PRACTICAL INNOVATION

DataPaq™ Remote Control

User Guide

DataPaq™ Version 3.12
Manual Version rev.1

DataPaq™ Remote Control User Guide

All Rights Reserved. **Ziath™** and **DataPaq™** are 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.

DataPaq™ User Guide © Copyright 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017 Ziath Ltd.

This is version 3.12 of the DataPaq Remote Control User Guide.

Revised 17/10/2017

DataPaq™ Remote Control

Introduction

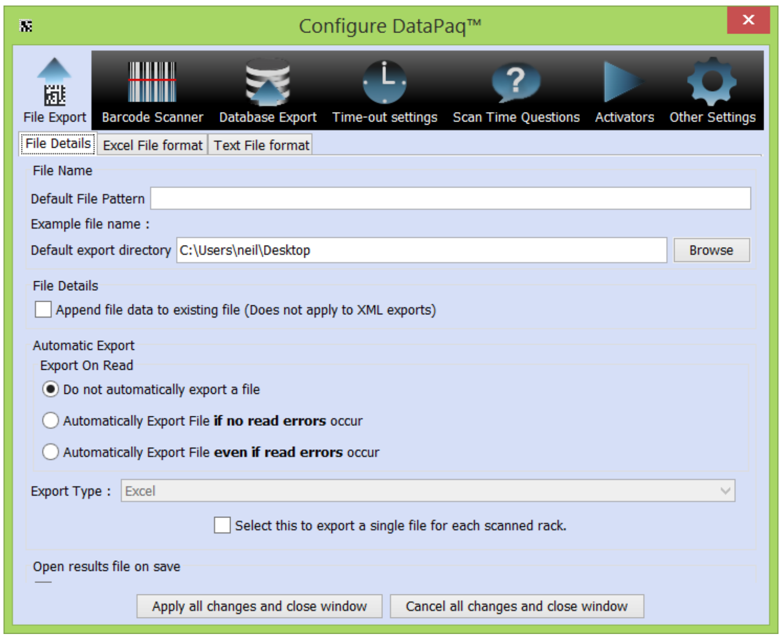
The headless enhancement to **DataPaq** allows for the interaction of a separate computer program to control the scanner and retrieve the results. This is enabled by a separate program entitled *Server.exe* which is provided. This will interact with **DataPaq** and control the scanner without a Graphical 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).
- Open the options window and expand the width to the right to reveal the Dev Settings option:



File Name
Default File Pattern
Example file name :
Default export directory C:\Users\neil\Desktop

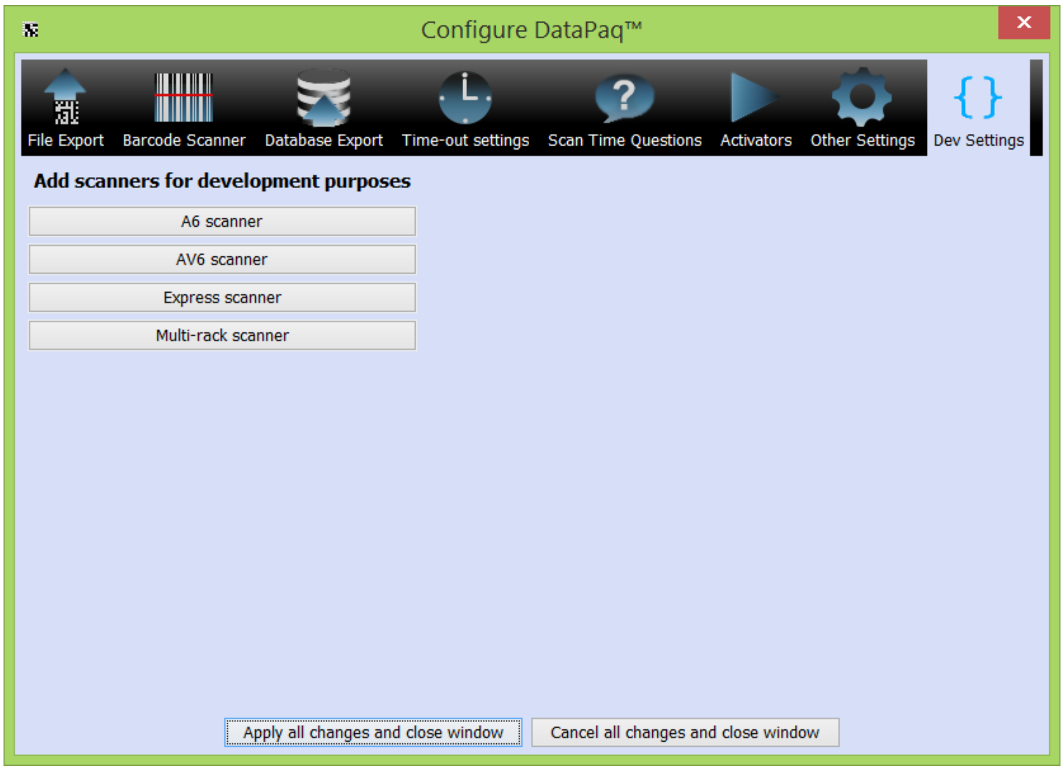
File Details
☐ Append file data to existing file (Does not apply to XML exports)

