JoeScan DLL

Generated by Doxygen 1.6.3

Thu Oct 14 09:45:17 2010

Contents

1 JoeScan API Documentation

Tutorials:

- Tutorial for Synchronized Scanning
- Tutorial for Snapshot Scanning
- Tutorial for Parallel Request/Read Functions
- Tricks for High Speed Scanning

References:

- Managing Scanner Connections
- Finding Scanners on the Network
- Data Structures
- Encoder/Time Synchronized Scanning
- Scanners with Multiple Lasers
- Scan and Image Data

- Sending Parameter Files to the Scanner
- Configuring Calibration, IP Address, and Cable ID
- Checking the Scanner Status
- The Other Functions

Developing Applications With The API

In order to create a program that uses the API, you must `#include jcam_dll.h` in your own source code and link to the `jcam_dll.lib` file in your project. In Visual C++ .NET 2003, this is accomplished by going to:

Project > Properties > Linker > Input > Additional Dependencies

From there, enter the location of the `jcam_dll.lib` file. I suggest that you put it in your project directory.

Static Linking

If you statically link the API functions into your own application, you won't need to distribute `jcam_dll.dll` with your application. However, you do need to do five things:

- Link jcam_dll.lib with your project.
- `#define JCAM_STATIC_LIB`
- `#include jcam_dll.h`
- Use the `joescan` namespace.
- Call `jsInitialize()` before calling any other Joescan API functions.

```
#define JCAM_STATIC_LIB
#include "jcam_dll.h"
using namespace joescan;

main(int argc, char ** argv)
{
    if(jsInitialize() == FALSE)
        exit(0);

    //Your code goes here...
}
```

Dynamic Linking

If you dynamically link the API functions to your own application, you will need to distribute `jcam_dll.dll` with your application. `jsInitialize()` will be automatically called by the Windows dynamic library loader. That leaves you with five things you must do:

- Link jcam_dll.lib with your project.
- Distribute jcam_dll.dll with your application.
- `#define JCAM_DYNAMIC_LIB`
- `#include jcam_dll.h`
- Use the `joescan` namespace.

You should put the `jcam_dll.dll` file in your project's Debug and Release directories so your program runs when you test it. When you distribute your application, the `jcam_dll.dll` file must be in the search path for the program, probably the install directory.

```
#define JCAM_DYNAMIC_LIB
#include "jcam_dll.h"
using namespace joescan;
```

All the symbolic constants used in this manual are defined in the file `jcam_dll.h`.

The `joescan` Namespace

If you're using C++, you can either use the `joescan` namespace as above and in the examples, or you can fully qualify its members. For example:

```
joescan::JCONNECTION jc = joescan::jsOpenConnection("192.168.1.105");
```

Yes, this is very simple, basic stuff, but somewhere there's someone who's learning it for the first time.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

6 Module Documentation

6.1 Scan and Image Data

Functions

- JCAM_DLL_API int STDCALL jsGetImage (JCONNECTION const jc, jsImage *const image)

  Requests and reads an image.

- JCAM_DLL_API int STDCALL jsGetScan (JCONNECTION const jc, jsScan *const scan)

  Requests and reads a scan.

- JCAM_DLL_API int STDCALL jsGetImageScan (JCONNECTION const jc, jsImage *const image, jsScan *const scan)

  Requests and reads an image and scan.

- JCAM_DLL_API int STDCALL jsGetImagePixel (jsImage const *const image, size_t const x, size_t const y)

  Returns a pixel value from an image.

6.1.1 Detailed Description

These functions will probably not be used very often. Most optimizers aren't interested in the images or the raw scans, but if you do need them, here they are.

6.1.2 Function Documentation

6.1.2.1 JCAM_DLL_API int STDCALL jsGetImage (JCONNECTION const *jc*, jsImage *const *const *image*)

Requests and reads an image.

Parameters

jc The connection to read the image from.

image Where the image is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `image` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.1.2.2 JCAM_DLL_API int STDCALL jsGetScan (JCONNECTION const *jc*, jsScan *const *const *scan*)

Requests and reads a scan.

Parameters

jc The connection to read the scan from.

scan Where the scan is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `scan` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.1.2.3 JCAM_DLL_API int STDCALL jsGetImageScan (JCONNECTION const *jc*, jsImage *const *image*, jsScan *const *scan*)

Requests and reads an image and scan.

Parameters

jc The connection to read the image and scan from.

image Where the image is stored.

scan Where the scan is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `scan` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.1.2.4 JCAM_DLL_API int STDCALL jsGetImagePixel (jsImage const *const *image*, size_t const *x*, size_t const *y*)

Returns a pixel value from an image.

This function checks to ensure that `image` isn't `NULL`, but doesn't do any error checking to ensure that `x` and `y` are within bounds.

`(0, 0)` is the upper-left pixel. `(image->numberHorizontal - 1, image->numberVertical - 1)` is the lower-right pixel.

Parameters

image The image to get a pixel value from.

x The x coordinate of the pixel to get.

y The y coordinate of the pixel to get.

Returns

The greyscale pixel value in the range of 0 to 255.
0 if `image` is `NULL`.

6.2 Tricks for High Speed Scanning

High speed scanning, which is roughly anything over 300 scans per second, can get tricky. These are the factors that must go into your system.

1. **The `MaxLaserOn` Parameter**

   `MaxLaserOn` is related to the inverse of the number of the scans per second that you want and the 300 microsecond overhead for reading image data from the camera. Here's the formula for calculating the `MaxLaserOn` value:

   $1000 / \text{Scan Rate} - 0.3 =$ `MaxLaserOn`

   A system that requires 500 scans per second would have a `MaxLaserOn` calculated like this:

   $1000 / 500 - 0.3 = 2 - 0.3 = 1.7$ (milliseconds)

   The down side to low exposure times is that the laser may not be bright enough to fully illuminate the object. You should move your scanner closer to the object in that case, or lower the `LaserThreshold` parameter.

2. **Reducing Analyzed Image Area**

   - **Scan Window**

     JSDiag visually displays the current Scan Window as a yellow bounding box in the Image View and Laser Image View. The Scan Window should be reduced to the smallest possible area to improve speed and ambient light immunity.

     The scanner will only analyze the absolute minimum portion of the image that could produce valid data points, increasing the maximum Scan Rate. Also, automatic exposure is calculated using data points only from within the Scan Window, which improves ambient light immunity. Due to the camera type, the scanner analyzes the data from the near end until it reaches the end of the valid area.

     The Scan Window parameters are `WindowTop`, `WindowLeft`, `WindowRight` and `WindowBottom`.

   - **Far Away Scan Window**

     A common situation for scan windows is that they are far away from the scanner. Valid data can not appear near the scanner, so in this example the Scan Window is defined to include only the area of interest. In this case, the image will be analyzed from the near part of the image to the far extent of the valid Scan Window. The remainder of the image will not be analyzed, which will reduce the amount of time it takes to process it.

     In the image below, the analyzed portions of the image are highlighted in red, and the Scan Window is the yellow box.

   - **Near Scan Window**

     If the Scan Window includes the area closer to the scanner than it can see, then only the Scan Window is analyzed.

   - **The `ImagePercentage` Parameter (Deprecated)**

     `ImagePercentage` is a brute force way of accomplishing the same thing, but it is more quantifiable. If you set the ImagePercentage to 33,

then a maximum of one third of the image will be analyzed. We recommend not using this parameter; the Scan Window accomplishes the same thing as perfectly as possible and has more reliable ambient light immunity.

3. **Background Subtraction (Don't use it for high speed)**

   Setting `BackgroundSubtraction` parameter to true will cut the effective Scan Rate in half for a given exposure time. This is because one exposure and read-out period is used for the background image without the laser and one for the data image with the laser. Generally, high speed systems should avoid using the `BackgroundSubtraction` parameter.

4. **What Happens When The Scanner Overdrives**

   While in the process of tuning a system, these are the indicators of trying to run a scanner too fast.

   - The `flags` field of the `jsProfile` will have bit 0 set to 1.
   - Your scan data will stop at the point where the image was analyzed to when the next scan was triggered. For example, if the image was around 65% analyzed when the next scan was triggered, you would only get data points from the top 65% of the image.

6.3 Scanners with Multiple Lasers

Functions

- JCAM_DLL_API int STDCALL jsSendProfileRequestN (JCONNECTION const jc, jsLaserIndex const laserIndex)

  Sends a request for a profile from the specified laser.

- JCAM_DLL_API int STDCALL jsReadProfileN (JCONNECTION const jc, jsProfile *const profile)

  Reads a profile from the connection.

- JCAM_DLL_API int STDCALL jsGetProfileN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsProfile *const profile)

  Requests and reads a profile from the specified laser.

- JCAM_DLL_API int STDCALL jsGetProfileFromAllLasers (JCONNECTION const jc, jsProfile *const profiles, size_t const cProfiles)

  Requests and reads a profile from each laser the scanner has.

- JCAM_DLL_API int STDCALL jsGetImageN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsImage *const image)

  Requests and reads an image with the specified laser on.

- JCAM_DLL_API int STDCALL jsGetScanN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsScan *const scan)

  Requests and reads a scan with the specified laser on.

- JCAM_DLL_API int STDCALL jsGetImageScanN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsImage *const image, jsScan *const scan)

  Requests and reads an image and scan with the specified laser on.

6.3.1 Detailed Description

These functions allow you to choose which laser to use. While only the JS20-DL has more than one laser, you can use these functions with any JS20 so long as you specify the valid laser, `LASER0`.

If you attempt to access an invalid laser, like `LASER1` on a single-laser JS20, the connection will close. This is fail-fast behavior. If an empty `jsProfile` were returned, people would spend all day trying to figure out why the profile had no data. Now, they'll simply spend all day trying to figure out why the connection died.

You may be thinking to yourself, "Why an `enum` instead of an `int`?" Good question. The answer is type safety. This way, you can't index an inappropriate laser like -3 - it simply won't compile. You'll have to actively work to index an inappropriate laser instead of having it accidentally happen.

If you still want to use an `int`, put the `enum` values into an array like this and pass in the value from the array:

```
jsLaserIndex laserIndexArray[] = {LASER0, LASER1};
for(int i = 0; i < 2; i++)
  jsGetProfileN(jc, laserIndexArray[i], &profile);
```

6.3.2 Function Documentation

6.3.2.1 JCAM_DLL_API int STDCALL jsSendProfileRequestN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*)

Sends a request for a profile from the specified laser.

May **not** be called while the scanner is in Synchronized Scanning Mode.

Nothing is read from the connection during this call. Increments the number of outstanding requests for this connection by one.

Parameters

jc The connection to send the request to.

laserIndex Which laser to use for the scan.

See also

`jsReadProfileN()`
`jsGetNumberOfOutstandingRequests()`
`jsCleanUpOutstandingRequests()`

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.3.2.2 JCAM_DLL_API int STDCALL jsReadProfileN (JCONNECTION const *jc*, jsProfile *const *const *profile*)

Reads a profile from the connection.

May **not** be called while the scanner is in Synchronized Scanning Mode.

