanner attached to their computer when performing integration work. To activate this mode, take the following steps:

- Install **DataPaq**, start the application (not in headless mode) and select *Trial mode* when prompted (You may enter a license key but the license key will be locked to your development machine – should you require a developer's license, please contact support at support@ziath.com).
- Install a scanner configuration and configure it (there is a dummy scanner available for this reason if there are no suitable scanners attached).
- Setup a 96 well portrait plate (don't forget the unique ID).
- Close **DataPaq**.
- An image of the plate is required. If there is a rack available then place a plate onto the scanner and take an image (using the windows document and scanner wizard or other software such as IrfanView <http://www.irfanview.com>) of the entire available space on the scanner (not just the rack) – save this image in a format which is lossless such as *png* or *bmp*. Copy this image to the install directory of **DataPaq** (default is *C:\Program Files\Ziath\DataPaq*). If no rack or scanner is available then please contact support at support@ziath.com with the information of the make and model you wish to scan.
- Open the configuration file at *C:\Documents and Settings\\.ziath\datapaq\settings.xml* (note that the *.ziath* directory starts with a period and you will need to activate the *Show Hidden Files and Folders* option in File Explorer).

- At the top of the file is the plate group you have just configured with an empty attribute called *demo* - add into this the name (just the filename, not the path) of the image file you just copied into the **DataPaq** install directory
- Restart **DataPaq** and read this plate group; it should show *Loading Demo Image* and then calibrate the image. Click again and it should show *Loading Demo Image* and then decode the images.

This plate group will now execute with no scanner attached. This will operate in normal mode or in headless mode.

Command Line Mode

Command line mode allows the user to simply execute a command line at the command prompt: the software will start up, execute the scanner and return the results formatted according to the supplied commands in the command line. To start the server in command line mode you simply need to specify the following options:

| Short Form | Long Form | Params | Default | Description |
|------------|--------------|------------------------|---------|--|
| c | calibrate | none | None | Instructs the scanner to calibrate the specified plate group. The plate group id must be supplied for this command. Note that if this is not specified, the scanner will run a scan operation. |
| e | exportformat | excel text xml | Text | Sets the format of the export, set to either <i>excel</i> or <i>text</i> . |
| f | exportfile | filename | Stdout | Specifies the file to write to. If this is not specified the results will be written to the standard out of the process (typically to the console). |
| b | rackbarcodes | comma separated string | Unknown | A comma separated list of the rack barcodes (note – should the rack barcode scanner be installed, this option will be ignored). |
| g | scannergroup | group uid | None | The id of the scanner group to scan or calibrate. |
| v | verbose | none | None | If specified DataPaq will post comments on progress to the screen. |

Therefore, due to the defaults, the simplest command line operation of **DataPaq** is as follows:

```
Server -g 1234
```

This will run the scanner to scan in command line mode, returning the results in text format to the command line. A more sophisticated example follows:

```
Server -g 1234 -f results.xls -e excel -b CODE1, CODE2, CODE3
```

This will execute the scanner, using the codes CODE1 to CODE3 as the rack barcodes. The results will be returned into an excel file which is called *results.xls*.

When the process exits, it returns an exit code. The number returned gives an indication of the success of the command:

Exit code 0 – the process executed and returned cleanly, no errors occurred

Exit code 1 – the command line options could not be specified

Exit code 2 – the specified port for server mode (see below) could not be opened

Exit code 3 – the plate group was not specified

Exit code 4 – the execution of the scan or calibrate failed

Exit code 5 – the result file could not be written

Server Mode

The server mode starts **DataPaq** and listens on a specified port for incoming commands. Multiple clients can attach to **DataPaq**; however, only one operation at a time can be performed. If starting in server mode, you need to specify the command line option *s* (server for longform). **DataPaq** will start and listen to a specified port for commands (by default the server will listen on port 8888) – however, this can be changed with the following command:

| Short Form | Long Form | Params | Default | Description |
|------------|-----------|-------------|---------|--|
| p | port | port number | 8888 | The port that the server will listen on. |

Once the server is running in socket mode, the process will continue to listen to the socket until it is either closed or receives the command SHUTDOWN. An example of a command line to start the server is below:

```
Server -s -p 8899
```

This will start the server running in socket mode which will listen for incoming connections on port 8899.