Automatic Export
Export On Read
● Do not automatically export a file
○ Automatically Export File **if no read errors** occur
○ Automatically Export File **even if read errors** occur

Export Type : Excel

☐ Select this to export a single file for each scanned rack.

Open results file on save



- Add in a demo scanner; for this example we will use the AV6 scanner
- 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 a greyscale 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 a defined location. 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:\%PROGRAMDATA%\Ziath\DataPaq\settings.xml* (note that this is the ProgamData folder and not the Program Files folder and you will need to activate the *Show Hidden Files and Folders* option in File Explorer). If on windows XP this folder will be different and depend upon the language setting of your computer.
- At the top of the file is the plate group you have just configured with an empty attribute called *demo* - add into this the full filename and 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 decode the image.

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

.NET Library

Ziath provides a .NET library for integration use; contact support@ziath.com for a reference.

Java & Python Sample Code

Ziath provide sample code in Java and Python; the Java code is at the end of the manual and contact Ziath for the Python code.

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 *xml*, *json*, *excel* or *text.*
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.
j	Saveimage	filename	None	If specified along with a scan this will save the raw image in png format to the specified filename
k	Scanimage	filename	None	This will scan and save a raw image to

				the entered filename. It will not attempt to decode the image it will just save it
l	Linear	None	none	This will scan the linear barcode on the rack and return. The scan result will be returned on stdout

For all filename you can use the following placeholders:

- #date# - inserts the date in iso format
- #time# - inserts the time in iso format
- #barcode# - inserts the rack barcode in iso format
- #uid# – inserts the unique id
- #plategroup# - inserts the name of the group represented by the uid

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	commandPort	port number	8888	The port that the server will listen on.
N	notificationPort	port number	8686	The port that the notification server will transmit messages 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. Note that there is a maximum number of simultaneous client connections; the amount can be discovered by calling GET_MAX_CONNECTIONS. If this number is exceeded a string describing the current connections is returned and the socket closed fomr the serer side. 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.

Should you need to close the server down and for some reason the server is not responding then if you connect to port 6699 then the server will immediately close down and close the connection. This is designed to be an emergency shutdown and should not typically be used to close the server.

In addition; if the Ziath Linear reader is attached the presence of a plate can be automatically detected and this be notified and/or a scan automatically triggered. This notification is done on port 8686; this can be changed using the –n parameter if required.

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 Max Connections***

  Command: `GET_MAX_CONNECTIONS\r\n`

  When the server receives the command, it returns the maximum number of simultaneous client connections allowed

  Response: `<number of maximum allowed connections>\r\nOK\r\n`

  Example

  Received by Server: `GET_MAX_CONNECTIONS\r\n`

  Response from Server: `20\r\nOK\r\n`

- ***Get Thread Report***

  Command: `GET_THREAD_REPORT\r\n`

  When called this returns details on the threads on the server; note that client threads are labelled Server with the IP address of the client and the local port number the server is listening on.