A call to `jsSendProfileRequestN()` must have been made to the specified `JCONNECTION` before this function is called. Otherwise the call will block and eventually time out waiting for a profile that isn't coming.

Decrements the number of outstanding requests for this connection by one.

See also

`jsSendProfileRequestN()`
`jsGetNumberOfOutstandingRequests()`
`jsCleanUpOutstandingRequests()`

Parameters

jc The connection to read the profile from.

profile Where the profile is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `profile` is `NULL`.
`SCANNER_FAILURE` on all connection failures.
`SOCKET_TIMEOUT` if the read took longer than 5 seconds.

6.3.2.3 JCAM_DLL_API int STDCALL jsGetProfileN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, jsProfile *const *profile*)

Requests and reads a profile from the specified laser.

May **not** be called while the scanner is in Synchronized Scanning Mode.

Parameters

jc The connection to read a profile from.

laserIndex Which laser to use for the scan.

profile Where the profile is stored.

Returns

0 on success.
INVALID_PARAMETER if jc or profile is NULL.
SCANNER_FAILURE on all connection failures.
SOCKET_TIMEOUT if the read took longer than 5 seconds.

6.3.2.4 JCAM_DLL_API int STDCALL jsGetProfileFromAllLasers (JCONNECTION const *jc*, jsProfile *const *profiles*, size_t const *cProfiles*)

Requests and reads a profile from each laser the scanner has.

May **not** be called while the scanner is in Synchronized Scanning Mode.

If cProfiles indicates profiles doesn't have enough space, then extra profiles are discarded. Really, profiles should point to an array with enough space for a profile from every laser.

Parameters

jc The connection to read the profiles from.

profiles Pointer to an array of jsProfile's.

cProfiles The number of jsProfile's that profiles points to.

Returns

0 on success.
INVALID_PARAMETER if jc or profile is NULL.
SCANNER_FAILURE on all connection failures.
SOCKET_TIMEOUT if the read took longer than 5 seconds.

6.3.2.5 JCAM_DLL_API int STDCALL jsGetImageN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, jsImage *const *image*)

Requests and reads an image with the specified laser on.

Parameters

jc The connection to read the image from.

laserIndex Which laser to use for the image.

image Where the image is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `image` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.3.2.6 JCAM_DLL_API int STDCALL jsGetScanN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, jsScan *const *scan*)

Requests and reads a scan with the specified laser on.

Parameters

jc The connection to read the scan from.

laserIndex Which laser to use for the scan.

scan Where the scan is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `scan` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.3.2.7 JCAM_DLL_API int STDCALL jsGetImageScanN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, jsImage *const *image*, jsScan *const *scan*)

Requests and reads an image and scan with the specified laser on.

Parameters

jc The connection to read the image and scan from.

laserIndex Which laser to use for the image and scan.

image Where the image is stored.

scan Where the scan is stored.

Returns

0 on success.
INVALID_PARAMETER if jc or scan is NULL.
SCANNER_FAILURE on all connection failures.

6.4 Managing Scanner Connections

Functions

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnection (char const *const host)

  Opens a connection to the specified host.

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionInt (int const host)

  Opens a connection to the specified host.

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionBase (char const *const host, UINT32 const cableID)

  Opens a connection to the specified scanner using Base IP + Cable ID addressing.

- JCAM_DLL_API int STDCALL jsCloseConnection (JCONNECTION const jc)

  Closes a JCONNECTION.

6.4.1 Detailed Description

JCONNECTION's are allocated data structures that must be properly de-allocated; otherwise, there will be memory leaks in your program. Consequently, every successful call to jsOpenConnection() or jsOpenConnectionInt() must have a corresponding call to jsCloseConnection(). If the open call failed, the returned JCONNECTION will be NULL and the close call is not necessary, although it is perfectly safe.

After calling jsCloseConnection(), the JCONNECTION is an invalid pointer and should be set to NULL.

If a call fails, call `WSAGetLastError()` to get a more specific reason for failure according to Microsoft's documentation.

6.4.2 Function Documentation

6.4.2.1 JCAM_DLL_API JCONNECTION STDCALL jsOpenConnection (char const *const *host*)

Opens a connection to the specified host.

All successful calls to this function **must** be closed with a call to `jsCloseConnection()`.

See also

`jsCloseConnection()`

Parameters

host Can be in either xxx.xxx.xxx.xxx format or can be a DNS name for lookup.

Returns

The `JCONNECTION` on success.
`NULL` if a failure occurred.

6.4.2.2 JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionInt (int const *host*)

Opens a connection to the specified host.

All successful calls to this function **must** be closed with a call to `jsCloseConnection()`.

See also

`jsCloseConnection()`

Parameters

host **Must be in network byte order.** The best way to get this is by using the Scanner Discovery functions.

Returns

The JCONNECTION on success.
NULL if a failure occurred.

6.4.2.3 JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionBase (char const *const *host*, UINT32 const *cableID*)

Opens a connection to the specified scanner using Base IP + Cable ID addressing.

All successful calls to this function **must** be closed with a call to jsCloseConnection().

See also

jsCloseConnection()

Parameters

host Must be in xxx.xxx.xxx.xxx format.

cableID The Cable ID of the scanner to connect to.

Returns

The JCONNECTION on success.
NULL if a failure occurred.

6.4.2.4 JCAM_DLL_API int STDCALL jsCloseConnection (JCONNECTION const *jc*)

Closes a JCONNECTION.

It is safe to pass in NULL.

If a function returns SCANNER_FAILURE, you must close the connection with this function. JCONNECTION's are allocated data structures that must be properly deallocated, otherwise there will be memory leaks in your program.

See also

jsOpenConnection()
jsOpenConnectionInt()
jsOpenConnectionBase()

Parameters

jc The connection to be closed.

Returns

0 on success.
-1 on all failures.

6.5 The Other Functions

Functions

- JCAM_DLL_API int STDCALL jsGetJcamDllMajorVersionNumber ()
  Returns the major revision number of the DLL.
- JCAM_DLL_API int STDCALL jsGetJcamDllMinorVersionNumber ()
  Returns the minor revision number of the DLL.
- BOOL jsInitialize ()
  Initializes TCP/IP communications.
- void jsCleanup ()
  Gracefully ends TCP/IP communications.

6.5.1 Detailed Description

These didn't fit nicely into another category. You can use the DLL Version number for help diagnosing problems — not that there are any bugs in this DLL, naturally. If for some reason you need to get a specific pixel from an image, you can also do that.

6.5.2 Function Documentation

6.5.2.1 JCAM_DLL_API int STDCALL jsGetJcamDllMajorVersionNumber ()

Returns the major revision number of the DLL.

6.5.2.2 JCAM_DLL_API int STDCALL jsGetJcamDllMinorVersionNumber ()

Returns the minor revision number of the DLL.

6.5.2.3 BOOL jsInitialize ()

Initializes TCP/IP communications.

In Windows, TCP/IP communications must be initialized. No one knows why, it simply is that way.

If you are using this as a statically linked library, call this function before you use the functions in this library, otherwise bad things will happen.

If you are using this as a dynamically linked library (DLL), you don't need to call this function, the library loader will do it for you.

Since

1.24

Returns

`TRUE` on success.
`FALSE` on failures.

6.5.2.4 void jsCleanup ()

Gracefully ends TCP/IP communications.

After a succesful call to `jsInitialize()`, this function can be called if no futher calls to joescan API functions will occur.

Since

1.24

6.6 Sending Parameter Files to the Scanner

Functions

- JCAM_DLL_API int STDCALL jsSendParameterFileToScanner (JCONNECTION const jc, char const *const filename)

  Sends a parameter file to the scanhead.

- JCAM_DLL_API int STDCALL jsSendParametersToScanner (JCONNECTION const jc, char const *const parameters)

  Sends parameters to the scanhead.

- JCAM_DLL_API int STDCALL jsGetParameterFileFromScanner (JCONNECTION const jc, char const *const filename)

  Reads the current parameter file from the scanhead and writes them to the specified file.

- JCAM_DLL_API size_t STDCALL jsGetNumberOfErrorMessages (JCONNECTION const jc)

  Returns the number of error messages after a `jsSendParameterFileToScanner()` call returned `OPERATION_FAILURE`.

- JCAM_DLL_API char const *const STDCALL jsGetErrorMessage (JCONNECTION const jc, size_t i)

  Returns the specified error message.

6.6.1 Detailed Description

These functions are for sending, reading, and checking correctness of parameter files sent to the scanner.

This sample code opens a connection and sends parameters to the scanner.

Please, please check your error codes from the function. If there is a syntax error in the parameters, the parameters will not be saved to the scanner. You should tell users what the errors are so they can fix them. There are line numbers and everything.

Yes, I realize there are `goto` statments in this code. It makes things cleaner in this case.

```
#define JCAM_STATIC_LIB
#include "jcam_dll.h"
#include <stdio.h>

using namespace joescan;

int main(int argc, char * argv[])
{
    if(jsInitialize() == FALSE)
        exit(0);

    size_t i = 0;
    JCONNECTION jc;

    if((jc = jsOpenConnection("192.168.1.105")) == NULL)
    {
        printf("Couldn't open connection.\n");
        goto CLEANUP;
    }
```

```
    switch(jsSendParameterFileToScanner(jc, "param.dat"))
    {
        case SCANNER_FAILURE:
            printf("Lost connection.\n");
            goto CLEANUP;
        case OPERATION_FAILURE:
            for(i = 0; i < jsGetNumberOfErrorMessages(jc); i++)
                printf("%s\n", jsGetErrorMessage(jc, i));
            break;
        default:
            printf("Sent parameters successfully.\n");
    }

CLEANUP:
    jsCloseConnection(jc);
    jc = NULL;
    return 0;
}
```

6.6.2 Function Documentation

6.6.2.1 JCAM_DLL_API int STDCALL jsSendParameterFileToScanner (JCONNECTION const *jc*, char const *const *const *filename*)

Sends a parameter file to the scanhead.

If OPERATION_FAILURE is returned, the error messages should be checked using jsGetNumberOfErrorMessages() and jsGetErrorMessage(). The error messages are stored inside the opaque JCONNECTION.

See also

jsGetErrorMessage()
jsSendParametersToScanner()
jsGetNumberOfErrorMessages()
jsGetParameterFileFromScanner()

Parameters

jc The connection to the scanner.

filename The name of the file to send to the scanner.

Returns

0 on success.
SCANNER_FAILURE on all connection failures.
INVALID_PARAMETER if jc or filename is NULL.
OPERATION_FAILURE on all other failures, including parse and file I/O errors.

6.6.2.2 JCAM_DLL_API int STDCALL jsSendParametersToScanner (JCONNECTION const *jc*, char const *const *const *parameters*)

Sends parameters to the scanhead.

If `OPERATION_FAILURE` is returned, the error messages should be checked using `jsGetNumberOfErrorMessages()` and `jsGetErrorMessage()`. The error messages are stored inside the opaque `JCONNECTION`.