The server responds to a set of commands: all commands must be terminated by a carriage return and a line feed; all responses will be terminated with a carriage return and a line feed.

Commands

- **Version**

Command: VERSION\r\n

Response: <version code>\r\nOK\r\n

When the server receives the command, it returns the version number of **DataPaq** that it is connected to.

Example

Received by Server: VERSION\r\n

Response from Server: 1.1\r\nOK\r\n

- **Get UIDs**

Command: GET_UIDS\r\n

Response: 2|Standard Single Plate|96 Well Plate\r\n
1|Standard Single Plate|48 Well Plate\r\n
<uid>|<Scanner Name>|<Group Name>\r\n
OK\r\n

When the server receives this command, it will return all the UIDs in the system. The return will be the UID, followed by the plate group name, separated by a pipe (|). Each line is terminated by a carriage return and a line feed and signifies a new plate group; the final line is OK followed by a line feed and carriage return.

Example

Received by Server: GET_UIDS\r\n

Response from Server:

```
1|Single Plate Standard Focus |96 Well Plate\r\n2|Single Plate Standard Focus |48 Well Plate\r\n5|Single Deep Standard Focus|96 Well Plate\r\nOK\r\n
```

- **Status**

Command: STATUS\r\n

Response: <status>\r\nOK\r\n

When the server receives the command, it returns the status of **DataPaq**. These can be as follows:

- IDLE – **DataPaq** is ready to accept a command

- BUSY – **DataPaq** is currently executing a command
- ERROR – **DataPaq** encountered an error; it will keep this status until a command is successfully executed

Example

Received by Server: STATUS\r\n

Response from Server: IDLE\r\nOK\r\n

• Scan

Command: SCAN <uid of group to scan> <export format> <comma separated list of rack barcodes>\r\n

Params

- **uid** – this is the unique id of the scanner group. This can either be manually set in the **DataPaq** config file or can be specified using the **DataPaq** GUI.
- **export format** – this is the return format for the read: it can either be *excel* or *text*
- **barcodes** – this is a comma-separated list of rack barcodes; for example: *CODE1,CODE2,CODE3*. Note that should the rack 1D barcode scanner option be installed then this option will be ignored and the results of the barcode scanner will be used instead.

Response: OK\r\n [the scanner scans and then] <result of scan>\r\nOK\r\n

When the server receives this command, it scans the deck and returns the results to the caller via the connected socket. The scanner will either return a string, if text as the export format is supplied, or it will return a stream of bytes which represent a binary excel file. Should the scan fail, the system will return ERR\r\n with a description of the error. Note that the format of the text returned can be changed upon request; should you require this to be done then please contact Ziath at queries@ziath.com.

Example

Received by Server: SCAN 1234 text 1,2,3\r\n

Response from Server: OK\r\n

[The scanner does its scan]

ScanID,Date,RackBarcode,Row,Col,tubeBarcode

```
1,13-Nov-2008 22:06:18,CODE1,A,1,1013587786
1,13-Nov-2008 22:06:18,CODE1,A,2, 1013586701
1,13-Nov-2008 22:06:18,CODE1,A,3,1013587788
...
...
1,13-Nov-2008 22:06:18,CODE3,H,10,1013588736
1,13-Nov-2008 22:06:18,CODE3,H,11,1013588735
1,13-Nov-2008 22:06:18,CODE3,H,12,1013588208
OK\r\n
```

- **Calibrate**

Command: CALIBRATE <plate group id to calibrate>

Params

- **uid** – the plate group to calibrate

Response: OK\r\n [the scanner calibrates and then] OK\r\n

Upon receiving this command, the scanner will calibrate the specified plate group with the racks on the deck of the scanner. It will initially return OK, perform the scan and then return OK when the scan is finished. Should the calibration fail, the scan will return ERR\r\n with a description of the error message.