  Response: `<the thread report>r\nOK\r\n`

  Example

  Received by Server: `GET_THREAD_REPORT\r\n`

  Response from Server:

```
Group[system:class java.lang.ThreadGroup]
  Thread[Reference Handler:class java.lang.ref.Reference$ReferenceHandler]
  Thread[Finalizer:class java.lang.ref.Finalizer$FinalizerThread]
  Thread[Signal Dispatcher:class java.lang.Thread]
  Thread[Attach Listener:class java.lang.Thread]
  Thread[Java2D Disposer:class java.lang.Thread]
  Thread[Swing-Shell:class java.lang.Thread]
  Group[main:class java.lang.ThreadGroup]
    Thread[main:class java.lang.Thread]
    Thread[AWT-Windows:class java.lang.Thread]

    Thread[Thread-2:class
com.ziath.datapaq.scannerconnection.ScanThreadImpl]

    Thread[Thread-3:class org.apache.log4j.xml.XMLWatchdog]

    Thread[Emergency      shutdown      socket      server:class
java.lang.Thread]

    Thread[Server      -      /0:0:0:0:0:0:0:1:58576:58576:class
java.lang.Thread]

    Thread[Emergency      shutdown      socket      server:class
java.lang.Thread]\r\nOK\r\n
```

- ***Get Memory Usage***

  Command: `GET_MEMORY_USAGE\r\n`

  When the server receives the command, it returns the current memory usage split into Allocated, Free, Max and Total Free

  Response: `<memory usage report>\r\nOK\r\n`

  Example

  Received by Server: `GET_MEMORY_USAGE\r\n`

  Response from Server: `FREE_MEMORY:12111|ALLOCATED_MEMORY:15872|MAX_MEMORY:253440|TOTAL_FREE_MEMORY:249679\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\n

2|Single Plate Standard Focus |48 Well Plate\r\n

5|Single Deep Standard Focus|96 Well Plate\r\n

OK\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 Image***

Command: `SCAN_IMAGE <uid>\r\n`

Response: `OK\r\n(Scanner scans rack)<base 64 encoded images\r\n\r\nOK\r\n`

When the server receives this comamdn it will scan with the specified uid but will not attempt to decode the image; instead it will return the image to the caller as a base 64 encoded stream.