See also

`jsGetErrorMessage()`
`jsGetNumberOfErrorMessages()`
`jsSendParameterFileToScanner()`
`jsGetParameterFileFromScanner()`

Parameters

jc The connection to the scanner.

parameters A NULL-terminated string with the parameters to send to the scanner.

Returns

0 on success.
`SCANNER_FAILURE` on all connection failures.
`INVALID_PARAMETER` if `jc` or `parameters` is `NULL`.
`OPERATION_FAILURE` on all other failures, including parse and file I/O errors.

6.6.2.3 JCAM_DLL_API int STDCALL jsGetParameterFileFromScanner (JCONNECTION const *jc*, char const *const *const *filename*)

Reads the current parameter file from the scanhead and writes them to the specified file.

See also

`jsSendParametersToScanner()`
`jsSendParameterFileToScanner()`

Parameters

jc the connection to the scanner.

filename The name of the file to save the parameters in. The file will be created if it doesn't exist.

Returns

0 on success.
SCANNER_FAILURE on all connection failures.
INVALID_PARAMETER if jc or filename is NULL.
OPERATION_FAILURE on all other failures.

6.6.2.4 JCAM_DLL_API size_t STDCALL jsGetNumberOfErrorMessages (JCONNECTION const *jc*)

Returns the number of error messages after a jsSendParameterFileToScanner() call returned OPERATION_FAILURE.

See also

jsGetErrorMessage()
jsSendParametersToScanner()
jsSendParameterFileToScanner()

Parameters

jc The connection to the scanner.

Returns

The number of error messages.
0 if jc is NULL.

6.6.2.5 JCAM_DLL_API char const* const STDCALL jsGetErrorMessage (JCONNECTION const *jc*, size_t *i*)

Returns the specified error message.

Do not modify the returned string.

See also

jsSendParametersToScanner()
jsGetNumberOfErrorMessages()
jsSendParameterFileToScanner()

Parameters

jc The connection to the scanner.

i The index of the error message to retrieve.

Returns

A C string of the error message.
`NULL` if `i` is out of range or `jc` is `NULL`.

6.7 Configuring Calibration, IP Address, and Cable ID

These functions allow you to read and set the scanner's calibration, IP address configuration, and Cable ID.

Functions

- JCAM_DLL_API int STDCALL jsReadOldPositionCalibrationsN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsOldCalibrationValue oldCalibrations[ ], UINT32 nCalibrations)

  Reads the specified number of old position calibrations for the selected laser from the scanner.

- JCAM_DLL_API int STDCALL jsReadPositionCalibrationN (JCONNECTION const jc, jsLaserIndex const laserIndex, double ∗xOffset, double ∗yOffset, double ∗roll)

  Reads the current position calibration for the selected laser from the scanner.

- JCAM_DLL_API int STDCALL jsSendPositionCalibrationN (JCONNECTION const jc, jsLaserIndex const laserIndex, double const xOffset, double const yOffset, double const roll)

  Sends a position calibration for the selected laser to the scanner.

- JCAM_DLL_API int STDCALL jsSetStaticIpInt (UINT32 const serialNumber, UINT32 const staticIpAddress, UINT32 const netmask)

  Sets a scanner's static IP address.

- JCAM_DLL_API int STDCALL jsSetStaticIpChar (UINT32 const serialNumber, char const ∗const staticIpAddress, char const ∗const netmask)

  Sets a scanner's static IP address.

- JCAM_DLL_API int STDCALL jsSetBaseIpInt (UINT32 const serialNumber, UINT32 const baseIpAddress, UINT32 const netmask)

  Sets a scanner's base IP address.

- JCAM_DLL_API int STDCALL jsSetBaseIpChar (UINT32 const serialNumber, char const ∗const baseIpAddress, char const ∗const netmask)

Sets a scanner's base IP address.

- JCAM_DLL_API int STDCALL jsSetCableId (UINT32 const serialNumber, UINT32 const cableId)

  Overrides the cable ID physically wired into a scanner.

- JCAM_DLL_API int STDCALL jsClearCableId (UINT32 const serialNumber)

  Reverts to the cable ID physically wired into a scanner.

6.7.1 Detailed Description

These functions allow you to read and set the scanner's calibration, IP address configuration, and Cable ID.

6.7.2 Function Documentation

6.7.2.1 JCAM_DLL_API int STDCALL jsReadOldPositionCalibrationsN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, jsOldCalibrationValue *oldCalibrations*[ ], UINT32 *nCalibrations*)

Reads the specified number of old position calibrations for the selected laser from the scanner.

See also

`jsSendPositionCalibrationN()`
`jsReadPositionCalibrationN()`

Parameters

jc The connection to the scanner.

laserIndex Which laser to read the position calibration for.

oldCalibrations An array of `jsOldCalibrationValue` objects to store the old calibrations in.

nCalibrations The number of elements in jsOldCalibrationValue.

Returns

The number of calibrations read on success.
0 if there are no old calibrations.

INVALID_PARAMETER if jc or oldCalibrations NULL or if laserIndex is invalid.
SCANNER_FAILURE on all connection failures.

6.7.2.2 JCAM_DLL_API int STDCALL jsReadPositionCalibrationN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, double * *xOffset*, double * *yOffset*, double * *roll*)

Reads the current position calibration for the selected laser from the scanner.

See also

jsSendPositionCalibrationN()
jsReadOldPositionCalibrationsN()

Since

Revision 1932

Parameters

jc The connection to the scanner.

laserIndex Which laser to read the position calibration for.

xOffset Stores the value of the scanner's x offset.

yOffset Stores the value of the scanner's y offset.

roll Stores the value of the scanner's rotation.

Returns

0 on success.
INVALID_PARAMETER if jc, xOffset, yOffset, or roll is NULL.
SCANNER_FAILURE on all connection failures.

6.7.2.3 JCAM_DLL_API int STDCALL jsSendPositionCalibrationN (JCONNECTION const *jc*, jsLaserIndex const *laserIndex*, double const *xOffset*, double const *yOffset*, double const *roll*)

Sends a position calibration for the selected laser to the scanner.

See also

jsReadPositionCalibrationN()
jsReadOldPositionCalibrationsN()

Parameters

jc The connection to the scanner.

laserIndex Which laser to send the position calibration for.

xOffset The new value of the scanner's x offset.

yOffset The new value of the scanner's y offset.

roll The new value of the scanner's rotation.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.7.2.4 JCAM_DLL_API int STDCALL jsSetStaticIpInt (UINT32 const *serialNumber*, UINT32 const *staticIpAddress*, UINT32 const *netmask*)

Sets a scanner's static IP address.

The IP address will be static; it will not depend on the Cable ID of the scanner. If the scanner is moved to a different location with a different Cable ID, it will have the same IP address.

Parameters

serialNumber The serial number of the scanner whose IP address you want to set.

staticIpAddress **Must be in network byte order.** An integer representing the IP address the scanner will use.

netmask **Must be in network byte order.** An integer representing the IP netmask the scanner will use.

Returns

0 on success.
-1 on all failures.

6.7.2.5 JCAM_DLL_API int STDCALL jsSetStaticIpChar (UINT32 const *serialNumber*, char const *const *staticIpAddress*, char const *const *netmask*)

Sets a scanner's static IP address.

The IP address will be static; it will not depend on the Cable ID of the scanner. If the scanner is moved to a different location with a different Cable ID, it will have the same IP address.

Parameters

serialNumber The serial number of the scanner whose IP address you want to set.

staticIpAddress The IP address the scanner will use.

netmask The IP netmask the scanner will use.

Returns

0 on success.
`INVALID_PARAMETER` if `baseIpAddress` or `netmask` is `NULL` or invalid.
-1 on all other failures.

6.7.2.6 JCAM_DLL_API int STDCALL jsSetBaseIpInt (UINT32 const *serialNumber*, UINT32 const *baseIpAddress*, UINT32 const *netmask*)

Sets a scanner's base IP address.

The IP address will be determined by adding the scanner's Cable ID to the base IP address.

Parameters

serialNumber The serial number of the scanner whose IP address you want to set.

baseIpAddress **Must be in network byte order.** An integer representing the base IP address the scanner will use to determine its final IP address.

netmask **Must be in network byte order.** An integer representing the IP netmask the scanner will use.

Returns

0 on success.
-1 on all failures.

6.7.2.7 JCAM_DLL_API int STDCALL jsSetBaseIpChar (UINT32 const *serialNumber*, char const *const *baseIpAddress*, char const *const *netmask*)

Sets a scanner's base IP address.

The IP address will be determined by adding the scanner's Cable ID to the base IP address.

Parameters

serialNumber The serial number of the scanner whose IP address you want to set.

baseIpAddress The base IP address the scanner will use to determine its final IP address.

netmask The IP netmask the scanner will use.

Returns

0 on success.
`INVALID_PARAMETER` if `baseIpAddress` or `netmask` is `NULL` or invalid.
-1 on all other failures.

6.7.2.8 JCAM_DLL_API int STDCALL jsSetCableId (UINT32 const *serialNumber*, UINT32 const *cableId*)

Overrides the cable ID physically wired into a scanner.

The new cable ID is persistent. If the scanner is rebooted, it will ignore the physically wired in cable ID. If the scanner is set to use Base+IP addressing, the scanner's IP address will change according to the new IP address.

See also

`jsClearCableId()`

Parameters

serialNumber The serial number of the scanner to change.

cableId The cable ID you wish the scanner to have.

Returns

0 on success.
`INVALID_PARAMETER` if `cableId` is over 31.
-1 on all other failures.

6.7.2.9 JCAM_DLL_API int STDCALL jsClearCableId (UINT32 const *serialNumber*)

Reverts to the cable ID physically wired into a scanner.

If the scanner is set to use Base+IP addressing, the scanner's IP address will change according to the new IP address.

See also

`jsSetCableId()`

Parameters

serialNumber The serial number of the scanner to change.

Returns

0 on success.
`INVALID_PARAMETER` `cableId` is over 31.
-1 on all other failures.

6.8 Finding Scanners on the Network

Functions

- JCAM_DLL_API int STDCALL jsFindAllScanners (jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover all scanners on the network.

- JCAM_DLL_API int STDCALL jsFindScannerByCableId (int const cableId, jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover the scanner(s) on the network with the specified Cable ID.

- JCAM_DLL_API int STDCALL jsFindScannerBySerialNumber (int const serialNumber, jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover the scanner on the network with the specified serial number.

6.8.1 Detailed Description

Scanner discovery follows a request/response model, whereby the user's application requests scanners that meet certain criteria (it can also request all available scanners), and the scanner(s) respond to the request by sending their ID information. The responses are not guaranteed to be returned in real-time; they're sent via UDP. Consequently, after the request, the functions used for discovering scanners block for approximately one second in order to reasonably ensure that all scanners get their responses in.

If there is a failure, everything will automatically be cleaned up. `WSAGetLastError()` can be called to get a more specific reason for failure according to Microsoft's documentation.

Internet-use fields will be explicit as to whether or not they are in network byte order.

6.8.2 Function Documentation

6.8.2.1 JCAM_DLL_API int STDCALL jsFindAllScanners (jsResponsePacket *responses*[ ], const size_t *nResponses*)

Attempts to discover all scanners on the network.