Example

Received by Server: CALIBRATE 1234\r\n

Response from Server: OK\r\n [The scanner does its calibration] OK\r\n

- **Last Image**

Command: LAST_IMAGE <The Scanned Position, starting at 0>\r\n

Response: <the image - as a multi-line base 64 encoded string>,
<A blank line>, OK\r\n

Upon receiving this command, the server will retrieve the last image scanned for the current position, annotate the image, encode the image as a *png*, further encode the image into a base64 string and send it to the client. Once the entire image has been sent, **DataPaq** will

then send an empty line, followed by `OK\r\n`. Should there be no image available (because no scan was run after **DataPaq** started up), **DataPaq** will return `ERR12`.

Example

Received by Server: `LAST_IMAGE 0\r\n`

Response from Server: `<A Number of base64 encoded lines containing the image>`

`<A blank line>`

`OK\r\n`

- **Save Last Image**

Command: `SAVE_LAST_IMAGE <The Scanned Position, starting at 0>
<Location to save file>\r\n`

Response: `OK\r\n`

Upon receiving this command, the server will retrieve the last image scanned for the current position, annotate the image, encode the image as a *png*, and save it to the location specified in the second parameter. Should the filename contain spaces, wrap the filename in double quotes to escape the spaces. Once the entire image has been sent, **DataPaq** will then send an empty line, followed by `OK\r\n`. Should there be no image available (because no scan was run after **DataPaq** started up), **DataPaq** will return `ERR12`. Should the incorrect parameters be returned **DataPaq** will return `ERR16`, `ERR17` is returned if the image cannot be saved (unusually due to either a bad file name or permission problems with the file location)

Example

Received by Server: `SAVE_ LAST_IMAGE 0 "c:\Users\benn\datapaq
image.png"\r\n`

Response from Server:

`OK\r\n`

- **Last Raw Image**

Command: `LAST_RAW_IMAGE <The Scanned Position, starting at 0>\r\n`

Response: <the image - as a multi-line base 64 encoded string>, <A blank line>, OK\r\n

Upon receiving this command, the server will retrieve the last image scanned for the current position, encode the image as a *png*, further encode the image into a base64 string and send it to the client. Note that this image is the raw image from the imaging device and is therefore quite large. Once the entire image has been sent, **DataPaq** will then send an empty line, followed by OK\r\n. Should there be no image available (because no scan was run after **DataPaq** started up), **DataPaq** will return ERR12.

Example

Received by Server: LAST_IMAGE 0\r\n

Response from Server: <A Number of base64 encoded lines containing the image>

<A blank line>

OK\r\n

- **Save Last Raw Image**

Command: SAVE_LAST_RAW_IMAGE <The Scanned Position, starting at 0> <Location to save file>\r\n

Response: OK\r\n

Upon receiving this command, the server will retrieve the last image scanned for the current position, encode the image as a *png*, and save it to the location specified in the second parameter. Should the filename contain spaces, wrap the filename in double quotes to escape the spaces. Note that this is the raw image from the imaging device and is therefore rather large. Once the entire image has been sent, **DataPaq** will then send an empty line, followed by OK\r\n. Should there be no image available (because no scan was run after **DataPaq** started up), **DataPaq** will return ERR12. Should the incorrect parameters be returned DataPaq will return ERR16, ERR17 is returned if the image cannot be saved (unusually due to either a bad file name or permission problems with the file location)

Example

Received by Server: SAVE_ LAST_IMAGE 0 "c:\Users\benn\datapaq image.png"\r\n

Response from Server:

OK\r\n

- **Close**

Command: CLOSE\r\n

Response: OK\r\n

Upon receiving this command, the scanner will close the client which requested the close. To cleanly disconnect your client, it is recommended to call this method. Note that this will not affect any other clients attached to **DataPaq** at the time.

Example

Received by Server: CLOSE\r\n

Response from Server: OK\r\n [The server disconnects the client]

- **Shutdown**

Command: SHUTDOWN\r\n

Response: OK\r\n

Upon receiving this command, the scanner will shutdown and the server process will exit, but only if the scanner is idle. Should the scanner fail to shut down, ERR\r\n will be written, followed by a description of the error message; however should this happen the Server will *still* shutdown (except in the case of the server being busy).