Example

Received by Server: `SCAN_IMAGE1\r\n`

Response from Server: `OK\r\n(Scanner scans rack)<base 64 encoded images\r\n\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 be *excel, xml, json* 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
```

- ***Last Image***

Command: `LAST_IMAGE <The Scanned Position, starting at 0> <the scale factor (optional)> <format (optional)>\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 the requested format, further encode the image into a base64 string and send it to the client, if the scale factor is supplied the image will be scaled otherwise it will be scaled to 1. The format can be png, jpeg or bmp; if not supplied a png image will be returned. Once the entire image has been sent, **DataPaq** will then send an empty line, followed by `OK\r\n`.

Errors:

ERR10 – Image number cannot be parsed, typically because it is not an integer

ERR12 - No image available (because no scan was run after **DataPaq** started up)

ERR16 – not enough parameters supplied, at least the image number must be supplied

ERR22 – Scale factor cannot be parsed, typically because it is not an Integer or a double

ERR25 – Image format was not one of png, jpeg or bmp

Example:

Received by Server: `LAST_IMAGE 0 0.5 jpeg\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> <the scale factor (optional)> <format (optional)>\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 saved, **DataPaq** will then send `OK\r\n`. IF no scale factor is entered then the image will not be scaled and if no format is requested the image will be a png file. Available formats are png, jpeg and bmp.

Errors:

ERR10 – Image number cannot be parsed, typically because it is not an integer

ERR12 - No image available (because no scan was run after **DataPaq** started up)

ERR17 – Image cannot be saved *(unusually due to either a bad file name or permission problems with the file location)*

ERR22 – Scale factor cannot be parsed, typically because it is not an Integer or a double

ERR25 – Image format was not one of png, jpeg or bmp

Example:

Received by Server: `SAVE_ LAST_IMAGE 0 "c:\Users\benn\datapaq image.jpg" 0.2 jpeg\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`.

  Errors:

  ERR10 – Image number cannot be parsed, typically because it is not an integer

  ERR12 - No image available (because no scan was run after **DataPaq** started up)

  ERR22 – Scale factor cannot be parsed, typically because it is not an Integer or a double

  ERR25 – Image format was not one of png, jpeg or bmp

  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 <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 `OK\r\n`.

Errors:

ERR12 - No image available (because no scan was run after **DataPaq** started up)

ERR16 – not enough parameters supplied

ERR17 – Image cannot be saved *(unusually due to either a bad file name or permission problems with the file location)*

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 Licence Details***

  Command: `GET_LICENCE_DETAILS\r\n`

  Response: `OK\r\n`

  When receiving this command DataPaq returns the licence details it returns the licence details in the format of <LICENCE OWNER>|<LICENCE COMPANY>|<LICENCE KEY>|<IS TRIAL – TRUE or FALSE>

  Example

  Received by Server: `GET_LICENCE_DETAILS\r\n`

  Response from Server: `NEIL BENN|ZIATH|ZHGYTI2|FALSE\r\n`

- ***Get Last Barcodes***

  Command: `GET_LAST_BARCODES\r\n`

  Response: {<barcode>|NONE,}\r\n`OK\r\n`

  When receiving this command DataPaq returns the last barcodes scanned or NONE if no barcodes have been scanned since the server was started.

  Example

  Received by Server: `GET_LICENCE_DETAILS\r\n`

  Response from Server: `NEIL BENN|ZIATH|ZHGYTI2|FALSE\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 Configured Linear Scanner***

  Command: `GET_CONFIGURED_LINEAR_SCANNER\r\n`

  Response: `MANUAL|OPTICAL|ZIATH_HARDWARE|NO_SCANNER|NONE\r\n`

  When receiving this command DataPaq returns the currently configured linear barcode scanner.

  Example

  Received by Server: `GET_CONFIGURED_LINEAR_SCANNER\r\n`

Response from Server: `MANUAL\r\nOK\r\n`

- ***Is Ziath Linear Scanner Connected***

  Command: `IS_ZIATH_LINEAR_SCANNER_CONNECTED\r\n`

  Response: `TRUE|FALSE\r\n`

  When receiving this command DataPaq will return true if the linear scanner is plugged in and false if it is not. If the linear scanner is not configured ERR35 will be returned.

  Example

  Received by Server: `GET_BARCODE_SCANNER_ENABLED\r\n`