Up to `nResponses` from the scanners will be stored in `responses`.

Blocks for approximately one second while waiting for responses from the scanners.

Parameters

responses An array for holding responses from the scanners on the network.

nResponses The number of elements in `responses`.

Returns

The number of responses on success.
-1 on all failures.

6.8.2.2 JCAM_DLL_API int STDCALL jsFindScannerByCableId (int const *cableId*, jsResponsePacket *responses*[ ], const size_t *nResponses*)

Attempts to discover the scanner(s) on the network with the specified Cable ID.

If there is more than one scanner with the same Cable ID, they will both respond.

Up to `nResponses` from the scanners will be stored in `responses`.

Blocks for approximately one second while waiting for responses from the scanners.

Parameters

cableId The cable ID of the scanner(s) you want to discover.

responses An array for holding responses from the scanners on the network.

nResponses The number of elements in `responses`.

Returns

The number of responses on success.
-1 on all failures.

6.8.2.3 JCAM_DLL_API int STDCALL jsFindScannerBySerialNumber (int const *serialNumber*, jsResponsePacket *responses*[ ], const size_t *nResponses*)

Attempts to discover the scanner on the network with the specified serial number.

Since serial numbers are unique, there should only ever be one scanner responding to this search.

Up to `nResponses` from the scanners will be stored in `responses`.

Blocks for approximately one second while waiting for responses from the scanners.

Parameters

serialNumber The serial number of the scanner you want to discover.

responses An array for holding responses from the scanners on the network.

nResponses The number of elements in `responses`.

Returns

The number of responses on success.
-1 on all failures.

6.9 Tutorial for Parallel Request/Read Functions

If you have connections to lots of scanners (more than six or so), the total roundtrip time for requesting and reading a profile from each one can take a while. Fortunately, there's an easy way to parallelize requests and reads from groups of scanners. The key is to send all the requests at once, and then read all the profiles back at once. This causes all the scanners' processing time and network travel time to happen simultaneously.

To do this during Synchronized Scanning Mode, do something like this:

1. `jsEnterEncoderSyncMode()` or `jsEnterTimeSyncMode()` for each connection to start Synchronized Scanning Mode.
2. `jsSendMultipleProfileRequest()` for each connection.
3. `jsReadMultipleProfiles()` for each connection.

4. jsExitSyncMode() for each connection to stop Synchronized Scanning Mode.

For snapshot scanning, care must be taken to ensure that the scanners don't see each other's lasers. Do something like this:

1. Call jsSendProfileRequestN() for each connection.
2. Call jsReadProfileN() for each connection.

```
#include <vector>
#define JCAM_STATIC_LIB
#include "jcam_dll.h"

using namespace joescan;

//Returns the first failed connection if there is one
JCONNECTION readProfiles(std::vector<JCONNECTION> &connections, std::vector<jsPro
      file> &profiles)
{
    for(i = 0; i < connections.size(); ++i)
    {
        //Request one profile from each scanner
        switch(jsSendMultipleProfileRequest(connections[i], 1))
        {
            case 0:
                break;
            case INVALID_PARAMETER:
                printf("Invalid parameter to jsSendMultipleProfileRequest().\n");

                return connections[i];
            case SCANNER_FAILURE:
                printf("Scanner failure during jsSendMultipleProfileRequest().\n"
      );
                return connections[i];
        }
    }

    //Make sure there's enough storage space for one profile from each scanner
    profiles.resize(connections.size());

    for(i = 0; i < connections.size(); ++i)
    {
        //Read one profile from each scanner
        switch(jsReadMultipleProfiles(connections[i], &profiles[i], 1))
        {
            case 0:
                break;
            case INVALID_PARAMETER:
                printf("Invalid parameter to jsReadMultipleProfiles().\n");
                return connections[i];
            case SCANNER_FAILURE:
                printf("Scanner failure during jsReadMultipleProfiles().\n");
                return connections[i];
        }
```

```
    }

    //No failed connections, return NULL.
    return 0;
}
```

6.10 Tutorial for Snapshot Scanning

You just want to get some data from the scanner quickly. Here's how:

1. Call `jsInitialize()` if the Joescan API is statically linked.
2. Open a connection using `jsOpenConnection()`.
3. Save the parameters using `jsSendParameterFileToScanner()` in case a scanner has been swapped out.
4. Get profiles using `jsGetProfile()`.
5. Close the connection using `jsCloseConnection()`.

```
#define JCAM_STATIC_LIB
#include <string>
#include <stdio.h>
#include "jcam_dll.h"

using namespace joescan;

int main(int argc, char **argv)
{
    if(jsInitialize() == FALSE)
    {
        printf("Could not initialize TCP\IP communications.\n");
        return -1;
    }

    std::string ip("192.168.1.205");
    if(argc > 1)
        ip = argv[1];

    JCONNECTION jc = jsOpenConnection(ip.c_str());
    if(jc == 0)
    {
        printf("Could not open connection to %s.\n", ip.c_str());
        return -1;
    }

    switch(jsSendParameterFileToScanner(jc, "param.dat"))
    {
        case SCANNER_FAILURE:
            printf("Lost connection.\n");
            jsCloseConnection(jc);
            jc = 0;
```

```
            return -1;
        case OPERATION_FAILURE:
            for(i = 0; i < jsGetNumberOfErrorMessages(jc); i++)
                printf("%s\n", jsGetErrorMessage(jc, i));
            jsCloseConnection(jc);
            jc = 0;
            return -1;
        default:
            printf("Sent parameters successfully.\n");
    }

    jsProfile profile;

    while(PLC.isLogVisible())
    {
        switch(jsGetProfile(jc, &profile))
        {
            case 0:
                logModel.addProfile(profile);
                break;
            case PROFILE_UNAVAILABLE:
                //In normal mode, this return value is impossible.
                printf("PROFILE_UNAVAILABLE\n");
                break;
            case INVALID_PARAMETER:
                printf("INVALID_PARAMETER\n");
                jsCloseConnection(jc);
                jc = 0;
                return -1;
            case SCANNER_FAILURE:
                printf("SCANNER_FAILURE\n");
                jsCloseConnection(jc);
                jc = 0;
                return -1;
        }
    }

    optimizer.fullyOptimize(logModel);
    jsCloseConnection(jc);
    jc = 0;
    return 0;
}
```

6.11 Checking the Scanner Status

Functions

- JCAM_DLL_API int STDCALL jsGetScannerStatusFromScanner (JCONNECTION const jc)

  Reads status from the scanner into a data structure inside the opaque JCONNECTION.

- JCAM_DLL_API int STDCALL jsGetScannerStatusValue (JCONNECTION

const jc, size_t i, int ∗value)

Gets the specified status value from the data structure inside the JCONNECTION.

- JCAM_DLL_API int STDCALL jsGetStatusDescriptionFromScanner (JCONNECTION const jc, UINT32 i, char ∗const description, UINT32 description_length)

  Reads a descriptive text string from the scanner for the particular status index.

6.11.1 Detailed Description

The order of operations here is important.

1. `jsGetScannerStatusFromScanner()`
2. `jsGetScannerStatusValue()`

Here are the valid indexes:

1. DSP Version Code
2. Mode: 0-Init, 1-Idle, 2-Expose, 3-Scan
3. Host Request: 0-None, 1-Scan Cam, 2-Laser Scan
4. Current Camera Exposure Time
5. Current Laser On Time
6. Current Encoder Count
7. Optically Isolated Inputs
8. Results of Last Host Interface Communication
9. Dark Pixel Value
10. Current Bias (offset)
11. End Code
12. Scan Server Version
13. Serial Number

```
#include "jcam_dll.h"
#include <stdio.h>

using namespace joescan;
```

```
int main(int argc, char * argv[])
{
    JCONNECTION jc;

    if((jc = jsOpenConnection("192.168.1.205")) == NULL)
    {
        printf("Couldn't open connection.\n");
        goto CLEANUP;
    }

    switch(jsGetScannerStatusFromScanner(jc))
    {
        case INVALID_PARAMETER:
            printf("Invalid Parameter.\n");
            goto CLEANUP;
        case SCANNER_FAILURE:
            printf("Scanner Failure in jsGetScannerStatusFromScanner ().\n");
            goto CLEANUP;
        default:
            printf("Got status from scanner.\n");
            break;
    }

    size_t i = 0;
    int value;
    char statusDescription[128];

    bool finished = false;
    while(finished == false)
    {
        switch(jsGetStatusDescriptionFromScanner(jc, i, statusDescription, 128))
        {
            case INVALID_PARAMETER:
                printf("Invalid Parameter.\n");
                break;
            case SCANNER_FAILURE:
                printf("Scanner Failure in jsGetStatusDescriptionFromScanner().\n
      ");
                goto CLEANUP;
            default:
                break;
        }

        switch(jsGetScannerStatusValue (jc, i, &value))
        {
            case INVALID_PARAMETER:
                printf("%2d Invalid parameter -- Out of bounds index.\n", i);
                finished = true;
                break;
            default:
                printf("%2d %s %d\n", i, statusDescription, value);
                break;
        }
        ++i;
    }

CLEANUP:
```

```
    jsCloseConnection(jc);
    jc = NULL;
    return 0;
}
```

6.11.2 Function Documentation

6.11.2.1 JCAM_DLL_API int STDCALL jsGetScannerStatusFromScanner (JCONNECTION const *jc*)

Reads status from the scanner into a data structure inside the opaque JCONNECTION.

See also

jsGetScannerStatusValue()
jsGetStatusDescriptionFromScanner()

Parameters

jc The connection to the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE if the connection closes.

6.11.2.2 JCAM_DLL_API int STDCALL jsGetScannerStatusValue (JCONNECTION const *jc*, size_t *i*, int * *value*)

Gets the specified status value from the data structure inside the JCONNECTION.

See also

jsGetScannerStatusFromScanner()
jsGetStatusDescriptionFromScanner()

Parameters

jc The connection to the scanner.

i The index of the status value to retrieve.

value Pointer to an int that will hold the value.

Returns

0 on success.
INVALID_PARAMETER if jc or value is NULL, or if i is out of range.

6.11.2.3 JCAM_DLL_API int STDCALL jsGetStatusDescriptionFromScanner (JCONNECTION const *jc*, UINT32 *i*, char *const *description*, UINT32 *description_length*)

Reads a descriptive text string from the scanner for the particular status index.

The description length should be a minimum of 128 characters. After this call, description will be properly null-terminated. If description isn't long enough, the result will be truncated, but still properly null-terminated. The first valid index is 0, the last valid index is 12.

Invalid indexes will return the string "ERROR unsuported status type".

See also

jsGetScannerStatusValue()
jsGetScannerStatusFromScanner()

Parameters

jc The connection to the scanner.

i The index of the status name to retrieve.

description The character buffer that will contain the status name.

description_length The number of characters that description points to.

Returns

0 on success.
INVALID_PARAMETER if jc or description is NULL, or if i is out of range.
SCANNER_FAILURE if the connection closes.