Example

Received by Server: SHUTDOWN\r\n

Response from Server: OK\r\n [The scanner shuts down] OK\r\n

- **Force Shutdown**

Command: FORCE_SHUTDOWN\r\n

Response: OK\r\n

Upon receiving this command, the scanner will shutdown and the server process will exit, even if the scanner is running – use this command with caution. Should the scanner fail to

shut down, ERR\r\n will be written, followed by a description of the error message; however should this happen the Server will *still* shutdown.

Example

Received by Server: SHUTDOWN\r\n

Response from Server: OK\r\n [The scanner shuts down] OK\r\n

- **Get Barcode Scanner Com Port**

Command: GET_BARCODE_SCANNER_COM_PORT\r\n

Response: COM5\r\nOK\r\n

When receiving this command DataPaq report the com port used by the 1D barcode rack scanner. Should the port not be set then ERR21 will be returned.

Example

Received by Server: GET_BARCODE_SCANNER_COM_PORT\r\n

Response from Server: COM5\r\nOK\r\n

- **Get All Com Port IDs**

Command: GET_ALL_COM_PORT_IDS\r\n

Response: COM1, COM3, COM4, COM5\r\nOK\r\n

When receiving this command DataPaq reports the com ports on the host computer with each port name separated by a comma.

Example

Received by Server: GET_ALL_COM_PORT_IDS\r\n

Response from Server: COM1, COM3, COM4, COM5\r\nOK\r\n

- **Set Barcode Scanner Com Port**

Command: SET_BARCODE_SCANNER_COM_PORT <The com port to connect to>\r\n

Response: OK\r\n

When receiving this command DataPaq disconnects the current 1D barcode scanner com port if it is connected. DataPaq then attempts to connect to the specified com port. If the specified com port cannot be found by DataPaq, ERR18 is returned. If for some reason DataPaq cannot connect to the com port then ERR19 is returned.

Example

Received by Server: SET_BARCODE_SCANNER_COM_PORT COM1\r\n

Response from Server: [DataPaq disconnects from the current com port and reconnects to the specified com port]OK\r\n

- **Enable Barcode Scanner**

Command: ENABLE_BARCODE_SCANNER\r\n

Response: OK\r\n

When receiving this command DataPaq enables the 1D rack barcode scanner.

Example

Received by Server: ENABLE_BARCODE_SCANNER \r\n

Response from Server: OK\r\n

- **Disable Barcode Scanner**

Command: DISABLE_BARCODE_SCANNER\r\n

Response: OK\r\n

When receiving this command DataPaq disables the 1D rack barcode scanner.

Example

Received by Server: DISABLE_BARCODE_SCANNER \r\n

Response from Server: OK\r\n

- **Get Barcode Scanner Enabled**

Command: GET_BARCODE_SCANNER_ENABLED\r\n

Response: true\r\n

When receiving this command DataPaq returns the state of the barcode scanner enabled/disabled flag. If enabled true is returned, otherwise false is returned.

Example

Received by Server: GET_BARCODE_SCANNER_ENABLED\r\n

Response from Server: true\r\nOK\r\n

- **Scan Linear Barcode**

Command: SCAN_LINEAR_BARCODE\r\n

Response: OK\r\n

Upon receiving this command, the 1D rack scanner will attempt to scan a 1D rack barcode using the ZTS-1DR attached reader. If no barcode could be read ERR13 will be returned, ERR14 is returned when the barcode scanner cannot be communicated to and ERR15 when the barcode scanner is not enabled.