  Response from Server: `TRUE\r\nOK\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: `<BARCODE>\r\nOK\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 (hardware linear scanner only) and ERR15 when neither the hardware or Ziath linear barcode scanner are enabled.

  Example

  Received by Server: `SCAN_LINEAR_BARCODE\r\n`

  Response from Server: `B1268FR\r\nOK\r\n`

- ***Get Last Detailed Exception***

  Command: `GET_LAST_DETAILED_EXCEPTION\r\n`

  Response: `OK\r\n`

  This command will return the last detailed scan exception that occurred. IF there was no detailed exception NONE is returned. For a list of detailed exception please see in the appendix.

  Example

  Received by Server: `CLEAR_LAST_DETAILED_EXCEPTION\r\n`

  Response from Server: `OK\r\n`

- ***Clear Last Detailed Exception***

  Command: `CLEAR_LAST_DETAILED_EXCEPTION\r\n`

  Response: `OK\r\n`

  This command will remove the last detailed exception on the server; note that this is the last detailed exception across all clients and not just this connected one. If there is no exception this will just return.

  Example

  Received by Server: `CLEAR_LAST_DETAILED_EXCEPTION\r\n`

  Response from Server: `OK\r\n`

- ***Decode Image***

  Command: `DECODE IMAGE <UID> <FORMAT> <MIME ENCODED RAW IMAGE> <RACK BARCODE(S)>\r\n`

  Response: `OK\r\n[IMAGE IS DECODED]<SCAN RESULTS>OK\r\n`

  Upon receiving this command, DataPaq will load the image from the MIME string and attempt to decode it with the specified UID. Note that both the DataPaq install that made the image and the server that is decoding the image need to be licenced.

  Example

  Received by Server: `DECODE IMAGE 1234 XML [MIME STRING] BAR1\r\n`

  Response from Server: `OK\r\n [WAIT] [SCAN RESULTS]\r\n`

- ***Enable Decode Updates***

  Command: `ENABLE_DECODE_UPDATES\r\n`

  Response: `OK\r\n`

  Upon receiving this command; the next time DataPaq decodes a rack a notification of each well decoded will be sent along with the number of wells to decode, note that also ticks up empty decoded wells. Note that the decoding is typically so quick that this will *slow down* decoding of the rack. These notifications appear between the OK received by a decode request and the scan results the notifications are in the format of 1 OF 96\r\n, 2 OF 96\r\n and so on.

  Example

  Received by Server: `ENABLE_DECODE_UPDATES\r\n`

  Response from Server: `OK\r\n`

- ***Disable Decode Updates***

  Command: `DISABLE_DECODE_UPDATES\r\n`

  Response: `OK\r\n`

  Upon receiving this command; the next time DataPaq decodes a rack no notifications of each well decoded will be sent.

  Example

  Received by Server: `DISABLE_DECODE_UPDATES\r\n`

  Response from Server: `OK\r\n`

- ***Is Decode Updates***

  Command: `IS_DECODE_UPDATES\r\n`

  Response: <TRUE|FALSE>`OK\r\n`

  If DataPaq is set to return decode updates when a tube is decoded this will return TRUE if not then FALSE will be returned.

  Example

  Received by Server: `IS_DECODE_UPDATES\r\n`

  Response from Server: FALSE\r\n`OK\r\n`

- ***Get Current Number of Connections***

  Command: `GET_CURRENT_NUMBER_OF_CONNECTIONS\r\n`

  Response: `<the number of currently connected clients>\r\nOK\r\n`

  Upon receiving this command; DataPaq will return the number of currently connected clients including this client.

  Example

  Received by Server: `GET_CURRENT_NUMBER_OF_CONNECTIONS \r\n`

  Response from Server: `2\r\nOK\r\n`

- ***Get Connected Scanners***

  Command: `GET_CONNECTED_SCANNERS\r\n`

  Response: `<as list of the connected scanners>\r\nOK\r\n`

  Upon receiving this command; DataPaq will return a list of connected scanners delimited by pipes. If no scanners are connected |NONE| will be returned. The possible scanner identifiers that could be returned are :

  DEEP_FOCUS – this is version 1 and version 2

  HIGH_SPEED – this is version 1 of the high speed scanner

  HIGH_SPEED_2 – this is version 2 of the high speed scanner

  MULTIRACK_8800 – this is version 1 of the multirack scanner

  MULTIRACK_9000 – this is version 2 of the multirack scanner

  PLUSTEC_A3 – this is a specialist extra large scanner

  CAMERA – This is the express scanner

  If for some reason this list cannot be returned error 33 is thrown

  Example

  Received by Server: `GET_CONNECTED_SCANNERS\r\n`

  Response from Server: `|CAMERA|DEEP_FOCUS|\r\nOK\r\n`

- ***Get Detailed Rack Group Info***

  Command: `GET_DETAILED_RACK_GROUP_INFO\r\n`

  Response:

```
UID=<uid>|NAME=<name>|NUMBER_OF_PLATES=<number of plates in rack group>|CALIBRATED =<true|false>|{NUMBER_OF_COLUMNS-<n>=<number of columns>|NUMBER_OF_ROWS-<n>=<number of rows>}\r\nOK\r\n
```

  Upon receiving this command; DataPaq will return the parameters of the rack group. If there is more than one plate calibrated the number of columns and number of rows will have -1,-2,-3 etc appended onto them.

  Example

  Received by Server: `GET_DETAILED_RACK_GROUP_INFO \r\n`

Response from Server:

```
UID=1|NAME=96 vial SBS rack|NUMBER_OF_PLATES=1|CALIBRATED =true
|NUMBER_OF_COLUMNS-1=12|NUMBER_OF_ROWS-1=8\r\nOK\r\n
```

- ***Get Connected Scanners***

  Command: `GET_CONNECTED_SCANNERS\r\n`

  Response:

```
|DEEP_FOCUS|HIGH_SPEED|HIGH_SPEED_2|MULTIRACK_8800|MULTIRACK_9000|PLUSTEC_A3|CAMERA|CUBE\r\n
```

  Upon receiving this command; DataPaq will all the scanners currently connected to the computer or |NONE| if none are connected. Note that CAMERA is used for the express scanner.

  Example

  Received by Server: `GET_CONNECTED_SCANNERS \r\n`

  Response from Server: `|CAMERA|\r\nOK\r\n`

- ***Is Scanner Connected for Uid***

  Command: `IS_SCANNER_CONNECTED_FOR_UID\r\n`

  Response: `TRUE\r\nOK\r\n`

Upon receiving this command; DataPaq will return true if the scanner required for scan the uid is connected and false if it is not. If no uid is given ERR27 will be returned and if the uid is not known then ERR26 is returned.

Example

Received by Server: `IS_SCANNER_CONNECTED_FOR_UID 96deep\r\n`

Response from Server: `TRUE\r\nOK\r\n`

- ***Is Orientation Barcode for Uid***

Command: `IS_ORIENTATION_BARCODE_FOR_UID\r\n`

Response: `TRUE\r\nOK\r\n`

Upon receiving this command; DataPaq will return true if the uid has an orientation barcode configured and false if it is not. If no uid is given ERR27 will be returned and if the uid is not known then ERR26 is returned.

Example

Received by Server: `IS_SCANNER_CONNECTED_FOR_UID 96deep\r\n`

Response from Server: `TRUE\r\nOK\r\n`

- ***Activate Plate Remote***

Command: `ACTIVATE_PLATE_REMOTE\r\n`

Response: <CURRENT BARCODE>\r\n`OK\r\n`

Upon receiving this command; DataPaq will turn on the activator and if a plate is placed on the scanner then the text PLATE_PRESENT <RACK BARCODE> will be fired on the notificiation port. If there is no Ziath Linear Reader enabled then ERR32 will be thrown.

Example

Received by Server: ACTIVATE_PLATE_REMOTE`\r\n`

Response from Server: `ZIA6986765\r\nOK\r\n`

- ***Deactivate Plate Remote***

Command: `DEACTIVATE_PLATE_REMOTE\r\n`

Response: `OK\r\n`

Upon receiving this command; DataPaq will turn off the activator and if a plate is placed on the scanner then nothing will happen.

Example

Received by Server: DEACTIVATE_PLATE_REMOTE`\r\n`

Response from Server: `OK\r\n`

- ***Is Activator Plate Remote***

  Command: IS_ `ACTIVATOR_PLATE_REMOTE\r\n`

  Response: <TRUE|FALSE>\r\n`OK\r\n`

  Upon receiving this command; DataPaq return true if the activator is set to record the prescence of a plate; false if not.

  Example

  Received by Server: IS_ACTIVATOR_PLATE_REMOTE`\r\n`

  Response from Server: `TRUE\r\nOK\r\n`

- ***Activate Scan Remote***

  Command: `ACTIVATE_SCAN_REMOTE\r\n`

  Response: `OK\r\n`