6.12 Encoder/Time Synchronized Scanning

Functions

- JCAM_DLL_API int STDCALL jsEnterStartScanTriggeredMode (JCONNECTION const jc)

  Causes the scanner to enter Start Scan Triggered Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsEnterEncoderSyncMode (JCONNECTION const jc)

  Causes the scanner to enter Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsEnterTimeSyncMode (JCONNECTION const jc)

  Causes the scanner to enter Time Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsStartPulseMaster (JCONNECTION const jc, int pulseInterval, int pulseCount)

  Causes the scanner to ouput a pulse train on the Start Scan I/O. The period and number of pulses are configurable.

- JCAM_DLL_API int STDCALL jsStopPulses (JCONNECTION const jc)

  Causes the scanner to stop generating a pulse train on the Start Scan I/O.

- JCAM_DLL_API int STDCALL jsHaltSyncMode (JCONNECTION const jc)

  Causes the scanner to halt Time or Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsExitSyncMode (JCONNECTION const jc)

  Causes the scanner to exit Time or Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsSendMultipleProfileRequest (JCONNECTION const jc, UINT32 const nProfiles)

  Sends a request for up to `nProfiles` profiles to the scanner while in Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsReadMultipleProfiles (JCONNECTION const jc, jsProfile *const profiles, UINT32 nProfiles)

  Reads up to `nProfiles` profiles from `jc` after a `jsSendMultipleProfileRequest()` call in Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsReadMultipleProfilesTimeout (JCONNECTION const jc, jsProfile *const profiles, UINT32 cProfiles, long timeoutMilliseconds)

  Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsGetMultipleProfiles (JCONNECTION const jc, jsProfile *const profiles, UINT32 const cProfiles)

  Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsGetProfile (JCONNECTION const jc, jsProfile *const profile)

  Requests and reads a profile from the default laser.

- JCAM_DLL_API size_t STDCALL jsGetNumberOfOutstandingRequests (JCONNECTION const jc)

  Returns the number of unread requests from either `jsSendProfileRequestN()` *or* `jsSendMultipleProfileRequest()`.

- JCAM_DLL_API int STDCALL jsCleanUpOutstandingRequests (JCONNECTION const jc)

  Reads and discards all the profiles in the TCP/IP stack generated by `jsSendProfileRequestN()` *or* `jsSendMultipleProfileRequest()`.

- JCAM_DLL_API int STDCALL jsSetEncoderValue (JCONNECTION const jc, UINT16 newEncoderValue)

  Sets the scanner's internal encoder value.

- JCAM_DLL_API int STDCALL jsSetEncoderValue32 (JCONNECTION const jc, UINT32 newEncoderValue)

  Sets the scanner's internal encoder value.

6.12.1 Detailed Description

During Synchronized Scanning Mode, the scanner will automatically scan at specific intervals. The interval is set in the scanner parameters, and is based on either the encoder or time.

1. The scanner powers up into Non-Synchronized Scanning Mode.
2. Enter Synchronized Scanning Mode by calling one of these functions:
   - `jsEnterEncoderSyncMode()`
   - `jsEnterTimeSyncMode()`
3. In Synchronized Scanning Mode, retrieve profiles from the scanner by calling any of these functions:
   - `jsGetProfile()`
   - `jsGetMultipleProfiles()`
   - `jsSendMultipleProfileRequest()` followed by `jsReadMultipleProfiles()`
4. Leave Synchronized Scanning Mode by:

- Call jsEnterEncoderSyncMode() or jsEnterTimeSyncMode() to discard any remaining profiles and restart Synchronized Scanning Mode.
- Call jsExitSyncMode() to discard any remaining profiles and go to Non-Synchronized Scanning Mode.
- Stop the scanner from accumulating new scans by calling jsHaltSyncMode(), then retrieve any remaining profiles. Call jsExitSyncMode() to return to Non-Synchronized Scanning Mode, or jsEnterEncoderSyncMode() or jsEnterTimeSyncMode() to restart Synchronized Scanning Mode.

If you call a non-synchronized function during Synchronized Scanning Mode, the scanner will discard any remaining profiles, leave Synchronized Scanning Mode, and close the connection.

jsGetProfile() is the only function that can be called in both Non-Synchronized and Synchronized Scanning Mode.

6.12.2 Function Documentation

6.12.2.1 JCAM_DLL_API int STDCALL jsEnterStartScanTriggeredMode (JCONNECTION const *jc*)

Causes the scanner to enter Start Scan Triggered Synchronized Scanning Mode.

While in Start Scan Triggered Synchronized Scanning Mode, the scanners will trigger on the appropriate edge of the Start Scan signal, depending on the presence of StartScanTriggerOnHigh.

May **not** be called while the scanner is in Synchronized Scanning Mode.

See also

jsExitSyncMode()

Parameters

jc The connection to the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.12.2.2 JCAM_DLL_API int STDCALL jsEnterEncoderSyncMode (JCONNECTION const *jc*)

Causes the scanner to enter Encoder Synchronized Scanning Mode.

May **not** be called while the scanner is in Synchronized Scanning Mode.

See also

`jsExitSyncMode()`

Parameters

jc The connection to the scanner.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.12.2.3 JCAM_DLL_API int STDCALL jsEnterTimeSyncMode (JCONNECTION const *jc*)

Causes the scanner to enter Time Synchronized Scanning Mode.

May **not** be called while the scanner is in Synchronized Scanning Mode.

See also

`jsExitSyncMode()`

Parameters

jc The connection to the scanner.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.12.2.4 JCAM_DLL_API int STDCALL jsStartPulseMaster (JCONNECTION const *jc*, int *pulseInterval*, int *pulseCount*)

Causes the scanner to ouput a pulse train on the Start Scan I/O. The period and number of pulses are configurable.

May **only** be called while the scanner is in Synchronized Scanning Mode.

See also

`jsStopPulses()`

Parameters

jc The connection to the scanner.

pulseInterval The period in microseconds of the pulse train. Between 1,000 and 5,000,000.

pulseCount The number of pulses to send before stopping. If zero pulses will be continuously sent. Max value is 511.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`, or other parameters are out of range.
`SCANNER_FAILURE` on all connection failures.

6.12.2.5 JCAM_DLL_API int STDCALL jsStopPulses (JCONNECTION const *jc*)

Causes the scanner to stop generating a pulse train on the Start Scan I/O.

May **only** be called while the scanner is in Synchronized Scanning Mode.

See also

`jsStartPulseMaster()`

Parameters

jc The connection to the scanner.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.12.2.6 JCAM_DLL_API int STDCALL jsHaltSyncMode (JCONNECTION const *jc*)

Causes the scanner to halt Time or Encoder Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

Any scans queued up in the scanner can still be read using `jsGetProfile()`. The scanner is still in Synchronized Scanning Mode after calling this function. A call to `jsExitSyncMode()` must follow in order to exit Synchronized Scanning Mode.

See also

`jsExitSyncMode()`
`jsEnterEncoderSyncMode()`
`jsEnterTimeSyncMode()`

Parameters

jc The connection to the scanner.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` is `NULL`.
`SCANNER_FAILURE` on all connection failures.

6.12.2.7 JCAM_DLL_API int STDCALL jsExitSyncMode (JCONNECTION const *jc*)

Causes the scanner to exit Time or Encoder Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

Any profiles queued in the scanner will be lost. This functionalso calls `jsCleanUpOutstandingRequests()` in order to fully reset scanning.

See also

`jsEnterEncoderSyncMode()`
`jsEnterTimeSyncMode()`
`jsHaltSyncMode()`
`jsGetNumberOfOutstandingRequests()`
`jsCleanUpOutstandingRequests()`

Parameters

jc The connection to the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.12.2.8 JCAM_DLL_API int STDCALL jsSendMultipleProfileRequest (JCONNECTION const *jc*, UINT32 const *nProfiles*)

Sends a request for up to nProfiles profiles to the scanner while in Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

No profiles are read from the connection during this call. In order to read the profiles, a call to jsReadMultipleProfiles() must follow.

If the parameters attempt to run the scanner faster than it can physically scan, then bit 0 in the flags field of the jsProfile will be set.

Increments the number of outstanding requests for this connection by one.

See also

jsReadMultipleProfiles()
jsGetNumberOfOutstandingRequests()
jsCleanUpOutstandingRequests()

Parameters

jc The connection to the scanner.

nProfiles The number of profiles to request from the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.12.2.9 JCAM_DLL_API int STDCALL jsReadMultipleProfiles (JCONNECTION const *jc*, jsProfile *const *profiles*, UINT32 *nProfiles*)

Reads up to nProfiles profiles from jc after a jsSendMultipleProfileRequest() call in Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

nProfiles must be equal to the number of profiles requested in the corresponding call to jsSendMultipleProfileRequest(), and profiles must point to an array of nProfiles jsProfile structs.

If the parameters attempt to run the scanner faster than it can physically scan, then bit 0 in the flags field of the jsProfile's will be set.

Decrements the number of outstanding requests for this connection by one.

See also

jsSendMultipleProfileRequest()
jsGetNumberOfOutstandingRequests()
jsCleanUpOutstandingRequests()

Parameters

jc The connection to the scanner.

profiles Pointer to an array of jsProfile's.

nProfiles The number of jsProfile's that profiles points to.

Returns

The number of valid profiles on success.
0 if no profiles are available.
SCANNER_FAILURE on all connection failures.
INVALID_PARAMETER if jc or profiles is NULL.
SOCKET_TIMEOUT if the read took longer than 5 seconds.

6.12.2.10 JCAM_DLL_API int STDCALL jsReadMultipleProfilesTimeout (JCONNECTION const *jc*, jsProfile *const *profiles*, UINT32 *cProfiles*, long *timeoutMilliseconds*)

Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

If the parameters attempt to run the scanner faster than it can physically scan, then bit 0 in the flags field of the jsProfile's will be set.

Parameters

jc The connection to the scanner.

profiles Pointer to an array of jsProfile's.

cProfiles The number of jsProfile's that profiles points to. Also how many profiles are requested from the scanner.

timeoutMilliseconds The maximum number of milliseconds that a read is allowed to take.

Returns

The number of valid profiles on success.
0 if no profiles are available.
SCANNER_FAILURE on all connection failures.
INVALID_PARAMETER if jc or profiles is NULL.
SOCKET_TIMEOUT if the read took longer than the specified timeout period.

6.12.2.11 JCAM_DLL_API int STDCALL jsGetMultipleProfiles (JCONNECTION const *jc*, jsProfile *const *const *profiles*, UINT32 const *cProfiles*)

Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

May **only** be called while the scanner is in Synchronized Scanning Mode.

If the parameters attempt to run the scanner faster than it can physically scan, then bit 0 in the flags field of the jsProfile's will be set.

Parameters

jc The connection to the scanner.

profiles Pointer to an array of jsProfile's.

cProfiles The number of jsProfile's that profiles points to. Also how many profiles are requested from the scanner.

Returns

The number of valid profiles on success.
0 if no profiles are available.
SCANNER_FAILURE on all connection failures.
INVALID_PARAMETER if jc or profiles is NULL.
SOCKET_TIMEOUT if the read took longer than 5 seconds.