Example

Received by Server: SCAN_LINEAR_BARCODE\r\n

Response from Server: B1268FR\r\n

Error Codes

Should the server encounter an error, it will return the error plus a description in the following format:

```
<ERROR CODE>\r\n
```

```
<ERROR_DESCRIPTION> [:<ERROR_MESSAGE>]\r\n
```

In some cases there will not be a specific error message but the description will always be present and two lines will always be returned. An example of an error return is below:

```
ERR1\r\n
```

```
The unique ID and the export method must be supplied on a  
scan\r\n
```

In the case of an error with a description an example is below:

```
ERR8\r\n
```

```
Failed to scan: Cannot communicate to scanner\r\n
```

| Error Code | Error Description |
|-------------------|--|
| ERR1 | The unique ID and the export method must be supplied on a scan |
| ERR2 | The export methods can only be text or excel |
| ERR3 | The unique ID must be supplied on a calibrate |
| ERR4 | Failed to cleanly shut down |
| ERR5 | Failed to calibrate |
| ERR6 | Unknown Command |
| ERR7 | Server busy |
| ERR8 | Failed to scan |
| ERR9 | Please provide image number to return |
| ERR10 | Image number is not numerical |
| ERR11 | Error writing image to client |
| ERR12 | No image available to return |
| ERR13 | Linear barcode not read |
| ERR14 | Cannot communicate to linear barcode scanner |
| ERR15 | Linear barcode scanner not enabled |
| ERR16 | To save an image provide the scan position and save location |
| ERR17 | Save image failed |



[This Page Intentionally Left Blank]

Appendix A

The following is some example code written in Java that illustrates how to connect to and use the headless **DataPaq** server.

```
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import sun.misc.BASE64Decoder;

/**
 * An example class showing connection to DataPaq via a socket. Note that this
 * class writes a lot of information to standard out (System.out) as it is
 * example code. It is recommended that a logging framework be used to replace
 * these System.out calls.
 *
 * @author Neil Benn
 * @version 1.2
 */
public class DataPaqRemote {

    private BufferedReader in = null;
    private OutputStream out = null;
    private String host;
    private Integer port;

    private static final Integer DATAPAQ_SERVER_PORT = 8888;
    private static final String DATAPAQ_SERVER_HOST = "localhost";

    /**
     * Initialises and connects to the DataPaq scanner
     */
}
```

```
* @throws UnknownHostException
* @throws IOException
*/
public DataPaqRemote(String host, Integer port)
    throws UnknownHostException, IOException {
    // create the socket, incoming and outgoing streams
    this.host = host;
    this.port = port;
    Socket clientSocket = new Socket(this.host, this.port);
    out = clientSocket.getOutputStream();
    in = new BufferedReader(new InputStreamReader(clientSocket
        .getInputStream()));
    // this is returned on each connection
    System.out.println(in.readLine());
}

/**
 * Returns the version number of the running DataPaq server.
 *
 * @return
 * @throws IOException
 */
public String getVersion() throws IOException {
    // get the version number
    out.write("VERSION\r\n".getBytes());
    out.flush();
    String version = in.readLine();
    System.out.println("VERSION " + version);
    System.out.println("VERSION ACKNOWLEDGEMENT " + in.readLine());
    return version;
}

/**
 * Returns the status of the scanner
 * @return the scanner status
 * @throws IOException if the scanner cannot be communicated with
 */
public String getStatus() throws IOException{
    out.write("STATUS\r\n".getBytes());
    out.flush();
    String status = in.readLine();
    System.out.println("STATUS " + status);
    System.out.println("STATUS ACKNOWLEDGEMENT " + in.readLine());
    return status;
}
```

```

/**
 * Runs a scan in text mode and returns the results of the scan in a string.
 *
 * @param uid
 *         the unique id of the plate to scan as configured in the
 *         DataPaq application
 * @return a string representing the scan result
 * @throws IOException
 *         if there is a problem writing to the socket out or reading
 *         from socket in
 */
public String runScan(String uid) throws IOException {
    // perform a scan
    out.write(("SCAN " + uid + " TEXT\r\n").getBytes());
    out.flush();
    System.out.println("SCAN RECEIVED " + in.readLine());

    // read in all the results, appending to the StringBuffer
    StringBuffer scanBuffer = new StringBuffer();
    Boolean scanFirstLine = true;
    while (true) {
        String line = in.readLine();
        if (scanFirstLine) {
            if (line.startsWith("ERR")) {
                throw new RuntimeException("Scanner reported error - "
                    + line + in.readLine());
            }
            scanFirstLine = false;
        }
        if (line.equals("OK")) {
            break;
        }
        scanBuffer.append(line).append("\r\n");
    }
    return scanBuffer.toString();
}

/**
 * Returns the last scanned image from the DataPaq server.
 *
 * @param scanPos
 *         the position on the scanner to retrieve, starts counting at
 *         zero
 * @return a buffered image representing the scanned image