  Upon receiving this command; DataPaq will turn on the activator and if a plate is placed on the scanner then the rack will be scanned using the uid passed in by SET_ACTIVATOR_UID and the results returned the notificiation port according to the format set by SET_ACTIVATOR_RESULT_FORMAT. If there is no Ziath Linear Reader enabled then ERR32 will be thrown.

  Example

  Received by Server: ACTIVATE_PLATE_REMOTE`\r\n`

  Response from Server: `ZIA6986765\r\nOK\r\n`

- ***Deactivate Scan Remote***

  Command: `DECTIVATE_SCAN_REMOTE\r\n`

  Response: `OK\r\n`

Upon receiving this command; DataPaq will turn off the activator and if a plate is placed on the scanner then the rack will not be scanned.

Example

Received by Server: DEACTIVATE_PLATE_REMOTE`\r\n`

Response from Server: `OK\r\n`

- ***Is Activator Scan Remote***

  Command: IS_ `ACTIVATOR_SCAN_REMOTE\r\n`

  Response: <TRUE|FALSE>\r\n`OK\r\n`

  Upon receiving this command; DataPaq return true if the activator is set to scan when a new plate is detected by the linear scanner; false if not.

  Example

  Received by Server: IS_ACTIVATOR_PLATE_REMOTE`\r\n`

  Response from Server: `TRUE\r\nOK\r\n`

- ***Set Activator Duplicate Wait***

  Command: `SET_ACTIVATOR_DUPLICATE_WAIT <the time to wait between reading the same barcode in seconds>\r\n`

  Response: `OK\r\n`

  Upon receiving this command; DataPaq will record a value in seconds to repeatedly read a barcode on a rack the scanner when it is placed on the scanner every x seconds. If no positive integer is passed then ERR29 will be thrown.

  Example

  Received by Server: SET_ACTIVATOR_DUPLICATE_WAIT 1`\r\n`

  Response from Server: `OK\r\n`

- ***Get Activator Duplicate Wait***

  Command: `GET_ACTIVATOR_DUPLICATE_WAIT\r\n`

  Response: 1\r\n`OK\r\n`

Upon receiving this command; DataPaq will return the value used to repeatedly read a barcode on a rack the scanner when it is placed on the scanner every x seconds.

Example

Received by Server: GET_ACTIVATOR_DUPLICATE_WAIT`\r\n`

Response from Server: `1\r\nOK\r\n`

- ***Set Activator Read Duplicate***

  Command: `SET_ACTIVATOR_READ_DUPLICATE\r\n`

  Response: `OK\r\n`

  Upon receiving this command; DataPaq will repeatedly read a barcode on a rack when it is placed on the scanner every x seconds; the x seconds is set by SET_ACTIVATOR_READ_DUPLICATE_WAIT.

  Example

  Received by Server: SET_ACTIVATOR_READ_DUPLICATE`\r\n`

  Response from Server: `OK\r\n`

- ***Set Activator No Read Duplicate***

  Command: `SET_ACTIVATOR_NO_READ_DUPLICATE\r\n`

  Response: `OK\r\n`

  Upon receiving this command; DataPaq will no longer repeatedly read a barcode on a rack when it is placed on the scanner.

  Example

  Received by Server: SET_ACTIVATOR_NO_READ_DUPLICATE`\r\n`

  Response from Server: `OK\r\n`

- ***Get Activator Read Duplicate***

  Command: `GET_ACTIVATOR_READ_DUPLICATE\r\n`

  Response: <TRUE|FALSE>\r\n`OK\r\n`

Upon receiving this command; DataPaq will return back if it is set to repeatedly read a barcode on a rack the scanner when it is placed on the scanner every x seconds; the x seconds is set by SET_ACTIVATOR_READ_DUPLICATE_WAIT.

Example

Received by Server: GET_ACTIVATOR_READ_DUPLICATE`\r\n`

Response from Server: `TRUE\r\nOK\r\n`

- ***Set Activator Uid***

  Command: `SET_ACTIVATOR_UID <the uid to read when activated>\r\n`

  Response: `OK\r\n`

  This will set the uid to scan when the activator triggers a plate read. The uid must be present and known; if it is not then ERR26 will be thrown.

  Example

  Received by Server: SET_ACTIVATOR_UID 96deep`\r\n`