6.12.2.12 JCAM_DLL_API int STDCALL jsGetProfile (JCONNECTION const *jc*, jsProfile *const *profile*)

Requests and reads a profile from the default laser.

May be called while the scanner is or is not in Synchronized Scanning Mode.

Parameters

jc The connection to read a profile from.

profile Where the profile is stored.

Returns

0 on success.
`INVALID_PARAMETER` if `jc` or `profile` is `NULL`.
`SCANNER_FAILURE` on all connection failures.
`PROFILE_UNAVAILABLE` if the scanner is in Synchronized Scanning Mode and there is no profile available.
`SOCKET_TIMEOUT` if the read took longer than 5 seconds.

6.12.2.13 JCAM_DLL_API size_t STDCALL jsGetNumberOfOutstandingRequests (JCONNECTION const *jc*)

Returns the number of unread requests from either `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()`.

Returns the number of unread requests generated by either `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()`. This function does not differentiate between requests generated by `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()`.

The number of profiles generated by `jsSendMultipleProfileRequest()` may be higher than the number of outstanding requests since more than one profile could be returned by one request.

Any further calls to `jsReadProfileN()` or `jsReadMultipleProfiles()` will block unless a new call to `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()` is made.

See also

`jsCleanUpOutstandingRequests()`
`jsSendProfileRequestN()`

jsReadProfileN()
jsSendMultipleProfileRequest()
jsReadMultipleProfiles()

Parameters

jc The connection to ask how many unread responses there are.

Returns

The number of unread responses from either jsSendProfileRequestN() or jsSendMultipleProfileRequest().

6.12.2.14 JCAM_DLL_API int STDCALL jsCleanUpOutstandingRequests (JCONNECTION const *jc*)

Reads and discards all the profiles in the TCP/IP stack generated by jsSendProfileRequestN() or jsSendMultipleProfileRequest().

Reads any outstanding responses to jsSendProfileRequestN() or jsSendMultipleProfileRequest() calls.

See also

jsGetNumberOfOutstandingRequests()
jsSendProfileRequestN()
jsReadProfileN()
jsSendMultipleProfileRequest()
jsReadMultipleProfiles()

Parameters

jc The connection to discard unread profiles from.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.12.2.15 JCAM_DLL_API int STDCALL jsSetEncoderValue (JCONNECTION const *jc*, UINT16 *newEncoderValue*)

Sets the scanner's internal encoder value.

May **not** be called while the scanner is in Synchronized Scanning Mode.

Parameters

jc The connection to the scanner.

newEncoderValue The new encoder value for the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.12.2.16 JCAM_DLL_API int STDCALL jsSetEncoderValue32 (JCONNECTION const *jc*, UINT32 *newEncoderValue*)

Sets the scanner's internal encoder value.

May **not** be called while the scanner is in Synchronized Scanning Mode.

Parameters

jc The connection to the scanner.

newEncoderValue The new encoder value for the scanner.

Returns

0 on success.
INVALID_PARAMETER if jc is NULL.
SCANNER_FAILURE on all connection failures.

6.13 Tutorial for Synchronized Scanning

You just want to get some data from the scanner quickly. Here's a small program that uses some imaginary data types and functions to demonstrate how to use the scanner in Encoder Synchronized Mode:

1. Call jsInitialize() if the Joescan API is statically linked.
2. Open a connection using jsOpenConnection().
3. Save the parameters using jsSendParameterFileToScanner() in case a scanner has been swapped out.

4. Put it into encoder mode using `jsEnterEncoderSyncMode()`.
5. Get profiles using `jsGetProfile()`. Be careful to not create a hard loop if no profiles are currently available.
6. End encoder mode using `jsExitSyncMode()`.
7. Close the connection using `jsCloseConnection()`.

```
#include <stdio.h>
#define JCAM_STATIC_LIB
#include "jcam_dll.h"

using namespace joescan;

int main(int argc, char **argv)
{
    if(jsInitialize() == FALSE)
    {
        printf("Could not initialize TCP\IP communications.\n");
        return -1;
    }

    char *ip = "192.168.1.205";
    if(argc > 1)
        ip = argv[1];

    JCONNECTION jc = jsOpenConnection(ip);
    if(jc == 0)
    {
        printf("Could not open connection to %s.\n", ip);
        return -1;
    }

    switch(jsSendParameterFileToScanner(jc, "param.dat"))
    {
        case SCANNER_FAILURE:
            printf("Lost connection.\n");
            jsCloseConnection(jc);
            jc = 0;
            return -1;
        case OPERATION_FAILURE:
            for(i = 0; i < jsGetNumberOfErrorMessages(jc); i++)
                printf("%s\n", jsGetErrorMessage(jc, i));
            jsCloseConnection(jc);
            jc = 0;
            return -1;
        default:
            printf("Sent parameters successfully.\n");
    }

    switch(jsEnterEncoderSyncMode(jc))
    {
        case 0:
            break;
        case INVALID_PARAMETER:
            printf("Invalid parameter to jsEnterEncoderSyncMode().\n");
```

```
            jsCloseConnection(jc);
            jc = 0;
            return -1;
        case SCANNER_FAILURE:
            printf("Scanner failure during jsEnterEncoderSyncMode().\n");
            jsCloseConnection(jc);
            jc = 0;
            return -1;
    }

    jsProfile profile;
    while(PLC.isLogVisible())
    {
        switch(jsGetProfile(jc, &profile))
        {
            case 0:
                logModel.addProfile(profile);
                break;
            case PROFILE_UNAVAILABLE:
                optimizer.earlyOptimize(logModel);
                break;
            case INVALID_PARAMETER:
                printf("Invalid parameter to jsGetProfile().\n");
                jsCloseConnection(jc);
                jc = 0;
                return -1;
            case SCANNER_FAILURE:
                printf("Scanner failure during jsGetProfile().\n");
                jsCloseConnection(jc);
                jc = 0;
                return -1;
        }
    }

    switch(jsExitSyncMode(jc))
    {
        case 0:
            break;
        case INVALID_PARAMETER:
            printf("Invalid parameter to jsExitSyncMode().\n");
            jsCloseConnection(jc);
            jc = 0;
            return -1;
        case SCANNER_FAILURE:
            printf("Scanner failure during jsExitSyncMode().\n");
            jsCloseConnection(jc);
            jc = 0;
            return -1;
    }

    jsCloseConnection(jc);
    jc = 0;
    optimizer.fullyOptimize(logModel);
    return 0;
}
```

7 Namespace Documentation

7.1 joescan Namespace Reference

7.1.1 Detailed Description

This C++ namespace that contains all the API's functions, except `jsInitialize()`. This is not applicable to straight C programs.

8 Class Documentation

8.1 tagImage Struct Reference

Image data from the scanner.

`#include <jcam_dll.h>`

Public Attributes

- INT32 exposureTime

  How many 10 microsecond units the camera was exposed.

- INT32 reserved1

  Currently unused.

- INT32 reserved2

  Currently unused.

- INT32 numberHorizontal

  The number of horizontal pixels.

- INT32 numberVertical

  The number of vertical pixels.

- UINT8 image [MAX_HORIZONTAL ∗MAX_VERTICAL]

  The image pixels.

8.1.1 Detailed Description

Image data from the scanner.

8.1.2 Member Data Documentation

8.1.2.1 INT32 tagImage::exposureTime

How many 10 microsecond units the camera was exposed.

8.1.2.2 INT32 tagImage::reserved1

Currently unused.

8.1.2.3 INT32 tagImage::reserved2

Currently unused.

8.1.2.4 INT32 tagImage::numberHorizontal

The number of horizontal pixels.

8.1.2.5 INT32 tagImage::numberVertical

The number of vertical pixels.

8.1.2.6 UINT8 tagImage::image[MAX_HORIZONTAL ∗MAX_VERTICAL]

The image pixels.

MAX_VERTICAL and MAX_HORIZONTAL are defined in jsConstants.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.2 tagOldCalibrationValue Struct Reference

Old position calibration parameters sent to the scanner.

`#include <jcam_dll.h>`

Public Attributes

- double x
  The X offset of the scanner's position.
- double y
  The Y offset of the scanner's position.
- double roll
  The scanner's rotation.
- char date [DATE_LENGTH]
  The date the calibration was sent. This can be an empty string.

8.2.1 Detailed Description

Old position calibration parameters sent to the scanner.

8.2.2 Member Data Documentation

8.2.2.1 double tagOldCalibrationValue::x

The X offset of the scanner's position.

8.2.2.2 double tagOldCalibrationValue::y

The Y offset of the scanner's position.

8.2.2.3 double tagOldCalibrationValue::roll

The scanner's rotation.

8.2.2.4 char tagOldCalibrationValue::date[DATE_LENGTH]

The date the calibration was sent. This can be an empty string.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.3 tagProfile Struct Reference

Profile data from the scanner.

`#include <jcam_dll.h>`

Public Attributes

- INT32 sequenceNumber

  Sequence number from Synchronized Scanning Mode.

- INT32 location

  The encoder value when when the scanner created the profile.

- INT32 sendLocation

  The encoder value when the scanner sent the profile.

- INT32 laserOnTime

  The number of 10 microsecond units the laser was on.

- INT32 timeInHead

  The time since the scanner was turned on.

- INT32 inputs

  The cable input to the scanner.

- INT32 flags

  Information from the scanner.

- INT32 laserIndex

  Currently unused.

- INT32 reserved2

  Which laser this profile is from.

- INT32 numberPoints

  The number of valid coordinate points in this profile.

- ProfileDataPoint data [MAX_VERTICAL]

  Coordinates in 1000ths of an inch; i.e. 2000 is 2 inches. If units are set to metric units are micrometers, and 2000 in 2mm.

8.3.1 Detailed Description

Profile data from the scanner.

8.3.2 Member Data Documentation

8.3.2.1 INT32 tagProfile::sequenceNumber

Sequence number from Synchronized Scanning Mode.

Sequential `jsProfile's` during Synchronized Scanning Mode should have sequential `sequenceNumber's`. The counter is reset to 0 every time Synchronized Scanning Mode is restarted.

8.3.2.2 INT32 tagProfile::location

The encoder value when when the scanner created the profile.

8.3.2.3 INT32 tagProfile::sendLocation

The encoder value when the scanner sent the profile.

8.3.2.4 INT32 tagProfile::laserOnTime

The number of 10 microsecond units the laser was on.

8.3.2.5 INT32 tagProfile::timeInHead

The time since the scanner was turned on.

It is not the time of day, nor the current date.

8.3.2.6 INT32 tagProfile::inputs

The cable input to the scanner.

This includes the Cable ID, the encoder signals, and the Start Scan signal.

8.3.2.7 INT32 tagProfile::flags

Information from the scanner.

If bit 0 is set, then Synchronized Scanning Mode scanning is trying to run faster than the scanner is capable and some data will be lost on one end of the laser line.

If bit 1 is set, this indicates that an internal error has occured in the scanner. Contact a JoeScan support engineer if you encounter this flag.