```

```

* @throws IOException
*         if there is a problem writing to the socket out or reading
*         from socket in
*/
public BufferedImage getLastScanImage(Integer scanPos) throws IOException {
    // get the image of the last scan
    System.out.println("Getting last scanned image");
    out.write("LAST_IMAGE 0\r\n".getBytes());
    out.flush();

    System.out.println("reading image in");
    // reading image in
    StringBuffer buffer = new StringBuffer();
    Boolean imageFirstLine = true;
    while (true) {
        String line = in.readLine();
        buffer.append(line).append("\r\n");
        // the first line may be an error so we check for that
        // and then set first line to false as the rest of the
        // lines will not be in error
        if (imageFirstLine) {
            imageFirstLine = false;
            if (line.startsWith("ERR")) {
                System.out.println("Error " + line);
                throw new RuntimeException("failed to read image " +
                    line + " " + in.readLine());
            }
        } else {
            if (line.length() == 0) {
                break;
            }
        }
    }

    // we have the image we now need to convert the encoded string to bytes
    // and then write the image to a file
    byte[] decoded = new BASE64Decoder().decodeBuffer(buffer.toString());
    BufferedImage bi = ImageIO.read(new ByteArrayInputStream(decoded));
    return bi;
}

/**
 * Disconnected this client from the DataPaq server. The server remains
 * operational after this call.

```

```

*
* @throws IOException
*         if there is a problem writing to the socket out or reading
*         from socket in
*/
public void close() throws IOException {
    out.write("CLOSE\r\n".getBytes());
    out.flush();
    System.out.println(in.readLine());
}

/**
 * Shuts down the DataPaq server if it is not busy. Note that this will
 * disconnect all other connected clients.
 *
 * @throws IOException
 *         if there is a problem writing to the socket out or reading
 *         from socket in
 */
public void shutdown() throws IOException {
    out.write("SHUTDOWN\r\n".getBytes());
    out.flush();
    System.out.println(in.readLine());
}

/**
 * Tests execution of the DataPaq server via a socket. Before executing this
 * ensure that the DataPaq server is running (the port of 8888 is the
 * default port which will be used), this is assumes that the test is
 * running on the same machine as the DataPaq server. It also assumes the
 * server is already running.
 *
 *
 * @throws UnknownHostException
 * @throws IOException
 * @throws InterruptedException
 */
public static void main(String args[]) throws UnknownHostException,
        IOException {
    DataPaqRemote dpr = null;
    try {

        System.out.println("Connecting");
        dpr = new DataPaqRemote(DataPaqRemote.DATAPAQ_SERVER_HOST,
            DataPaqRemote.DATAPAQ_SERVER_PORT);
        System.out.println(dpr.getVersion());
        System.out.println(dpr.getStatus());
        System.out.println(dpr.runScan("1"));
    }
}

```

```
        Image i = dpr.getLastScanImage(0);
        i = i.getScaledInstance(i.getWidth(null)/5,
                               i.getHeight(null)/5,
                               Image.SCALE_SMOOTH);

        showImageInFrame(i);
        System.out.println("success");
    } finally {
        System.out.println("closing");
        if (dpr != null){
            dpr.close();
        }
    }
}

/**
 * Shows the image specified in the parameter in a frame.
 *
 * @param bi the image to show
 */
private static void showImageInFrame(Image bi){
    JFrame frame = new JFrame("Scanned Image");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel l = new JLabel();
    l.setIcon(new ImageIcon(bi));
    frame.getContentPane().add(l);
    frame.pack();
    frame.setResizable(false);
    frame.setLocation(
        (int)(Toolkit.getDefaultToolkit().getScreenSize().getWidth() -
            frame.getWidth())/2,
        (int)(Toolkit.getDefaultToolkit().getScreenSize().getHeight() -
            frame.getHeight())/2);
    frame.setVisible(true);
}
}
```