  Response from Server: `OK\r\n`

- ***Get Activator Uid***

  Command: `GET_ACTIVATOR_UID\r\n`

  Response: 96deep\r\n`OK\r\n`

  This will return the uid to scan when the activator triggers a plate read. If no activator uid has been set then ERR34 will be returned.

  Example

  Received by Server: GET_ACTIVATOR_UID`\r\n`

  Response from Server: `96deep\r\nOK\r\n`

- ***Set Activator Result Format***

  Command: `SET_ACTIVATOR_RESULT_FORMAT <the result format to export when activated>\r\n`

  Response: `OK\r\n`

This will set the result format to use when the activator triggers a plate read. The format must be xml, text, excel or json. If one of these is not provided ERR2 will be thrown. If this is not set then the format will default to xml.

Example

Received by Server: SET_ACTIVATOR_RESULT_FORMAT xml`\r\n`

Response from Server: `OK\r\n`

- ***Get Activator Result Format***

  Command: `GET_ACTIVATOR_RESULT Format\r\n`

  Response: json\r\n`OK\r\n`

  This will return the result format exported to scan when the activator triggers a plate read. The value is either text, xml, json or excel – if no value has previously been set this will default to xml.

  Example

  Received by Server: GET_ACTIVATOR_RESULT_FORMAT`\r\n`

  Response from Server: `json\r\nOK\r\n`

Notification

If the activator is present and turned on to scan or plate presence notify the following commands may be received on the notification port:

- PLATE_PRESENT <Rack Barcode>\r\nOK\r\n : this will be triggered when a plate is detected on the scanner.
- SCAN_STARTED <uid> <Rack Barcode>\r\nOK\r\n : this will be triggered when a scan is triggered and about to start
- <SCAN RESULTS>\r\nOK\r\n : this will be the results of the scan; the format of which determined by the SET_ACTVATOR_RESULTS_FORMAT command

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 xml, text, json or excel
ERR3	The unique ID must be supplied on a calibrate
ERR4	Failed to cleanly shut down
ERR5	NOT USED
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
ERR18	NOT USED
ERR19	NOT USED
ERR20	NOT USED
ERR21	NOT USED
ERR22	Please provide an integer as a scale factor
ERR23	Too many connections
ERR24	Cannot set hardware port while hardware linear scanner not used
ERR25	Acceptable image formats are png, jpeg or bmp

ERR26	Uid not known
ERR27	Uid must be provided
ERR28	Uid must be provided on scan image
ERR29	Duplicate wait value must be a positive number
ERR30	Activator Uid not set
ERR31	Cannot activate while processing command
ERR32	Ziath linear scanner must be enabled to use the activator
ERR33	Failed to get connected scanners
ERR34	Activator uid not set

Detailed Error Codes

These are returned in response to a scan failing; note that when responding to the codes use the error code as the description may change to provide more detail and may also be local specific.

DESCRIPTION	CODE
CANNOT_LOAD_SETTINGS	1
SCANNER_ERROR	2
SCANNER_NOT_CONNECTED	3
SCANNER_OPERATION_ERROR	4
SCANNER_LOCK_ERROR	5
UNKNOWN_UID	6
UID_NOT_CALIBRATED	7
UNKNOWN_PLATE_TYPE	8
AUTOMATIC_FILE_EXPORT_FAILED	9
AUTOMATIC_DATABASE_EXPORT_FAILED	10
EXPORT_FORMATTING_FAILED	11
USER_CANCELLED	12
UNENCRYPTED_IMAGE	13
NOTCH_DETECTION_COMMS_ERROR	14
NOTCH_DETECTION_WRONG_WAY_ROUND	15
ACTIVATOR_UID_NOT_SET	16

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 lien 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);

        }
```