The remaining bits are reserved and values may change in future DLLs. So you should mask them to be ensure future firmware revision don't break your software.

8.3.2.8 INT32 tagProfile::laserIndex

Currently unused.

8.3.2.9 INT32 tagProfile::reserved2

Which laser this profile is from.

8.3.2.10 INT32 tagProfile::numberPoints

The number of valid coordinate points in this profile.

8.3.2.11 ProfileDataPoint tagProfile::data[MAX_VERTICAL]

Coordinates in 1000ths of an inch; i.e. 2000 is 2 inches. If units are set to metric units are micrometers, and 2000 in 2mm.

MAX_VERTICAL is defined in jsConstants.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.4 tagProfileDataPoint Struct Reference

Represents a coordinate and its associated pixel brightness.

```
#include <jcam_dll.h>
```

Public Attributes

- INT32 x

  The x coordinate of the data point in 1000ths of an inch.

- INT32 y

  The y coordinate of the data point in 1000ths of an inch.

- INT32 brightness

  The intensity of the pixel where the laser was detected.

8.4.1 Detailed Description

Represents a coordinate and its associated pixel brightness.

8.4.2 Member Data Documentation

8.4.2.1 INT32 tagProfileDataPoint::x

The x coordinate of the data point in 1000ths of an inch.

8.4.2.2 INT32 tagProfileDataPoint::y

The y coordinate of the data point in 1000ths of an inch.

8.4.2.3 INT32 tagProfileDataPoint::brightness

The intensity of the pixel where the laser was detected.

The value is the sum of 7 pixels, each with 10 bits of resolution. Consequently, the maximum value is $7 * 2^{10} = 7168$. If you want to display this value in a traditional 255 level greyscale, you will have to map the brightness value to a 255 level greyscale on your own.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.5 tagResponsePacket Struct Reference

Responses from scanners on the network.

`#include <jcam_dll.h>`

Public Attributes

- UINT8 macAddress [6]

  The Ethernet hardware MAC address of the scanner.

- UINT8 currentIpSetup

  The current addressing technique the scanner is using.

- UINT32 ipAddress

  *The IP Address of the responding scanner, in **network byte order**.*

- UINT32 serialNumber

  The serial number of the responding scanner.

- UINT8 cableId

  The cable ID of the responding scanner.

- UINT8 status

The status of the responding scanner.

- UINT32 build

  The firmware revision number of the responding scanner.

- INT8 options [OPTIONS_SIZE]

  Internal use. Don't change the contents.

8.5.1 Detailed Description

Responses from scanners on the network.

8.5.2 Member Data Documentation

8.5.2.1 UINT8 tagResponsePacket::macAddress[6]

The Ethernet hardware MAC address of the scanner.

8.5.2.2 UINT8 tagResponsePacket::currentIpSetup

The current addressing technique the scanner is using.

`IPS_STATIC` Static IP Address for the scanner.

`IPS_BY_ID` IP Address is determined by the cable ID.

`IPS_BY_DHCP` IP Address is assigned with DHCP.

8.5.2.3 UINT32 tagResponsePacket::ipAddress

The IP Address of the responding scanner, in **network byte order**.

8.5.2.4 UINT32 tagResponsePacket::serialNumber

The serial number of the responding scanner.

8.5.2.5 UINT8 tagResponsePacket::cableId

The cable ID of the responding scanner.

8.5.2.6 UINT8 tagResponsePacket::status

The status of the responding scanner.

8.5.2.7 UINT32 tagResponsePacket::build

The firmware revision number of the responding scanner.

8.5.2.8 INT8 tagResponsePacket::options[OPTIONS_SIZE]

Internal use. Don't change the contents.

`OPTIONS_SIZE` is defined in `jsConstants`.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.6 tagScan Struct Reference

Scan data from the scanner.

`#include <jcam_dll.h>`

Public Attributes

- INT32 location

  The encoder value when the scan was taken.

- INT32 sendLocation

  The encoder value when the scan was sent.

- INT32 laserOnTime

The number of 10 microseconds units the laser was on.

- INT32 timeInHead

  The time since the scanner was turned on.

- INT32 inputs

  The cable input to the scanner.

- INT32 flags

  Information from the scanner.

- INT32 reserved1

  Currently unused.

- INT32 reserved2

  Currently unused.

- INT32 numberPoints

  The number of valid pixel points in this profile.

- ScanDataPoint data [MAX_VERTICAL]

  Pixel data points.

8.6.1 Detailed Description

Scan data from the scanner.

8.6.2 Member Data Documentation

8.6.2.1 INT32 tagScan::location

The encoder value when the scan was taken.

8.6.2.2 INT32 tagScan::sendLocation

The encoder value when the scan was sent.

8.6.2.3 INT32 tagScan::laserOnTime

The number of 10 microseconds units the laser was on.

8.6.2.4 INT32 tagScan::timeInHead

The time since the scanner was turned on.

It is not the time of day, nor the current date.

8.6.2.5 INT32 tagScan::inputs

The cable input to the scanner.

This includes the Cable ID, the encoder signals, and the Start Scan signal.

8.6.2.6 INT32 tagScan::flags

Information from the scanner.

If bit 0 is set, then Synchronized Scanning Mode scanning is trying to run faster than the scanner is capable. The rest of the bits are unused and should be 0.

8.6.2.7 INT32 tagScan::reserved1

Currently unused.

8.6.2.8 INT32 tagScan::reserved2

Currently unused.

8.6.2.9 INT32 tagScan::numberPoints

The number of valid pixel points in this profile.

8.6.2.10 ScanDataPoint tagScan::data[MAX_VERTICAL]

Pixel data points.

MAX_VERTICAL is defined in jsConstants.

The documentation for this struct was generated from the following file:

- jcam_dll.h

8.7 tagScanDataPoint Struct Reference

Represents a subpixel point from a scanner and its brightness.

`#include <jcam_dll.h>`

Public Attributes

- INT32 data

  Which subpixel the laser was centered at.

- INT32 status

  A simple representation of the intensity of the pixel.

- INT32 brightness

  The actual intensity of the pixel where the laser was detected.

8.7.1 Detailed Description

Represents a subpixel point from a scanner and its brightness.

8.7.2 Member Data Documentation

8.7.2.1 INT32 tagScanDataPoint::data

Which subpixel the laser was centered at.

Range is from 0 to 32,400 because there are 100 subpixels per pixel and 324 pixels.

8.7.2.2 INT32 tagScanDataPoint::status

A simple representation of the intensity of the pixel.

0 means invalid.

1 means normal.

2 means bright.

3 means very bright.

8.7.2.3 INT32 tagScanDataPoint::brightness

The actual intensity of the pixel where the laser was detected.

The value is the sum of 7 pixels, each with 10 bits of resolution. Consequently, the maximum value is $7 * 2^{10} = 7168$. If you want to display this value in a traditional 255 level greyscale. you will have to map the brightness value to a 255 level greyscale on your own.

The documentation for this struct was generated from the following file:

- jcam_dll.h

9 File Documentation

9.1 jcam_dll.h File Reference

Data structures used for communicating with the JS20s.

```
#include <stddef.h>
#include <atlsafe.h>
#include <windef.h>
```

Classes

- struct tagProfileDataPoint

  Represents a coordinate and its associated pixel brightness.

- struct tagScanDataPoint

  Represents a subpixel point from a scanner and its brightness.

- struct tagProfile

  Profile data from the scanner.

- struct tagImage

  Image data from the scanner.

- struct tagScan

  Scan data from the scanner.

- struct tagResponsePacket

  Responses from scanners on the network.

- struct tagOldCalibrationValue

  Old position calibration parameters sent to the scanner.

Typedefs

- typedef signed char INT8

  Signed 8 bit integer.

- typedef signed short INT16

  Signed 16 bit integer.

- typedef signed int INT32

  Signed 32 bit integer.

- typedef unsigned char UINT8

  Unsigned 8 bit integer.

- typedef unsigned short UINT16

  Unsigned 16 bit integer.

- typedef unsigned int UINT32

  Unsigned 32 bit integer.

- typedef enum jsLaserIndexTag jsLaserIndex

  A type for specifying which laser to use.

- typedef void * JCONNECTION

  An opaque handle for connections to scanners.

- typedef struct tagProfileDataPoint ProfileDataPoint
  Represents a coordinate and its associated pixel brightness.
- typedef struct tagScanDataPoint ScanDataPoint
  Represents a subpixel point from a scanner and its brightness.
- typedef struct tagProfile jsProfile
  Profile data from the scanner.
- typedef struct tagImage jsImage
  Image data from the scanner.
- typedef struct tagScan jsScan
  Scan data from the scanner.
- typedef struct tagResponsePacket jsResponsePacket
  Responses from scanners on the network.
- typedef struct tagOldCalibrationValue jsOldCalibrationValue
  Old position calibration parameters sent to the scanner.

Enumerations

- enum jsConstants {
  OPERATION_FAILURE = 21,
  SCANNER_FAILURE = -1,
  INVALID_PARAMETER = -2,
  PROFILE_UNAVAILABLE = -3,
  SOCKET_TIMEOUT = -4,
  SYNC_MODE_OVERRUN = 0x00000001,
  MAX_HORIZONTAL = 324,
  MAX_VERTICAL = 243,
  IPS_STATIC = 0,
  IPS_BY_ID = 1,
  IPS_BY_DHCP = 2,
  STATUS_NORMAL = 0,
  STATUS_ACTIVE = 1,

OPTIONS_SIZE = 512,

DATE_LENGTH = 32 }

Constants that avoid the traditional problems of `#define` constants.

- enum jsLaserIndexTag {

  LASER0 = 0,

  LASER1 = 1,

  LASER2 = 2,

  LASER3 = 3,

  LASER4 = 4 }

  A type for specifying which laser to use.

Functions

- JCAM_DLL_API int STDCALL jsGetJcamDllMajorVersionNumber ()

  Returns the major revision number of the DLL.

- JCAM_DLL_API int STDCALL jsGetJcamDllMinorVersionNumber ()

  Returns the minor revision number of the DLL.

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnection (char const *const host)

  Opens a connection to the specified host.

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionInt (int const host)

  Opens a connection to the specified host.

- JCAM_DLL_API JCONNECTION STDCALL jsOpenConnectionBase (char const *const host, UINT32 const cableID)

  Opens a connection to the specified scanner using Base IP + Cable ID addressing.

- JCAM_DLL_API int STDCALL jsCloseConnection (JCONNECTION const jc)

  Closes a `JCONNECTION`.

- JCAM_DLL_API int STDCALL jsSendProfileRequestN (JCONNECTION const jc, jsLaserIndex const laserIndex)

  Sends a request for a profile from the specified laser.

- JCAM_DLL_API int STDCALL jsReadProfileN (JCONNECTION const jc, jsProfile *const profile)

  Reads a profile from the connection.

- JCAM_DLL_API int STDCALL jsGetProfileN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsProfile *const profile)

  Requests and reads a profile from the specified laser.

- JCAM_DLL_API int STDCALL jsGetProfileFromAllLasers (JCONNECTION const jc, jsProfile *const profiles, size_t const cProfiles)

  Requests and reads a profile from each laser the scanner has.

- JCAM_DLL_API int STDCALL jsGetImage (JCONNECTION const jc, jsImage *const image)

  Requests and reads an image.

- JCAM_DLL_API int STDCALL jsGetImageN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsImage *const image)

  Requests and reads an image with the specified laser on.

- JCAM_DLL_API int STDCALL jsGetScan (JCONNECTION const jc, jsScan *const scan)

  Requests and reads a scan.

- JCAM_DLL_API int STDCALL jsGetScanN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsScan *const scan)

  Requests and reads a scan with the specified laser on.

- JCAM_DLL_API int STDCALL jsGetImageScan (JCONNECTION const jc, jsImage *const image, jsScan *const scan)

  Requests and reads an image and scan.

- JCAM_DLL_API int STDCALL jsGetImageScanN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsImage *const image, jsScan *const scan)

  Requests and reads an image and scan with the specified laser on.

- JCAM_DLL_API int STDCALL jsEnterStartScanTriggeredMode (JCONNECTION const jc)

  Causes the scanner to enter Start Scan Triggered Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsEnterEncoderSyncMode (JCONNECTION const jc)

  Causes the scanner to enter Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsEnterTimeSyncMode (JCONNECTION const jc)

  Causes the scanner to enter Time Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsStartPulseMaster (JCONNECTION const jc, int pulseInterval, int pulseCount)

  Causes the scanner to ouput a pulse train on the Start Scan I/O. The period and number of pulses are configurable.

- JCAM_DLL_API int STDCALL jsStopPulses (JCONNECTION const jc)

  Causes the scanner to stop generating a pulse train on the Start Scan I/O.

- JCAM_DLL_API int STDCALL jsHaltSyncMode (JCONNECTION const jc)

  Causes the scanner to halt Time or Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsExitSyncMode (JCONNECTION const jc)

  Causes the scanner to exit Time or Encoder Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsSendMultipleProfileRequest (JCONNECTION const jc, UINT32 const nProfiles)

  Sends a request for up to `nProfiles` profiles to the scanner while in Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsReadMultipleProfiles (JCONNECTION const jc, jsProfile *const profiles, UINT32 nProfiles)

  Reads up to `nProfiles` profiles from `jc` after a `jsSendMultipleProfileRequest()` call in Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsReadMultipleProfilesTimeout (JCONNECTION const jc, jsProfile *const profiles, UINT32 cProfiles, long timeoutMilliseconds)

  Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsGetMultipleProfiles (JCONNECTION const jc, jsProfile *const profiles, UINT32 const cProfiles)

  Requests and reads up to n profiles from the scanner while in either Synchronized Scanning Mode.

- JCAM_DLL_API int STDCALL jsGetProfile (JCONNECTION const jc, jsProfile *const profile)

  Requests and reads a profile from the default laser.

- JCAM_DLL_API size_t STDCALL jsGetNumberOfOutstandingRequests (JCONNECTION const jc)

  Returns the number of unread requests from either `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()`.

- JCAM_DLL_API int STDCALL jsCleanUpOutstandingRequests (JCONNECTION const jc)

  Reads and discards all the profiles in the TCP/IP stack generated by `jsSendProfileRequestN()` or `jsSendMultipleProfileRequest()`.

- JCAM_DLL_API int STDCALL jsSetEncoderValue (JCONNECTION const jc, UINT16 newEncoderValue)

  Sets the scanner's internal encoder value.

- JCAM_DLL_API int STDCALL jsSetEncoderValue32 (JCONNECTION const jc, UINT32 newEncoderValue)

  Sets the scanner's internal encoder value.

- JCAM_DLL_API int STDCALL jsSendParameterFileToScanner (JCONNECTION const jc, char const *const filename)

  Sends a parameter file to the scanhead.

- JCAM_DLL_API int STDCALL jsSendParametersToScanner (JCONNECTION const jc, char const *const parameters)

  Sends parameters to the scanhead.

- JCAM_DLL_API int STDCALL jsGetParameterFileFromScanner (JCONNECTION const jc, char const *const filename)

  Reads the current parameter file from the scanhead and writes them to the specified file.

- JCAM_DLL_API size_t STDCALL jsGetNumberOfErrorMessages (JCONNECTION const jc)

  Returns the number of error messages after a `jsSendParameterFileToScanner()` call returned `OPERATION_FAILURE`.

- JCAM_DLL_API char const *const STDCALL jsGetErrorMessage (JCONNECTION const jc, size_t i)

  Returns the specified error message.

- JCAM_DLL_API int STDCALL jsReadOldPositionCalibrationsN (JCONNECTION const jc, jsLaserIndex const laserIndex, jsOldCalibrationValue oldCalibrations[ ], UINT32 nCalibrations)

Reads the specified number of old position calibrations for the selected laser from the scanner.

- JCAM_DLL_API int STDCALL jsReadPositionCalibrationN (JCONNECTION const jc, jsLaserIndex const laserIndex, double *xOffset, double *yOffset, double *roll)

  Reads the current position calibration for the selected laser from the scanner.

- JCAM_DLL_API int STDCALL jsSendPositionCalibrationN (JCONNECTION const jc, jsLaserIndex const laserIndex, double const xOffset, double const yOffset, double const roll)

  Sends a position calibration for the selected laser to the scanner.

- JCAM_DLL_API int STDCALL jsGetScannerStatusFromScanner (JCONNECTION const jc)

  Reads status from the scanner into a data structure inside the opaque JCONNECTION.

- JCAM_DLL_API int STDCALL jsGetScannerStatusValue (JCONNECTION const jc, size_t i, int *value)

  Gets the specified status value from the data structure inside the JCONNECTION.

- JCAM_DLL_API int STDCALL jsGetStatusDescriptionFromScanner (JCONNECTION const jc, UINT32 i, char *const description, UINT32 description_length)

  Reads a descriptive text string from the scanner for the particular status index.

- JCAM_DLL_API int STDCALL jsGetImagePixel (jsImage const *const image, size_t const x, size_t const y)

  Returns a pixel value from an image.

- JCAM_DLL_API int STDCALL jsFindAllScanners (jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover all scanners on the network.

- JCAM_DLL_API int STDCALL jsFindScannerByCableId (int const cableId, jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover the scanner(s) on the network with the specified Cable ID.

- JCAM_DLL_API int STDCALL jsFindScannerBySerialNumber (int const serialNumber, jsResponsePacket responses[ ], const size_t nResponses)

  Attempts to discover the scanner on the network with the specified serial number.

- JCAM_DLL_API int STDCALL jsSetStaticIpInt (UINT32 const serialNumber, UINT32 const staticIpAddress, UINT32 const netmask)

  Sets a scanner's static IP address.

- JCAM_DLL_API int STDCALL jsSetStaticIpChar (UINT32 const serialNumber, char const *const staticIpAddress, char const *const netmask)

  Sets a scanner's static IP address.

- JCAM_DLL_API int STDCALL jsSetBaseIpInt (UINT32 const serialNumber, UINT32 const baseIpAddress, UINT32 const netmask)

  Sets a scanner's base IP address.

- JCAM_DLL_API int STDCALL jsSetBaseIpChar (UINT32 const serialNumber, char const *const baseIpAddress, char const *const netmask)

  Sets a scanner's base IP address.

- JCAM_DLL_API int STDCALL jsSetCableId (UINT32 const serialNumber, UINT32 const cableId)

  Overrides the cable ID physically wired into a scanner.

- JCAM_DLL_API int STDCALL jsClearCableId (UINT32 const serialNumber)

  Reverts to the cable ID physically wired into a scanner.

- BOOL jsInitialize ()

  Initializes TCP/IP communications.

- void jsCleanup ()

  Gracefully ends TCP/IP communications.

9.1.1 Detailed Description

Data structures used for communicating with the JS20s.

9.1.2 Typedef Documentation

9.1.2.1 typedef signed char INT8

Signed 8 bit integer.

9.1.2.2 typedef signed short INT16

Signed 16 bit integer.

9.1.2.3 typedef signed int INT32

Signed 32 bit integer.

9.1.2.4 typedef unsigned char UINT8

Unsigned 8 bit integer.

9.1.2.5 typedef unsigned short UINT16

Unsigned 16 bit integer.

9.1.2.6 typedef unsigned int UINT32

Unsigned 32 bit integer.

9.1.2.7 typedef enum jsLaserIndexTag jsLaserIndex

A type for specifying which laser to use.

If an invalid laser is specified, the connection will be closed.

9.1.2.8 typedef void* JCONNECTION

An opaque handle for connections to scanners.

JCONNECTION's are full-duplex request/reply connections. Functions have been provided that handle all the request/reply semantics.

9.1.2.9 typedef struct tagProfileDataPoint ProfileDataPoint

Represents a coordinate and its associated pixel brightness.

9.1.2.10 typedef struct tagScanDataPoint ScanDataPoint

Represents a subpixel point from a scanner and its brightness.

9.1.2.11 typedef struct tagProfile jsProfile

Profile data from the scanner.

9.1.2.12 typedef struct tagImage jsImage

Image data from the scanner.

9.1.2.13 typedef struct tagScan jsScan

Scan data from the scanner.

9.1.2.14 typedef struct tagResponsePacket jsResponsePacket

Responses from scanners on the network.

9.1.2.15 typedef struct tagOldCalibrationValue jsOldCalibrationValue

Old position calibration parameters sent to the scanner.

9.1.3 Enumeration Type Documentation

9.1.3.1 enum jsConstants

Constants that avoid the traditional problems of `#define` constants.

Enumerator:

OPERATION_FAILURE An operation was unable to complete.

SCANNER_FAILURE The connection to the scanner is invalid.

INVALID_PARAMETER A parameter to a function was `NULL` or out of range.

PROFILE_UNAVAILABLE A scanner in Synchronized Scanning Mode has no available profiles.

SOCKET_TIMEOUT A connection to a scanner timed out.

SYNC_MODE_OVERRUN If this bit is set in a `jsProfile`'s flags field, then Synchronized Scanning Mode is running the scanner too fast.

MAX_HORIZONTAL CCD sensor dimensions.

MAX_VERTICAL CCD sensor dimensions.

IPS_STATIC Static IP Address for a scanner.

IPS_BY_ID IP Address is determined by the cable ID.

IPS_BY_DHCP IP Address is assigned with DHCP.

STATUS_NORMAL The scanner is fine.

STATUS_ACTIVE The scanner is scanning.

OPTIONS_SIZE The number of bytes in a `jsResponsePacket`'s `options` field.

DATE_LENGTH The maximum length of the `date` field in a `jsOldCalibrationValue`.

9.1.3.2 enum jsLaserIndexTag

A type for specifying which laser to use.

If an invalid laser is specified, the connection will be closed.

Enumerator:

LASER0 Furthest from the camera.

LASER1 Closer to the camera than LASER0.

LASER2 Closer to the camera than LASER1.

LASER3 Closer to the camera than LASER2.

LASER4 Closer to the camera than LASER3.

Index